

MySQL 5.5 Reference Manual
Including MySQL Cluster NDB 7.2 Reference Guide

MySQL 5.5 Reference Manual: Including MySQL Cluster NDB 7.2 Reference Guide

Abstract

This is the MySQL™ Reference Manual. It documents MySQL 5.5 through 5.5.31, as well as MySQL Cluster releases based on version 7.2 of [NDBCLUSTER](#) through 5.5.29-ndb-7.2.10.

MySQL 5.5 features. This manual describes features that are not included in every edition of MySQL 5.5; such features may not be included in the edition of MySQL 5.5 licensed to you. If you have any questions about the features included in your edition of MySQL 5.5, refer to your MySQL 5.5 license agreement or contact your Oracle sales representative.

For release notes detailing the changes in each release, see the [MySQL 5.5 Release Notes](#).

For legal information, see the [Legal Notices](#).

Document generated on: 2013-01-20 (revision: 33984)

General	Administrators	MySQL Enterprise	Developers & Functionality	Connectors & APIs	HA/Scalability
Tutorial	Installation & Upgrades	MySQL Enterprise Edition	MySQL Workbench	Connectors and APIs	» HA/Scalability Guide
Server Administration	Installation Using Unbreakable Linux Network (ULN)	MySQL Enterprise Monitor	Globalization	Connector/J	MySQL Cluster NDB 7.2 (GA)
SQL Syntax	» MySQL Installer	MySQL Enterprise Backup	Optimization	Connector/ODBC	MySQL and DRBD
Storage Engines	» Security	MySQL Enterprise Security	Functions and Operators	Connector/Net	memcached
Server Option / Variable Reference	» Startup / Shutdown	MySQL Enterprise Audit	Views and Stored Programs	Connector/Python	» memcached with MySQL Cluster
» Release Notes	» Backup and Recovery Overview	MySQL Thread Pool	Partitioning	PHP	MySQL and Virtualization
» MySQL Version Reference	» MySQL Utilities		Precision Math	C API	MySQL Proxy
FAQs	» Linux/Unix Platform Guide		Information Schema	Connector/C	Replication
	» Windows Platform Guide		Performance Schema	Connector/C++	Semisynchronous Replication
	» Mac OS X Platform Guide		Spatial Extensions	» MySQL for Excel	
	» Solaris Platform Guide		Restrictions and Limitations	» MySQL Cluster API Developer	
	» Building from Source				

Table of Contents

Preface and Legal Notices	xxv
1. General Information	1
1.1. About This Manual	2
1.2. Typographical and Syntax Conventions	3
1.3. Overview of the MySQL Database Management System	4
1.3.1. What is MySQL?	4
1.3.2. The Main Features of MySQL	6
1.3.3. History of MySQL	9
1.4. What Is New in MySQL 5.5	9
1.4.1. Scalability Improvements	13
1.4.2. InnoDB I/O Subsystem Changes	14
1.4.3. Diagnostic and Monitoring Capabilities	14
1.4.4. Enhanced Solaris Support	15
1.5. MySQL Development History	16
1.6. MySQL Information Sources	16
1.6.1. MySQL Mailing Lists	16
1.6.2. MySQL Community Support at the MySQL Forums	19
1.6.3. MySQL Community Support on Internet Relay Chat (IRC)	19
1.6.4. MySQL Enterprise	20
1.7. How to Report Bugs or Problems	20
1.8. MySQL Standards Compliance	24
1.8.1. What Standards MySQL Follows	25
1.8.2. Selecting SQL Modes	25
1.8.3. Running MySQL in ANSI Mode	25
1.8.4. MySQL Extensions to Standard SQL	26
1.8.5. MySQL Differences from Standard SQL	29
1.8.6. How MySQL Deals with Constraints	33
1.9. Credits	36
1.9.1. Contributors to MySQL	36
1.9.2. Documenters and translators	41
1.9.3. Packages that support MySQL	42
1.9.4. Tools that were used to create MySQL	43
1.9.5. Supporters of MySQL	43
2. Installing and Upgrading MySQL	45
2.1. General Installation Guidance	47
2.1.1. Operating Systems Supported by MySQL Community Server	48
2.1.2. Choosing Which MySQL Distribution to Install	49
2.1.3. How to Get MySQL	52
2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG	52
2.1.5. Installation Layouts	55
2.1.6. Compiler-Specific Build Characteristics	56
2.2. Installing MySQL from Generic Binaries on Unix/Linux	56
2.3. Installing MySQL on Microsoft Windows	58
2.3.1. MySQL Installation Layout on Microsoft Windows	60
2.3.2. Choosing An Installation Package	61
2.3.3. Installing MySQL on Microsoft Windows Using MySQL Installer	61
2.3.4. Installing MySQL on Microsoft Windows Using an MSI Package	78
2.3.5. MySQL Notifier for Microsoft Windows	85
2.3.6. MySQL Server Instance Configuration Wizard	89
2.3.7. Installing MySQL on Microsoft Windows Using a noinstall Zip Archive	105
2.3.8. Troubleshooting a Microsoft Windows MySQL Server Installation	113

2.3.9. Upgrading MySQL on Windows	114
2.3.10. Windows Postinstallation Procedures	115
2.4. Installing MySQL on Mac OS X	117
2.4.1. General Notes on Installing MySQL on Mac OS X	117
2.4.2. Installing MySQL on Mac OS X Using Native Packages	119
2.4.3. Installing the MySQL Startup Item	121
2.4.4. Installing and Using the MySQL Preference Pane	124
2.4.5. Using the Bundled MySQL on Mac OS X Server	126
2.5. Installing MySQL on Linux	127
2.5.1. Installing MySQL from RPM Packages on Linux	127
2.5.2. Installing MySQL on Linux using Native Package Manager	131
2.6. Installing MySQL Using Unbreakable Linux Network (ULN)	135
2.7. Installing MySQL on Solaris and OpenSolaris	135
2.7.1. Installing MySQL on Solaris using a Solaris <code>PKG</code>	136
2.7.2. Installing MySQL on OpenSolaris using IPS	137
2.8. Installing MySQL on HP-UX	138
2.8.1. General Notes on Installing MySQL on HP-UX	138
2.8.2. Installing MySQL on HP-UX using DEPOT	138
2.9. Installing MySQL on FreeBSD	139
2.10. Installing MySQL from Source	140
2.10.1. MySQL Layout for Source Installation	142
2.10.2. Installing MySQL from a Standard Source Distribution	142
2.10.3. Installing MySQL from a Development Source Tree	146
2.10.4. MySQL Source-Configuration Options	147
2.10.5. Dealing with Problems Compiling MySQL	158
2.10.6. MySQL Configuration and Third-Party Tools	159
2.11. Postinstallation Setup and Testing	159
2.11.1. Unix Postinstallation Procedures	160
2.11.2. Securing the Initial MySQL Accounts	171
2.12. Upgrading or Downgrading MySQL	175
2.12.1. Upgrading MySQL	175
2.12.2. Downgrading MySQL	187
2.12.3. Checking Whether Tables or Indexes Must Be Rebuilt	188
2.12.4. Rebuilding or Repairing Tables or Indexes	190
2.12.5. Copying MySQL Databases to Another Machine	191
2.13. Environment Variables	192
2.14. Perl Installation Notes	194
2.14.1. Installing Perl on Unix	194
2.14.2. Installing ActiveState Perl on Windows	195
2.14.3. Problems Using the Perl <code>DBI/DBD</code> Interface	195
3. Tutorial	197
3.1. Connecting to and Disconnecting from the Server	197
3.2. Entering Queries	198
3.3. Creating and Using a Database	201
3.3.1. Creating and Selecting a Database	202
3.3.2. Creating a Table	203
3.3.3. Loading Data into a Table	204
3.3.4. Retrieving Information from a Table	206
3.4. Getting Information About Databases and Tables	219
3.5. Using <code>mysql</code> in Batch Mode	220
3.6. Examples of Common Queries	221
3.6.1. The Maximum Value for a Column	222
3.6.2. The Row Holding the Maximum of a Certain Column	222
3.6.3. Maximum of Column per Group	223

3.6.4. The Rows Holding the Group-wise Maximum of a Certain Column	223
3.6.5. Using User-Defined Variables	224
3.6.6. Using Foreign Keys	224
3.6.7. Searching on Two Keys	226
3.6.8. Calculating Visits Per Day	226
3.6.9. Using <code>AUTO_INCREMENT</code>	226
3.7. Using MySQL with Apache	229
4. MySQL Programs	231
4.1. Overview of MySQL Programs	232
4.2. Using MySQL Programs	236
4.2.1. Invoking MySQL Programs	236
4.2.2. Connecting to the MySQL Server	237
4.2.3. Specifying Program Options	241
4.2.4. Setting Environment Variables	253
4.3. MySQL Server and Server-Startup Programs	254
4.3.1. <code>mysqld</code> — The MySQL Server	254
4.3.2. <code>mysqld_safe</code> — MySQL Server Startup Script	254
4.3.3. <code>mysql.server</code> — MySQL Server Startup Script	259
4.3.4. <code>mysqld_multi</code> — Manage Multiple MySQL Servers	260
4.4. MySQL Installation-Related Programs	264
4.4.1. <code>comp_err</code> — Compile MySQL Error Message File	264
4.4.2. <code>mysqlbug</code> — Generate Bug Report	265
4.4.3. <code>mysql_install_db</code> — Initialize MySQL Data Directory	265
4.4.4. <code>mysql_plugin</code> — Configure MySQL Server Plugins	267
4.4.5. <code>mysql_secure_installation</code> — Improve MySQL Installation Security	269
4.4.6. <code>mysql_tzinfo_to_sql</code> — Load the Time Zone Tables	269
4.4.7. <code>mysql_upgrade</code> — Check and Upgrade MySQL Tables	270
4.5. MySQL Client Programs	272
4.5.1. <code>mysql</code> — The MySQL Command-Line Tool	272
4.5.2. <code>mysqladmin</code> — Client for Administering a MySQL Server	293
4.5.3. <code>mysqlcheck</code> — A Table Maintenance Program	300
4.5.4. <code>mysqldump</code> — A Database Backup Program	307
4.5.5. <code>mysqlimport</code> — A Data Import Program	325
4.5.6. <code>mysqlshow</code> — Display Database, Table, and Column Information	330
4.5.7. <code>mysqlslap</code> — Load Emulation Client	334
4.6. MySQL Administrative and Utility Programs	341
4.6.1. <code>innochecksum</code> — Offline InnoDB File Checksum Utility	341
4.6.2. <code>myisam_ftdump</code> — Display Full-Text Index information	342
4.6.3. <code>myisamchk</code> — MyISAM Table-Maintenance Utility	343
4.6.4. <code>myisamlog</code> — Display MyISAM Log File Contents	360
4.6.5. <code>myisampack</code> — Generate Compressed, Read-Only MyISAM Tables	361
4.6.6. <code>mysqlaccess</code> — Client for Checking Access Privileges	367
4.6.7. <code>mysqlbinlog</code> — Utility for Processing Binary Log Files	370
4.6.8. <code>mysqldumpslow</code> — Summarize Slow Query Log Files	384
4.6.9. <code>mysqlhotcopy</code> — A Database Backup Program	386
4.6.10. <code>mysql_convert_table_format</code> — Convert Tables to Use a Given Storage Engine	389
4.6.11. <code>mysql_find_rows</code> — Extract SQL Statements from Files	390
4.6.12. <code>mysql_fix_extensions</code> — Normalize Table File Name Extensions	391
4.6.13. <code>mysql_setpermission</code> — Interactively Set Permissions in Grant Tables	391
4.6.14. <code>mysql_waitpid</code> — Kill Process and Wait for Its Termination	392
4.6.15. <code>mysql_zap</code> — Kill Processes That Match a Pattern	393
4.7. MySQL Program Development Utilities	393
4.7.1. <code>mysql2mysql</code> — Convert mSQL Programs for Use with MySQL	394

4.7.2. <code>mysql_config</code> — Get Compile Options for Compiling Clients	394
4.7.3. <code>my_print_defaults</code> — Display Options from Option Files	395
4.7.4. <code>resolve_stack_dump</code> — Resolve Numeric Stack Trace Dump to Symbols	396
4.8. Miscellaneous Programs	397
4.8.1. <code>pererror</code> — Explain Error Codes	397
4.8.2. <code>replace</code> — A String-Replacement Utility	397
4.8.3. <code>resolveip</code> — Resolve Host name to IP Address or Vice Versa	398
5. MySQL Server Administration	401
5.1. The MySQL Server	401
5.1.1. Server Option and Variable Reference	402
5.1.2. Server Configuration Defaults	433
5.1.3. Server Command Options	433
5.1.4. Server System Variables	467
5.1.5. Using System Variables	573
5.1.6. Server Status Variables	585
5.1.7. Server SQL Modes	611
5.1.8. Server Plugins	618
5.1.9. IPv6 Support	623
5.1.10. Server-Side Help	627
5.1.11. Server Response to Signals	627
5.1.12. The Shutdown Process	628
5.2. MySQL Server Logs	629
5.2.1. Selecting General Query and Slow Query Log Output Destinations	630
5.2.2. The Error Log	632
5.2.3. The General Query Log	633
5.2.4. The Binary Log	634
5.2.5. The Slow Query Log	647
5.2.6. Server Log Maintenance	648
5.3. Running Multiple MySQL Instances on One Machine	650
5.3.1. Setting Up Multiple Data Directories	651
5.3.2. Running Multiple MySQL Instances on Windows	652
5.3.3. Running Multiple MySQL Instances on Unix	655
5.3.4. Using Client Programs in a Multiple-Server Environment	656
5.4. Tracing <code>mysqld</code> Using DTrace	657
5.4.1. <code>mysqld</code> DTrace Probe Reference	659
6. Security	677
6.1. General Security Issues	678
6.1.1. Security Guidelines	678
6.1.2. Keeping Passwords Secure	679
6.1.3. Making MySQL Secure Against Attackers	687
6.1.4. Security-Related <code>mysqld</code> Options and Variables	689
6.1.5. How to Run MySQL as a Normal User	690
6.1.6. Security Issues with <code>LOAD DATA LOCAL</code>	690
6.1.7. Client Programming Security Guidelines	691
6.2. The MySQL Access Privilege System	692
6.2.1. Privileges Provided by MySQL	693
6.2.2. Privilege System Grant Tables	698
6.2.3. Specifying Account Names	703
6.2.4. Access Control, Stage 1: Connection Verification	705
6.2.5. Access Control, Stage 2: Request Verification	708
6.2.6. When Privilege Changes Take Effect	710
6.2.7. Causes of Access-Denied Errors	711
6.3. MySQL User Account Management	716
6.3.1. User Names and Passwords	716

6.3.2. Adding User Accounts	718
6.3.3. Removing User Accounts	721
6.3.4. Setting Account Resource Limits	721
6.3.5. Assigning Account Passwords	723
6.3.6. Pluggable Authentication	725
6.3.7. Proxy Users	742
6.3.8. Using SSL for Secure Connections	745
6.3.9. Connecting to MySQL Remotely from Windows with SSH	756
6.3.10. MySQL Enterprise Audit	757
6.3.11. SQL-Based MySQL Account Activity Auditing	768
7. Backup and Recovery	771
7.1. Backup and Recovery Types	772
7.2. Database Backup Methods	775
7.3. Example Backup and Recovery Strategy	777
7.3.1. Establishing a Backup Policy	778
7.3.2. Using Backups for Recovery	780
7.3.3. Backup Strategy Summary	780
7.4. Using <code>mysqldump</code> for Backups	780
7.4.1. Dumping Data in SQL Format with <code>mysqldump</code>	781
7.4.2. Reloading SQL-Format Backups	782
7.4.3. Dumping Data in Delimited-Text Format with <code>mysqldump</code>	782
7.4.4. Reloading Delimited-Text Format Backups	784
7.4.5. <code>mysqldump</code> Tips	784
7.5. Point-in-Time (Incremental) Recovery Using the Binary Log	786
7.5.1. Point-in-Time Recovery Using Event Times	787
7.5.2. Point-in-Time Recovery Using Event Positions	788
7.6. <code>MyISAM</code> Table Maintenance and Crash Recovery	789
7.6.1. Using <code>myisamchk</code> for Crash Recovery	789
7.6.2. How to Check <code>MyISAM</code> Tables for Errors	790
7.6.3. How to Repair <code>MyISAM</code> Tables	791
7.6.4. <code>MyISAM</code> Table Optimization	793
7.6.5. Setting Up a <code>MyISAM</code> Table Maintenance Schedule	794
8. Optimization	795
8.1. Optimization Overview	796
8.2. Optimizing SQL Statements	798
8.2.1. Optimizing <code>SELECT</code> Statements	798
8.2.2. Optimizing DML Statements	802
8.2.3. Optimizing Database Privileges	804
8.2.4. Optimizing <code>INFORMATION_SCHEMA</code> Queries	804
8.2.5. Other Optimization Tips	809
8.3. Optimization and Indexes	810
8.3.1. How MySQL Uses Indexes	810
8.3.2. Using Primary Keys	811
8.3.3. Using Foreign Keys	811
8.3.4. Column Indexes	811
8.3.5. Multiple-Column Indexes	812
8.3.6. Verifying Index Usage	814
8.3.7. <code>InnoDB</code> and <code>MyISAM</code> Index Statistics Collection	814
8.3.8. Comparison of B-Tree and Hash Indexes	815
8.4. Optimizing Database Structure	817
8.4.1. Optimizing Data Size	817
8.4.2. Optimizing MySQL Data Types	819
8.4.3. Optimizing for Many Tables	820
8.5. Optimizing for <code>InnoDB</code> Tables	823

8.5.1. Optimizing Storage Layout for <code>InnoDB</code> Tables	823
8.5.2. Optimizing <code>InnoDB</code> Transaction Management	823
8.5.3. Optimizing <code>InnoDB</code> Logging	824
8.5.4. Bulk Data Loading for <code>InnoDB</code> Tables	824
8.5.5. Optimizing <code>InnoDB</code> Queries	825
8.5.6. Optimizing <code>InnoDB</code> DDL Operations	826
8.5.7. Optimizing <code>InnoDB</code> Disk I/O	826
8.5.8. Optimizing <code>InnoDB</code> Configuration Variables	827
8.5.9. Optimizing <code>InnoDB</code> for Systems with Many Tables	828
8.6. Optimizing for <code>MyISAM</code> Tables	828
8.6.1. Optimizing <code>MyISAM</code> Queries	828
8.6.2. Bulk Data Loading for <code>MyISAM</code> Tables	830
8.6.3. Speed of <code>REPAIR TABLE</code> Statements	831
8.7. Optimizing for <code>MEMORY</code> Tables	833
8.8. Understanding the Query Execution Plan	833
8.8.1. Optimizing Queries with <code>EXPLAIN</code>	833
8.8.2. <code>EXPLAIN</code> Output Format	834
8.8.3. <code>EXPLAIN EXTENDED</code> Output Format	844
8.8.4. Estimating Query Performance	846
8.8.5. Controlling the Query Optimizer	846
8.9. Buffering and Caching	848
8.9.1. The <code>InnoDB</code> Buffer Pool	848
8.9.2. The <code>MyISAM</code> Key Cache	851
8.9.3. The MySQL Query Cache	855
8.10. Optimizing Locking Operations	862
8.10.1. Internal Locking Methods	862
8.10.2. Table Locking Issues	864
8.10.3. Concurrent Inserts	866
8.10.4. Metadata Locking Within Transactions	866
8.10.5. External Locking	867
8.11. Optimizing the MySQL Server	868
8.11.1. System Factors and Startup Parameter Tuning	868
8.11.2. Tuning Server Parameters	868
8.11.3. Optimizing Disk I/O	875
8.11.4. Optimizing Memory Use	879
8.11.5. Optimizing Network Use	882
8.11.6. The Thread Pool Plugin	884
8.12. Measuring Performance (Benchmarking)	890
8.12.1. Measuring the Speed of Expressions and Functions	891
8.12.2. The MySQL Benchmark Suite	891
8.12.3. Using Your Own Benchmarks	892
8.12.4. Measuring Performance with <code>performance_schema</code>	892
8.12.5. Examining Thread Information	892
8.13. Internal Details of MySQL Optimizations	907
8.13.1. Range Optimization	907
8.13.2. Index Merge Optimization	910
8.13.3. Engine Condition Pushdown Optimization	913
8.13.4. <code>IS NULL</code> Optimization	915
8.13.5. <code>LEFT JOIN</code> and <code>RIGHT JOIN</code> Optimization	916
8.13.6. Nested-Loop Join Algorithms	917
8.13.7. Nested Join Optimization	918
8.13.8. Outer Join Simplification	924
8.13.9. <code>ORDER BY</code> Optimization	926
8.13.10. <code>GROUP BY</code> Optimization	929

8.13.11. <code>DISTINCT</code> Optimization	931
8.13.12. Optimizing Subqueries with <code>EXISTS</code> Strategy	932
9. Language Structure	937
9.1. Literal Values	937
9.1.1. String Literals	937
9.1.2. Number Literals	940
9.1.3. Date and Time Literals	940
9.1.4. Hexadecimal Literals	942
9.1.5. Boolean Literals	942
9.1.6. Bit-Field Literals	943
9.1.7. <code>NULL</code> Values	943
9.2. Schema Object Names	943
9.2.1. Identifier Qualifiers	945
9.2.2. Identifier Case Sensitivity	946
9.2.3. Mapping of Identifiers to File Names	948
9.2.4. Function Name Parsing and Resolution	950
9.3. Reserved Words	953
9.4. User-Defined Variables	956
9.5. Expression Syntax	959
9.6. Comment Syntax	961
10. Globalization	963
10.1. Character Set Support	963
10.1.1. Character Sets and Collations in General	964
10.1.2. Character Sets and Collations in MySQL	965
10.1.3. Specifying Character Sets and Collations	966
10.1.4. Connection Character Sets and Collations	973
10.1.5. Configuring the Character Set and Collation for Applications	976
10.1.6. Character Set for Error Messages	977
10.1.7. Collation Issues	979
10.1.8. String Repertoire	987
10.1.9. Operations Affected by Character Set Support	989
10.1.10. Unicode Support	992
10.1.11. Upgrading from Previous to Current Unicode Support	996
10.1.12. UTF-8 for Metadata	998
10.1.13. Column Character Set Conversion	1000
10.1.14. Character Sets and Collations That MySQL Supports	1001
10.2. Setting the Error Message Language	1013
10.3. Adding a Character Set	1014
10.3.1. Character Definition Arrays	1016
10.3.2. String Collating Support for Complex Character Sets	1017
10.3.3. Multi-Byte Character Support for Complex Character Sets	1017
10.4. Adding a Collation to a Character Set	1017
10.4.1. Collation Implementation Types	1018
10.4.2. Choosing a Collation ID	1020
10.4.3. Adding a Simple Collation to an 8-Bit Character Set	1021
10.4.4. Adding a UCA Collation to a Unicode Character Set	1022
10.5. Character Set Configuration	1025
10.6. MySQL Server Time Zone Support	1026
10.6.1. Staying Current with Time Zone Changes	1028
10.6.2. Time Zone Leap Second Support	1030
10.7. MySQL Server Locale Support	1031
11. Data Types	1035
11.1. Data Type Overview	1036
11.1.1. Numeric Type Overview	1036

11.1.2. Date and Time Type Overview	1039
11.1.3. String Type Overview	1041
11.2. Numeric Types	1044
11.2.1. Integer Types (Exact Value) - <code>INTEGER</code> , <code>INT</code> , <code>SMALLINT</code> , <code>TINYINT</code> , <code>MEDIUMINT</code> , <code>BIGINT</code>	1044
11.2.2. Fixed-Point Types (Exact Value) - <code>DECIMAL</code> , <code>NUMERIC</code>	1045
11.2.3. Floating-Point Types (Approximate Value) - <code>FLOAT</code> , <code>DOUBLE</code>	1045
11.2.4. Bit-Value Type - <code>BIT</code>	1046
11.2.5. Numeric Type Attributes	1046
11.2.6. Out-of-Range and Overflow Handling	1047
11.3. Date and Time Types	1048
11.3.1. The <code>DATE</code> , <code>DATETIME</code> , and <code>TIMESTAMP</code> Types	1050
11.3.2. The <code>TIME</code> Type	1051
11.3.3. The <code>YEAR</code> Type	1051
11.3.4. <code>YEAR(2)</code> Limitations and Migrating to <code>YEAR(4)</code>	1052
11.3.5. Automatic Initialization and Updating for <code>TIMESTAMP</code>	1054
11.3.6. Fractional Seconds in Time Values	1057
11.3.7. Conversion Between Date and Time Types	1057
11.3.8. Two-Digit Years in Dates	1058
11.4. String Types	1058
11.4.1. The <code>CHAR</code> and <code>VARCHAR</code> Types	1058
11.4.2. The <code>BINARY</code> and <code>VARBINARY</code> Types	1060
11.4.3. The <code>BLOB</code> and <code>TEXT</code> Types	1061
11.4.4. The <code>ENUM</code> Type	1062
11.4.5. The <code>SET</code> Type	1065
11.5. Data Type Default Values	1068
11.6. Data Type Storage Requirements	1069
11.7. Choosing the Right Type for a Column	1072
11.8. Using Data Types from Other Database Engines	1072
12. Functions and Operators	1075
12.1. Function and Operator Reference	1076
12.2. Type Conversion in Expression Evaluation	1083
12.3. Operators	1086
12.3.1. Operator Precedence	1087
12.3.2. Comparison Functions and Operators	1088
12.3.3. Logical Operators	1094
12.3.4. Assignment Operators	1095
12.4. Control Flow Functions	1097
12.5. String Functions	1099
12.5.1. String Comparison Functions	1111
12.5.2. Regular Expressions	1114
12.6. Numeric Functions and Operators	1120
12.6.1. Arithmetic Operators	1121
12.6.2. Mathematical Functions	1123
12.7. Date and Time Functions	1131
12.8. What Calendar Is Used By MySQL?	1152
12.9. Full-Text Search Functions	1153
12.9.1. Natural Language Full-Text Searches	1154
12.9.2. Boolean Full-Text Searches	1157
12.9.3. Full-Text Searches with Query Expansion	1160
12.9.4. Full-Text Stopwords	1160
12.9.5. Full-Text Restrictions	1164
12.9.6. Fine-Tuning MySQL Full-Text Search	1164
12.9.7. Adding a Collation for Full-Text Indexing	1166

12.10. Cast Functions and Operators	1168
12.11. XML Functions	1171
12.12. Bit Functions	1182
12.13. Encryption and Compression Functions	1183
12.14. Information Functions	1190
12.15. Miscellaneous Functions	1198
12.16. Functions and Modifiers for Use with <code>GROUP BY</code> Clauses	1203
12.16.1. <code>GROUP BY</code> (Aggregate) Functions	1203
12.16.2. <code>GROUP BY</code> Modifiers	1207
12.16.3. MySQL Extensions to <code>GROUP BY</code>	1209
12.17. Spatial Extensions	1211
12.17.1. Introduction to MySQL Spatial Support	1211
12.17.2. The OpenGIS Geometry Model	1212
12.17.3. Supported Spatial Data Formats	1218
12.17.4. Creating a Spatially Enabled MySQL Database	1219
12.17.5. Spatial Analysis Functions	1225
12.17.6. Optimizing Spatial Analysis	1234
12.17.7. MySQL Conformance and Compatibility	1237
12.18. Precision Math	1237
12.18.1. Types of Numeric Values	1238
12.18.2. <code>DECIMAL</code> Data Type Changes	1238
12.18.3. Expression Handling	1240
12.18.4. Rounding Behavior	1241
12.18.5. Precision Math Examples	1242
13. SQL Statement Syntax	1247
13.1. Data Definition Statements	1248
13.1.1. <code>ALTER DATABASE</code> Syntax	1248
13.1.2. <code>ALTER EVENT</code> Syntax	1249
13.1.3. <code>ALTER LOGFILE GROUP</code> Syntax	1251
13.1.4. <code>ALTER FUNCTION</code> Syntax	1252
13.1.5. <code>ALTER PROCEDURE</code> Syntax	1252
13.1.6. <code>ALTER SERVER</code> Syntax	1253
13.1.7. <code>ALTER TABLE</code> Syntax	1253
13.1.8. <code>ALTER TABLESPACE</code> Syntax	1270
13.1.9. <code>ALTER VIEW</code> Syntax	1271
13.1.10. <code>CREATE DATABASE</code> Syntax	1272
13.1.11. <code>CREATE EVENT</code> Syntax	1272
13.1.12. <code>CREATE FUNCTION</code> Syntax	1277
13.1.13. <code>CREATE INDEX</code> Syntax	1277
13.1.14. <code>CREATE LOGFILE GROUP</code> Syntax	1281
13.1.15. <code>CREATE PROCEDURE</code> and <code>CREATE FUNCTION</code> Syntax	1283
13.1.16. <code>CREATE SERVER</code> Syntax	1288
13.1.17. <code>CREATE TABLE</code> Syntax	1289
13.1.18. <code>CREATE TABLESPACE</code> Syntax	1312
13.1.19. <code>CREATE TRIGGER</code> Syntax	1313
13.1.20. <code>CREATE VIEW</code> Syntax	1317
13.1.21. <code>DROP DATABASE</code> Syntax	1321
13.1.22. <code>DROP EVENT</code> Syntax	1322
13.1.23. <code>DROP FUNCTION</code> Syntax	1322
13.1.24. <code>DROP INDEX</code> Syntax	1322
13.1.25. <code>DROP LOGFILE GROUP</code> Syntax	1323
13.1.26. <code>DROP PROCEDURE</code> and <code>DROP FUNCTION</code> Syntax	1323
13.1.27. <code>DROP SERVER</code> Syntax	1323
13.1.28. <code>DROP TABLE</code> Syntax	1323

13.1.29. DROP TABLESPACE Syntax	1324
13.1.30. DROP TRIGGER Syntax	1324
13.1.31. DROP VIEW Syntax	1325
13.1.32. RENAME TABLE Syntax	1325
13.1.33. TRUNCATE TABLE Syntax	1326
13.2. Data Manipulation Statements	1327
13.2.1. CALL Syntax	1327
13.2.2. DELETE Syntax	1329
13.2.3. DO Syntax	1333
13.2.4. HANDLER Syntax	1333
13.2.5. INSERT Syntax	1334
13.2.6. LOAD DATA INFILE Syntax	1342
13.2.7. LOAD XML Syntax	1351
13.2.8. REPLACE Syntax	1357
13.2.9. SELECT Syntax	1358
13.2.10. Subquery Syntax	1377
13.2.11. UPDATE Syntax	1388
13.3. MySQL Transactional and Locking Statements	1390
13.3.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax	1390
13.3.2. Statements That Cannot Be Rolled Back	1392
13.3.3. Statements That Cause an Implicit Commit	1392
13.3.4. SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax	1393
13.3.5. LOCK TABLES and UNLOCK TABLES Syntax	1394
13.3.6. SET TRANSACTION Syntax	1400
13.3.7. XA Transactions	1402
13.4. Replication Statements	1406
13.4.1. SQL Statements for Controlling Master Servers	1406
13.4.2. SQL Statements for Controlling Slave Servers	1408
13.5. SQL Syntax for Prepared Statements	1415
13.5.1. PREPARE Syntax	1418
13.5.2. EXECUTE Syntax	1418
13.5.3. DEALLOCATE PREPARE Syntax	1419
13.5.4. Automatic Prepared Statement Repreparation	1419
13.6. MySQL Compound-Statement Syntax	1419
13.6.1. BEGIN ... END Compound-Statement Syntax	1419
13.6.2. Statement Label Syntax	1420
13.6.3. DECLARE Syntax	1421
13.6.4. Variables in Stored Programs	1421
13.6.5. Flow Control Statements	1423
13.6.6. Cursors	1427
13.6.7. Condition Handling	1428
13.7. Database Administration Statements	1443
13.7.1. Account Management Statements	1443
13.7.2. Table Maintenance Statements	1457
13.7.3. Plugin and User-Defined Function Statements	1465
13.7.4. SET Syntax	1468
13.7.5. SHOW Syntax	1471
13.7.6. Other Administrative Statements	1511
13.8. MySQL Utility Statements	1519
13.8.1. DESCRIBE Syntax	1519
13.8.2. EXPLAIN Syntax	1519
13.8.3. HELP Syntax	1520
13.8.4. USE Syntax	1522
14. Storage Engines	1523

14.1. Setting the Storage Engine	1527
14.2. Overview of MySQL Storage Engine Architecture	1527
14.2.1. Pluggable Storage Engine Architecture	1528
14.2.2. The Common Database Server Layer	1528
14.3. The <code>InnoDB</code> Storage Engine	1529
14.3.1. <code>InnoDB</code> as the Default MySQL Storage Engine	1531
14.3.2. Configuring <code>InnoDB</code>	1534
14.3.3. Using Per-Table Tablespaces	1538
14.3.4. <code>InnoDB</code> Startup Options and System Variables	1542
14.3.5. Creating and Using <code>InnoDB</code> Tables	1576
14.3.6. Adding, Removing, or Resizing <code>InnoDB</code> Data and Log Files	1591
14.3.7. Backing Up and Recovering an <code>InnoDB</code> Database	1592
14.3.8. Moving or Copying <code>InnoDB</code> Tables to Another Machine	1596
14.3.9. The <code>InnoDB</code> Transaction Model and Locking	1596
14.3.10. <code>InnoDB</code> Multi-Versioning	1609
14.3.11. <code>InnoDB</code> Table and Index Structures	1610
14.3.12. <code>InnoDB</code> Disk I/O and File Space Management	1613
14.3.13. <code>InnoDB</code> Error Handling	1615
14.3.14. <code>InnoDB</code> Performance Tuning and Troubleshooting	1620
14.3.15. Limits on <code>InnoDB</code> Tables	1632
14.4. New Features of <code>InnoDB</code> 1.1	1635
14.4.1. Introduction to <code>InnoDB</code> 1.1	1635
14.4.2. Fast Index Creation in the <code>InnoDB</code> Storage Engine	1638
14.4.3. <code>InnoDB</code> Data Compression	1642
14.4.4. <code>InnoDB</code> File-Format Management	1651
14.4.5. How <code>InnoDB</code> Stores Variable-Length Columns	1656
14.4.6. <code>InnoDB</code> <code>INFORMATION_SCHEMA</code> tables	1658
14.4.7. <code>InnoDB</code> Performance and Scalability Enhancements	1666
14.4.8. Changes for Flexibility, Ease of Use and Reliability	1678
14.4.9. Installing the <code>InnoDB</code> Storage Engine	1683
14.4.10. Upgrading the <code>InnoDB</code> Storage Engine	1684
14.4.11. Downgrading the <code>InnoDB</code> Storage Engine	1684
14.4.12. <code>InnoDB</code> Storage Engine Change History	1684
14.4.13. Third-Party Software	1685
14.4.14. List of Parameters Changed in <code>InnoDB</code> 1.1 and <code>InnoDB</code> Plugin 1.0	1688
14.5. The <code>MyISAM</code> Storage Engine	1691
14.5.1. <code>MyISAM</code> Startup Options	1693
14.5.2. Space Needed for Keys	1695
14.5.3. <code>MyISAM</code> Table Storage Formats	1695
14.5.4. <code>MyISAM</code> Table Problems	1698
14.6. The <code>MEMORY</code> Storage Engine	1699
14.7. The <code>CSV</code> Storage Engine	1703
14.7.1. Repairing and Checking <code>CSV</code> Tables	1704
14.7.2. <code>CSV</code> Limitations	1705
14.8. The <code>ARCHIVE</code> Storage Engine	1705
14.9. The <code>BLACKHOLE</code> Storage Engine	1706
14.10. The <code>MERGE</code> Storage Engine	1709
14.10.1. <code>MERGE</code> Table Advantages and Disadvantages	1711
14.10.2. <code>MERGE</code> Table Problems	1712
14.11. The <code>FEDERATED</code> Storage Engine	1714
14.11.1. <code>FEDERATED</code> Storage Engine Overview	1714
14.11.2. How to Create <code>FEDERATED</code> Tables	1715
14.11.3. <code>FEDERATED</code> Storage Engine Notes and Tips	1718
14.11.4. <code>FEDERATED</code> Storage Engine Resources	1719

14.12. The <code>EXAMPLE</code> Storage Engine	1719
14.13. Other Storage Engines	1720
15. High Availability and Scalability	1721
15.1. Oracle VM Template for MySQL Enterprise Edition	1724
15.2. Overview of MySQL with DRBD/Pacemaker/Corosync/Oracle Linux	1725
15.3. Overview of MySQL with Windows Failover Clustering	1727
15.4. Using MySQL within an Amazon EC2 Instance	1729
15.4.1. Setting Up MySQL on an EC2 AMI	1730
15.4.2. EC2 Instance Limitations	1731
15.4.3. Deploying a MySQL Database Using EC2	1731
15.5. Using ZFS Replication	1734
15.5.1. Using ZFS for File System Replication	1736
15.5.2. Configuring MySQL for ZFS Replication	1736
15.5.3. Handling MySQL Recovery with ZFS	1737
15.6. Using MySQL with <code>memcached</code>	1737
15.6.1. Installing <code>memcached</code>	1739
15.6.2. Using <code>memcached</code>	1740
15.6.3. Developing a <code>memcached</code> Application	1760
15.6.4. Getting <code>memcached</code> Statistics	1787
15.6.5. <code>memcached</code> FAQ	1796
15.7. MySQL Proxy	1799
15.7.1. MySQL Proxy Supported Platforms	1800
15.7.2. Installing MySQL Proxy	1800
15.7.3. MySQL Proxy Command Options	1803
15.7.4. MySQL Proxy Scripting	1813
15.7.5. Using MySQL Proxy	1828
15.7.6. MySQL Proxy FAQ	1833
16. Replication	1839
16.1. Replication Configuration	1840
16.1.1. How to Set Up Replication	1841
16.1.2. Replication Formats	1850
16.1.3. Replication and Binary Logging Options and Variables	1858
16.1.4. Common Replication Administration Tasks	1905
16.2. Replication Implementation	1907
16.2.1. Replication Implementation Details	1908
16.2.2. Replication Relay and Status Logs	1910
16.2.3. How Servers Evaluate Replication Filtering Rules	1913
16.3. Replication Solutions	1919
16.3.1. Using Replication for Backups	1920
16.3.2. Using Replication with Different Master and Slave Storage Engines	1923
16.3.3. Using Replication for Scale-Out	1925
16.3.4. Replicating Different Databases to Different Slaves	1926
16.3.5. Improving Replication Performance	1927
16.3.6. Switching Masters During Failover	1928
16.3.7. Setting Up Replication Using SSL	1931
16.3.8. Semisynchronous Replication	1932
16.4. Replication Notes and Tips	1937
16.4.1. Replication Features and Issues	1937
16.4.2. Replication Compatibility Between MySQL Versions	1960
16.4.3. Upgrading a Replication Setup	1962
16.4.4. Troubleshooting Replication	1963
16.4.5. How to Report Replication Bugs or Problems	1964
17. MySQL Cluster NDB 7.2	1967
17.1. MySQL Cluster Overview	1970

17.1.1. MySQL Cluster Core Concepts	1972
17.1.2. MySQL Cluster Nodes, Node Groups, Replicas, and Partitions	1975
17.1.3. MySQL Cluster Hardware, Software, and Networking Requirements	1977
17.1.4. MySQL Cluster Development History	1979
17.1.5. MySQL Server using InnoDB Compared with MySQL Cluster	1981
17.1.6. Known Limitations of MySQL Cluster	1986
17.2. MySQL Cluster Installation	1997
17.2.1. Installing MySQL Cluster on Linux	2000
17.2.2. Installing MySQL Cluster on Windows	2005
17.2.3. Initial Configuration of MySQL Cluster	2014
17.2.4. Initial Startup of MySQL Cluster	2016
17.2.5. MySQL Cluster Example with Tables and Data	2017
17.2.6. Safe Shutdown and Restart of MySQL Cluster	2021
17.2.7. Upgrading and Downgrading MySQL Cluster NDB 7.2	2022
17.3. MySQL Cluster Configuration	2022
17.3.1. Quick Test Setup of MySQL Cluster	2023
17.3.2. MySQL Cluster Configuration Files	2025
17.3.3. Overview of MySQL Cluster Configuration Parameters	2104
17.3.4. MySQL Server Options and Variables for MySQL Cluster	2126
17.3.5. Using High-Speed Interconnects with MySQL Cluster	2174
17.4. MySQL Cluster Programs	2176
17.4.1. ndbd — The MySQL Cluster Data Node Daemon	2176
17.4.2. ndbinfo_select_all — Select From ndbinfo Tables	2183
17.4.3. ndbmttd — The MySQL Cluster Data Node Daemon (Multi-Threaded)	2184
17.4.4. ndb_mgmd — The MySQL Cluster Management Server Daemon	2186
17.4.5. ndb_mgm — The MySQL Cluster Management Client	2194
17.4.6. ndb_config — Extract MySQL Cluster Configuration Information	2195
17.4.7. ndb_cpced — Automate Testing for NDB Development	2203
17.4.8. ndb_delete_all — Delete All Rows from an NDB Table	2203
17.4.9. ndb_desc — Describe NDB Tables	2204
17.4.10. ndb_drop_index — Drop Index from an NDB Table	2208
17.4.11. ndb_drop_table — Drop an NDB Table	2209
17.4.12. ndb_error_reporter — NDB Error-Reporting Utility	2209
17.4.13. ndb_index_stat — NDB Index Statistics Utility	2210
17.4.14. ndb_print_backup_file — Print NDB Backup File Contents	2215
17.4.15. ndb_print_schema_file — Print NDB Schema File Contents	2215
17.4.16. ndb_print_sys_file — Print NDB System File Contents	2215
17.4.17. ndbd_redo_log_reader — Check and Print Content of Cluster Redo Log	2216
17.4.18. ndb_restore — Restore a MySQL Cluster Backup	2217
17.4.19. ndb_select_all — Print Rows from an NDB Table	2229
17.4.20. ndb_select_count — Print Row Counts for NDB Tables	2232
17.4.21. ndb_show_tables — Display List of NDB Tables	2233
17.4.22. ndb_size.pl — NDBCLUSTER Size Requirement Estimator	2234
17.4.23. ndb_waiter — Wait for MySQL Cluster to Reach a Given Status	2237
17.4.24. Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs	2240
17.5. Management of MySQL Cluster	2243
17.5.1. Summary of MySQL Cluster Start Phases	2244
17.5.2. Commands in the MySQL Cluster Management Client	2246
17.5.3. Online Backup of MySQL Cluster	2249
17.5.4. MySQL Server Usage for MySQL Cluster	2254
17.5.5. Performing a Rolling Restart of a MySQL Cluster	2255
17.5.6. Event Reports Generated in MySQL Cluster	2258
17.5.7. MySQL Cluster Log Messages	2267

17.5.8. MySQL Cluster Single User Mode	2278
17.5.9. Quick Reference: MySQL Cluster SQL Statements	2279
17.5.10. The <code>ndbinfo</code> MySQL Cluster Information Database	2282
17.5.11. MySQL Cluster Security Issues	2302
17.5.12. MySQL Cluster Disk Data Tables	2310
17.5.13. Adding MySQL Cluster Data Nodes Online	2316
17.5.14. Distributed MySQL Privileges for MySQL Cluster	2327
17.5.15. NDB API Statistics Counters and Variables	2330
17.6. MySQL Cluster Replication	2342
17.6.1. MySQL Cluster Replication: Abbreviations and Symbols	2343
17.6.2. General Requirements for MySQL Cluster Replication	2344
17.6.3. Known Issues in MySQL Cluster Replication	2345
17.6.4. MySQL Cluster Replication Schema and Tables	2352
17.6.5. Preparing the MySQL Cluster for Replication	2355
17.6.6. Starting MySQL Cluster Replication (Single Replication Channel)	2356
17.6.7. Using Two Replication Channels for MySQL Cluster Replication	2358
17.6.8. Implementing Failover with MySQL Cluster Replication	2359
17.6.9. MySQL Cluster Backups With MySQL Cluster Replication	2360
17.6.10. MySQL Cluster Replication: Multi-Master and Circular Replication	2366
17.6.11. MySQL Cluster Replication Conflict Resolution	2371
17.7. MySQL Cluster Release Notes	2380
18. Partitioning	2381
18.1. Overview of Partitioning in MySQL	2383
18.2. Partitioning Types	2385
18.2.1. <code>RANGE</code> Partitioning	2387
18.2.2. <code>LIST</code> Partitioning	2391
18.2.3. <code>COLUMNS</code> Partitioning	2393
18.2.4. <code>HASH</code> Partitioning	2401
18.2.5. <code>KEY</code> Partitioning	2404
18.2.6. Subpartitioning	2406
18.2.7. How MySQL Partitioning Handles <code>NULL</code>	2409
18.3. Partition Management	2414
18.3.1. Management of <code>RANGE</code> and <code>LIST</code> Partitions	2415
18.3.2. Management of <code>HASH</code> and <code>KEY</code> Partitions	2420
18.3.3. Maintenance of Partitions	2421
18.3.4. Obtaining Information About Partitions	2422
18.4. Partition Pruning	2424
18.5. Restrictions and Limitations on Partitioning	2427
18.5.1. Partitioning Keys, Primary Keys, and Unique Keys	2433
18.5.2. Partitioning Limitations Relating to Storage Engines	2437
18.5.3. Partitioning Limitations Relating to Functions	2437
18.5.4. Partitioning and Table-Level Locking	2438
19. Stored Programs and Views	2441
19.1. Defining Stored Programs	2442
19.2. Using Stored Routines (Procedures and Functions)	2443
19.2.1. Stored Routine Syntax	2444
19.2.2. Stored Routines and MySQL Privileges	2444
19.2.3. Stored Routine Metadata	2445
19.2.4. Stored Procedures, Functions, Triggers, and <code>LAST_INSERT_ID()</code>	2445
19.3. Using Triggers	2445
19.3.1. Trigger Syntax	2446
19.3.2. Trigger Metadata	2448
19.4. Using the Event Scheduler	2448
19.4.1. Event Scheduler Overview	2449

19.4.2. Event Scheduler Configuration	2450
19.4.3. Event Syntax	2452
19.4.4. Event Metadata	2452
19.4.5. Event Scheduler Status	2453
19.4.6. The Event Scheduler and MySQL Privileges	2454
19.5. Using Views	2456
19.5.1. View Syntax	2457
19.5.2. View Processing Algorithms	2457
19.5.3. Updatable and Insertable Views	2459
19.5.4. View Metadata	2461
19.6. Access Control for Stored Programs and Views	2461
19.7. Binary Logging of Stored Programs	2462
20. INFORMATION_SCHEMA Tables	2469
20.1. The INFORMATION_SCHEMA CHARACTER_SETS Table	2472
20.2. The INFORMATION_SCHEMA COLLATIONS Table	2473
20.3. The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table	2473
20.4. The INFORMATION_SCHEMA COLUMNS Table	2473
20.5. The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table	2475
20.6. The INFORMATION_SCHEMA ENGINES Table	2475
20.7. The INFORMATION_SCHEMA EVENTS Table	2476
20.8. The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables	2479
20.9. The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables ...	2479
20.10. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table	2480
20.11. The INFORMATION_SCHEMA PARAMETERS Table	2481
20.12. The INFORMATION_SCHEMA PARTITIONS Table	2481
20.13. The INFORMATION_SCHEMA PLUGINS Table	2484
20.14. The INFORMATION_SCHEMA PROCESSLIST Table	2486
20.15. The INFORMATION_SCHEMA PROFILING Table	2486
20.16. The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table	2487
20.17. The INFORMATION_SCHEMA ROUTINES Table	2488
20.18. The INFORMATION_SCHEMA SCHEMATA Table	2489
20.19. The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table	2490
20.20. The INFORMATION_SCHEMA STATISTICS Table	2490
20.21. The INFORMATION_SCHEMA TABLES Table	2491
20.22. The INFORMATION_SCHEMA TABLESPACES Table	2492
20.23. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table	2493
20.24. The INFORMATION_SCHEMA TABLE_PRIVILEGES Table	2493
20.25. The INFORMATION_SCHEMA TRIGGERS Table	2494
20.26. The INFORMATION_SCHEMA USER_PRIVILEGES Table	2496
20.27. The INFORMATION_SCHEMA VIEWS Table	2496
20.28. INFORMATION_SCHEMA Tables for InnoDB	2497
20.28.1. The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables	2498
20.28.2. The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables	2498
20.28.3. The INFORMATION_SCHEMA INNODB_TRX Table	2499
20.28.4. The INFORMATION_SCHEMA INNODB_LOCKS Table	2500
20.28.5. The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table	2501
20.28.6. The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table	2502
20.28.7. The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table	2503
20.28.8. The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table	2503
20.29. INFORMATION_SCHEMA Tables for MySQL Cluster	2505
20.29.1. The INFORMATION_SCHEMA FILES Table	2505
20.29.2. The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table .	2510
20.30. Thread Pool INFORMATION_SCHEMA Tables	2512

20.30.1. The <code>INFORMATION_SCHEMA</code> <code>TP_THREAD_STATE</code> Table	2512
20.30.2. The <code>INFORMATION_SCHEMA</code> <code>TP_THREAD_GROUP_STATE</code> Table	2513
20.30.3. The <code>INFORMATION_SCHEMA</code> <code>TP_THREAD_GROUP_STATS</code> Table	2514
20.31. Extensions to <code>SHOW</code> Statements	2516
21. MySQL Performance Schema	2519
21.1. Performance Schema Quick Start	2520
21.2. Performance Schema Configuration	2526
21.2.1. Performance Schema Build Configuration	2526
21.2.2. Performance Schema Startup Configuration	2527
21.2.3. Performance Schema Runtime Configuration	2528
21.3. Performance Schema Queries	2535
21.4. Performance Schema Instrument Naming Conventions	2535
21.5. Performance Schema Status Monitoring	2537
21.6. Performance Schema General Table Characteristics	2540
21.7. Performance Schema Table Descriptions	2540
21.7.1. Performance Schema Setup Tables	2541
21.7.2. Performance Schema Instance Tables	2543
21.7.3. Performance Schema Wait Event Tables	2546
21.7.4. Performance Schema Summary Tables	2548
21.7.5. Performance Schema Miscellaneous Tables	2551
21.8. Performance Schema System Variables	2552
21.9. Performance Schema Status Variables	2559
21.10. Performance Schema and Plugins	2560
21.11. Using the Performance Schema to Diagnose Problems	2560
22. Connectors and APIs	2563
22.1. MySQL Connector/ODBC	2567
22.1.1. Connector/ODBC Versions	2568
22.1.2. Connector/ODBC Introduction	2569
22.1.3. Connector/ODBC Installation	2572
22.1.4. Configuring Connector/ODBC	2583
22.1.5. Connector/ODBC Examples	2604
22.1.6. Connector/ODBC Reference	2632
22.1.7. Connector/ODBC Notes and Tips	2639
22.1.8. Connector/ODBC Support	2650
22.2. MySQL Connector/Net	2651
22.2.1. Connector/Net Versions	2652
22.2.2. Installing Connector/Net	2654
22.2.3. Connector/Net Visual Studio Integration	2662
22.2.4. Connector/Net Tutorials	2697
22.2.5. Connector/Net Programming	2742
22.2.6. Connector/Net Connection String Options Reference	2784
22.2.7. Connector/Net API Reference	2791
22.2.8. Connector/Net Support	2901
22.2.9. Connector/Net FAQ	2902
22.3. MySQL Connector/J	2903
22.3.1. Overview of MySQL Connector/J	2904
22.3.2. Connector/J Versions	2904
22.3.3. Connector/J Installation	2905
22.3.4. Connector/J Examples	2910
22.3.5. Connector/J (JDBC) Reference	2911
22.3.6. JDBC Concepts	2945
22.3.7. Connection Pooling with Connector/J	2954
22.3.8. Load Balancing with Connector/J	2958
22.3.9. Failover with Connector/J	2960

22.3.10. Using the Connector/J Interceptor Classes	2962
22.3.11. Using Connector/J with Tomcat	2962
22.3.12. Using Connector/J with JBoss	2964
22.3.13. Using Connector/J with Spring	2964
22.3.14. Using Connector/J with GlassFish	2969
22.3.15. Troubleshooting Connector/J Applications	2975
22.3.16. Connector/J Support	2983
22.4. MySQL Connector/C++	2985
22.4.1. Installing the MySQL Connector/C++ Binary	2988
22.4.2. Installing MySQL Connector/C++ from Source	2990
22.4.3. MySQL Connector/C++ Building Windows Applications with Microsoft Visual Studio	2996
22.4.4. MySQL Connector/C++ Building Linux Applications with NetBeans	3004
22.4.5. MySQL Connector/C++ Getting Started: Usage Examples	3009
22.4.6. MySQL Connector/C++ Tutorials	3015
22.4.7. MySQL Connector/C++ Debug Tracing	3022
22.4.8. MySQL Connector/C++ Usage Notes	3024
22.4.9. MySQL Connector/C++ Known Bugs and Issues	3028
22.4.10. MySQL Connector/C++ Feature requests	3029
22.4.11. MySQL Connector/C++ Support	3029
22.5. MySQL Connector/C	3029
22.5.1. Building MySQL Connector/C from the Source Code	3030
22.5.2. Testing MySQL Connector/C	3032
22.5.3. MySQL Connector/C FAQ	3033
22.6. MySQL Connector/Python	3034
22.6.1. Guidelines for Python Developers	3034
22.6.2. Connector/Python Versions	3036
22.6.3. Connector/Python Installation	3036
22.6.4. Connector/Python Coding Examples	3038
22.6.5. Connector/Python Tutorials	3044
22.6.6. Connector/Python Connection Arguments	3045
22.6.7. Connector/Python API Reference	3047
22.6.8. Connector/Python FAQ	3066
22.7. libmysqld, the Embedded MySQL Server Library	3066
22.7.1. Compiling Programs with <code>libmysqld</code>	3066
22.7.2. Restrictions When Using the Embedded MySQL Server	3067
22.7.3. Options with the Embedded Server	3067
22.7.4. Embedded Server Examples	3068
22.8. MySQL C API	3071
22.8.1. C API Data Structures	3072
22.8.2. C API Function Overview	3077
22.8.3. C API Function Descriptions	3082
22.8.4. C API Prepared Statements	3132
22.8.5. C API Prepared Statement Data Structures	3133
22.8.6. C API Prepared Statement Function Overview	3140
22.8.7. C API Prepared Statement Function Descriptions	3142
22.8.8. C API Threaded Function Descriptions	3167
22.8.9. C API Embedded Server Function Descriptions	3168
22.8.10. C API Client Plugin Functions	3169
22.8.11. Common Questions and Problems When Using the C API	3172
22.8.12. Controlling Automatic Reconnection Behavior	3174
22.8.13. C API Support for Multiple Statement Execution	3175
22.8.14. C API Prepared Statement Problems	3177
22.8.15. C API Prepared Statement Handling of Date and Time Values	3177

22.8.16. C API Support for Prepared <code>CALL</code> Statements	3179
22.8.17. Building Client Programs	3183
22.9. MySQL PHP API	3185
22.9.1. Overview of the MySQL PHP drivers	3186
22.9.2. Original MySQL API (<code>Mysql</code>)	3193
22.9.3. MySQL Improved Extension (<code>Mysqli</code>)	3267
22.9.4. MySQL Functions (PDO_MYSQL) (<code>MySQL (PDO)</code>)	3497
22.9.5. MySQL Native Driver (<code>Mysqlnd</code>)	3502
22.9.6. Mysqlnd replication and load balancing plugin (<code>mysqlnd_ms</code>)	3535
22.9.7. Mysqlnd query result cache plugin (<code>mysqlnd_qc</code>)	3633
22.9.8. Mysqlnd user handler plugin (<code>mysqlnd_uh</code>)	3686
22.9.9. Mysqlnd connection multiplexing plugin (<code>mysqlnd_mux</code>)	3774
22.9.10. Mysqlnd Memache plugin (<code>mysqlnd_memcache</code>)	3778
22.9.11. Connector/PHP	3788
22.9.12. Common Problems with MySQL and PHP	3788
22.9.13. Enabling Both <code>mysql</code> and <code>mysqli</code> in PHP	3789
22.10. MySQL Perl API	3789
22.11. MySQL Python API	3790
22.12. MySQL Ruby APIs	3790
22.12.1. The MySQL/Ruby API	3790
22.12.2. The Ruby/MySQL API	3791
22.13. MySQL Tcl API	3791
22.14. MySQL Eiffel Wrapper	3791
23. Extending MySQL	3793
23.1. MySQL Internals	3793
23.1.1. MySQL Threads	3793
23.1.2. The MySQL Test Suite	3794
23.2. The MySQL Plugin API	3795
23.2.1. Plugin API Characteristics	3795
23.2.2. Plugin API Components	3796
23.2.3. Types of Plugins	3797
23.2.4. Writing Plugins	3800
23.2.5. MySQL Services for Plugins	3840
23.3. Adding New Functions to MySQL	3841
23.3.1. Features of the User-Defined Function Interface	3842
23.3.2. Adding a New User-Defined Function	3842
23.3.3. Adding a New Native Function	3852
23.4. Debugging and Porting MySQL	3854
23.4.1. Debugging a MySQL Server	3854
23.4.2. Debugging a MySQL Client	3861
23.4.3. The DEBUG Package	3861
24. MySQL Enterprise Edition	3863
24.1. MySQL Enterprise Monitor	3863
24.2. MySQL Enterprise Backup	3864
24.3. MySQL Enterprise Security	3865
24.4. MySQL Enterprise Audit	3865
24.5. MySQL Thread Pool	3865
25. MySQL Workbench	3867
26. MySQL for Excel	3869
26.1. Installing and Configuring	3869
26.2. Edit MySQL Data	3870
26.3. Import MySQL Data	3871
26.4. Append MySQL Data	3872
26.5. Export MySQL Data	3876

A. Licenses for Third-Party Components	3879
A.1. Ant-Contrib License	3882
A.2. ANTLR 3 License	3883
A.3. ANTLR 3.3 License	3884
A.4. Boost Library License	3884
A.5. c3p0 JDBC Library License	3885
A.6. dtoa.c License	3885
A.7. Editline Library (libedit) License	3885
A.8. FindGTest.cmake License	3887
A.9. Fred Fish's Dbug Library License	3888
A.10. getarg License	3889
A.11. GLib License (for MySQL Proxy)	3890
A.12. GNU General Public License Version 2.0, June 1991	3890
A.13. GNU Lesser General Public License Version 2.1, February 1999	3896
A.14. GNU Libtool License	3904
A.15. GNU Readline License	3905
A.16. Google Controlling Master Thread I/O Rate Patch License	3905
A.17. Google Perftools (TCMalloc utility) License	3905
A.18. Google SMP Patch License	3906
A.19. jboss-common-jdbc-wrapper.jar License	3907
A.20. lib_sql.cc License	3907
A.21. Libaio License	3907
A.22. libevent License	3907
A.23. Libiconv License	3909
A.24. libintl License	3910
A.25. Linux-PAM License	3910
A.26. LPeg Library License	3911
A.27. Lua (liblua) License	3911
A.28. LuaFileSystem Library License	3912
A.29. md5 (Message-Digest Algorithm 5) License	3912
A.30. memcached License	3913
A.31. nt_servc (Windows NT Service class library) License	3913
A.32. OpenPAM License	3913
A.33. Paramiko License	3914
A.34. PCRE License	3914
A.35. Percona Multiple I/O Threads Patch License	3915
A.36. Python License	3916
A.37. Red HAT RPM Spec File License	3926
A.38. RegEX-Spencer Library License	3927
A.39. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) License	3927
A.40. Richard A. O'Keefe String Library License	3928
A.41. SHA-1 in C License	3928
A.42. Simple Logging Facade for Java (SLF4J) License	3928
A.43. zlib License	3929
A.44. ZLIB.NET License	3929
B. MySQL 5.5 Frequently Asked Questions	3931
B.1. MySQL 5.5 FAQ: General	3931
B.2. MySQL 5.5 FAQ: Storage Engines	3933
B.3. MySQL 5.5 FAQ: Server SQL Mode	3934
B.4. MySQL 5.5 FAQ: Stored Procedures and Functions	3935
B.5. MySQL 5.5 FAQ: Triggers	3939
B.6. MySQL 5.5 FAQ: Views	3941
B.7. MySQL 5.5 FAQ: INFORMATION_SCHEMA	3942
B.8. MySQL 5.5 FAQ: Migration	3943

B.9. MySQL 5.5 FAQ: Security	3944
B.10. MySQL FAQ: MySQL 5.5 and MySQL Cluster	3945
B.11. MySQL 5.5 FAQ: MySQL Chinese, Japanese, and Korean Character Sets	3957
B.12. MySQL 5.5 FAQ: Connectors & APIs	3969
B.13. MySQL 5.5 FAQ: Replication	3969
B.14. MySQL 5.5 FAQ: MySQL Enterprise Scalability Thread Pool	3973
C. Errors, Error Codes, and Common Problems	3977
C.1. Sources of Error Information	3977
C.2. Types of Error Values	3977
C.3. Server Error Codes and Messages	3978
C.4. Client Error Codes and Messages	4029
C.5. Problems and Common Errors	4033
C.5.1. How to Determine What Is Causing a Problem	4033
C.5.2. Common Errors When Using MySQL Programs	4035
C.5.3. Installation-Related Issues	4048
C.5.4. Administration-Related Issues	4048
C.5.5. Query-Related Issues	4055
C.5.6. Optimizer-Related Issues	4063
C.5.7. Table Definition-Related Issues	4063
C.5.8. Known Issues in MySQL	4064
D. MySQL Release Notes	4069
D.1. Changes in MySQL 5.5 (Production)	4070
D.2. MySQL Connector/ODBC Change History	4070
D.3. MySQL Connector/Net Change History	4070
D.4. MySQL Visual Studio Plugin Change History	4070
D.5. MySQL Connector/J Change History	4070
D.6. MySQL Connector/C++ Change History	4070
D.7. MySQL Proxy Change History	4070
D.8. MySQL for Excel Change History	4070
D.9. MySQL Installer Change History	4071
E. Restrictions and Limits	4073
E.1. Restrictions on Stored Programs	4073
E.2. Restrictions on Condition Handling	4077
E.3. Restrictions on Server-Side Cursors	4077
E.4. Restrictions on Subqueries	4077
E.5. Restrictions on Views	4079
E.6. Restrictions on XA Transactions	4081
E.7. Restrictions on Character Sets	4081
E.8. Restrictions on Performance Schema	4082
E.9. Restrictions on Pluggable Authentication	4082
E.10. Limits in MySQL	4084
E.10.1. Limits of Joins	4084
E.10.2. Limits on Number of Databases and Tables	4084
E.10.3. Limits on Table Size	4085
E.10.4. Table Column-Count and Row-Size Limits	4086
E.10.5. Windows Platform Limitations	4088
F. MySQL Glossary	4091
General Index	4195
C Function Index	4277
Command Index	4287
Function Index	4313
INFORMATION_SCHEMA Index	4331
Join Types Index	4335
Operator Index	4337

Option Index	4343
Privileges Index	4397
SQL Modes Index	4403
Statement/Syntax Index	4407
System Variable Index	4449
Status Variable Index	4471
Transaction Isolation Level Index	4483

Preface and Legal Notices

This is the Reference Manual for the MySQL Database System, version 5.5, through release 5.5.31. Differences between minor versions of MySQL 5.5 are noted in the present text with reference to release numbers (5.5.x). For license information, see the [Legal Notices](#). This product may contain third-party code. For license information on third-party code, see [Appendix A, Licenses for Third-Party Components](#).

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.5 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, [MySQL 5.1 Reference Manual](#) covers the 5.1 series of MySQL software releases.

If you are using MySQL 5.6, please refer to the [MySQL 5.6 Reference Manual](#).

Legal Notices

Copyright © 1997, 2013, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used without Oracle's express written authorization. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and

its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle or as specifically provided below. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

For more information on the terms of this license, or for details on how the MySQL documentation is built and produced, please visit [MySQL Contact & Questions](#).

For additional licensing information, including licenses for third-party libraries used by MySQL products, see [Preface and Legal Notices](#).

For help with using MySQL, please visit either the [MySQL Forums](#) or [MySQL Mailing Lists](#) where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the [MySQL Documentation Library](#).

Chapter 1. General Information

Table of Contents

1.1. About This Manual	2
1.2. Typographical and Syntax Conventions	3
1.3. Overview of the MySQL Database Management System	4
1.3.1. What is MySQL?	4
1.3.2. The Main Features of MySQL	6
1.3.3. History of MySQL	9
1.4. What Is New in MySQL 5.5	9
1.4.1. Scalability Improvements	13
1.4.2. InnoDB I/O Subsystem Changes	14
1.4.3. Diagnostic and Monitoring Capabilities	14
1.4.4. Enhanced Solaris Support	15
1.5. MySQL Development History	16
1.6. MySQL Information Sources	16
1.6.1. MySQL Mailing Lists	16
1.6.2. MySQL Community Support at the MySQL Forums	19
1.6.3. MySQL Community Support on Internet Relay Chat (IRC)	19
1.6.4. MySQL Enterprise	20
1.7. How to Report Bugs or Problems	20
1.8. MySQL Standards Compliance	24
1.8.1. What Standards MySQL Follows	25
1.8.2. Selecting SQL Modes	25
1.8.3. Running MySQL in ANSI Mode	25
1.8.4. MySQL Extensions to Standard SQL	26
1.8.5. MySQL Differences from Standard SQL	29
1.8.6. How MySQL Deals with Constraints	33
1.9. Credits	36
1.9.1. Contributors to MySQL	36
1.9.2. Documenters and translators	41
1.9.3. Packages that support MySQL	42
1.9.4. Tools that were used to create MySQL	43
1.9.5. Supporters of MySQL	43

The MySQL™ software delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used by Customer without Oracle's express written authorization. Other names may be trademarks of their respective owners.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source product under the terms of the GNU General Public License (<http://www.fsf.org/licenses/>) or can purchase a standard commercial license from Oracle. See <http://www.mysql.com/company/legal/licensing/> for more information on our licensing policies.

The following list describes some sections of particular interest in this manual:

- For a discussion of MySQL Database Server capabilities, see [Section 1.3.2, “The Main Features of MySQL”](#).

- For an overview of new MySQL features, see [Section 1.4, “What Is New in MySQL 5.5”](#). For information about the changes in each version, see the [Release Notes](#).
- For installation instructions, see [Chapter 2, *Installing and Upgrading MySQL*](#). For information about upgrading MySQL, see [Section 2.12.1, “Upgrading MySQL”](#).
- For a tutorial introduction to the MySQL Database Server, see [Chapter 3, *Tutorial*](#).
- For information about configuring and administering MySQL Server, see [Chapter 5, *MySQL Server Administration*](#).
- For information about security in MySQL, see [Chapter 6, *Security*](#).
- For information about setting up replication servers, see [Chapter 16, *Replication*](#).
- For information about MySQL Enterprise, the commercial MySQL release with advanced features and management tools, see [Chapter 24, *MySQL Enterprise Edition*](#).
- For answers to a number of questions that are often asked concerning the MySQL Database Server and its capabilities, see [Appendix B, *MySQL 5.5 Frequently Asked Questions*](#).
- For a history of new features and bugfixes, see the [Release Notes](#).



Important

To report problems or bugs, please use the instructions at [Section 1.7, “How to Report Bugs or Problems”](#). If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to [<secalert_us@oracle.com>](mailto:secalert_us@oracle.com). Exception: Support customers should report all problems, including security bugs, to Oracle Support.

1.1. About This Manual

This is the Reference Manual for the MySQL Database System, version 5.5, through release 5.5.31. Differences between minor versions of MySQL 5.5 are noted in the present text with reference to release numbers (5.5.x). For license information, see the [Legal Notices](#). This product may contain third-party code. For license information on third-party code, see [Appendix A, *Licenses for Third-Party Components*](#).

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.5 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, [MySQL 5.1 Reference Manual](#) covers the 5.1 series of MySQL software releases.

If you are using MySQL 5.6, please refer to the [MySQL 5.6 Reference Manual](#).

Because this manual serves as a reference, it does not provide general instruction on SQL or relational database concepts. It also does not teach you how to use your operating system or command-line interpreter.

The MySQL Database Software is under constant development, and the Reference Manual is updated frequently as well. The most recent version of the manual is available online in searchable form at <http://dev.mysql.com/doc/>. Other formats also are available there, including HTML, PDF, and EPUB versions.

The Reference Manual source files are written in DocBook XML format. The HTML version and other formats are produced automatically, primarily using the DocBook XSL stylesheets. For information about DocBook, see <http://docbook.org/>

If you have questions about using MySQL, you can ask them using our mailing lists or forums. See [Section 1.6.1, “MySQL Mailing Lists”](#), and [Section 1.6.2, “MySQL Community Support at the MySQL Forums”](#). If you have suggestions concerning additions or corrections to the manual itself, please send them to the <http://www.mysql.com/company/contact/>.

This manual was originally written by David Axmark and Michael “Monty” Widenius. It is maintained by the MySQL Documentation Team, consisting of Paul DuBois, Stefan Hinz, Philip Olson, John Russell, and Jon Stephens.

1.2. Typographical and Syntax Conventions

This manual uses certain typographical conventions:

- *Text in this style* is used for SQL statements; database, table, and column names; program listings and source code; and environment variables. Example: “To reload the grant tables, use the `FLUSH PRIVILEGES` statement.”
- *Text in this style* indicates input that you type in examples.
- *Text in this style* indicates the names of executable programs and scripts, examples being `mysql` (the MySQL command line client program) and `mysqld` (the MySQL server executable).
- *Text in this style* is used for variable input for which you should substitute a value of your own choosing.
- *Text in this style* is used for emphasis.
- **Text in this style** is used in table headings and to convey especially strong emphasis.
- *Text in this style* is used to indicate a program option that affects how the program is executed, or that supplies information that is needed for the program to function in a certain way. *Example*: “The `--host` option (short form `-h`) tells the `mysql` client program the hostname or IP address of the MySQL server that it should connect to”.
- File names and directory names are written like this: “The global `my.cnf` file is located in the `/etc` directory.”
- Character sequences are written like this: “To specify a wildcard, use the `%` character.”

When commands are shown that are meant to be executed from within a particular program, the prompt shown preceding the command indicates which command to use. For example, `shell>` indicates a command that you execute from your login shell, `root-shell>` is similar but should be executed as `root`, and `mysql>` indicates a statement that you execute from the `mysql` client program:

```
shell> type a shell command here
root-shell> type a shell command as root here
mysql> type a mysql statement here
```

In some areas different systems may be distinguished from each other to show that commands should be executed in two different environments. For example, while working with replication the commands might be prefixed with `master` and `slave`:

```
master> type a mysql command on the replication master here
slave> type a mysql command on the replication slave here
```

The “shell” is your command interpreter. On Unix, this is typically a program such as `sh`, `csh`, or `bash`. On Windows, the equivalent program is `command.com` or `cmd.exe`, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses *db_name*, *tbl_name*, and *col_name*. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case sensitive and may be written in any lettercase. This manual uses uppercase.

In syntax descriptions, square brackets (“[” and “]”) indicate optional words or clauses. For example, in the following statement, `IF EXISTS` is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars (“|”). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets (“[” and “]”):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces (“{” and “}”):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (...) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, `SELECT ... INTO OUTFILE` is shorthand for the form of `SELECT` statement that has an `INTO OUTFILE` clause following other parts of the statement.

An ellipsis can also indicate that the preceding syntax element of a statement may be repeated. In the following example, multiple *reset_option* values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [,reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the `CC` environment variable and run the `configure` command looks like this in Bourne shell syntax:

```
shell> CC=gcc ./configure
```

If you are using `csh` or `tcsh`, you must issue commands somewhat differently:

```
shell> setenv CC gcc
shell> ./configure
```

1.3. Overview of the MySQL Database Management System

1.3.1. What is MySQL?

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation.

The MySQL Web site (<http://www.mysql.com/>) provides the latest information about MySQL software.

- **MySQL is a database management system.**

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

- **MySQL databases are relational.**

A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. The logical model, with objects such as databases, tables, views, rows, and columns, offers a flexible programming environment. You set up rules governing the relationships between different data fields, such as one-to-one, one-to-many, unique, required or optional, and “pointers” between different tables. The database enforces these rules, so that with a well-designed database, your application never sees inconsistent, duplicate, orphan, out-of-date, or missing data.

The SQL part of “MySQL” stands for “Structured Query Language”. SQL is the most common standardized language used to access databases. Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.

SQL is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” to mean the current version of the SQL Standard at any time.

- **MySQL software is Open Source.**

Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MySQL software uses the GPL (GNU General Public License), <http://www.fsf.org/licenses/>, to define what you may and may not do with the software in different situations. If you feel uncomfortable with the GPL or need to embed MySQL code into a commercial application, you can buy a commercially licensed version from us. See the MySQL Licensing Overview for more information (<http://www.mysql.com/company/legal/licensing/>).

- **The MySQL Database Server is very fast, reliable, scalable, and easy to use.**

If that is what you are looking for, you should give it a try. MySQL Server can run comfortably on a desktop or laptop, alongside your other applications, web servers, and so on, requiring little or no attention. If you dedicate an entire machine to MySQL, you can adjust the settings to take advantage of all the memory, CPU power, and I/O capacity available. MySQL can also scale up to clusters of machines, networked together.

You can find a performance comparison of MySQL Server with other database managers on our benchmark page. See [Section 8.12.2, “The MySQL Benchmark Suite”](#).

MySQL Server was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Although under constant development, MySQL Server today offers a rich and useful set of functions. Its connectivity, speed, and security make MySQL Server highly suited for accessing databases on the Internet.

- **MySQL Server works in client/server or embedded systems.**

The MySQL Database Software is a client/server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs).

We also provide MySQL Server as an embedded multi-threaded library that you can link into your application to get a smaller, faster, easier-to-manage standalone product.

- **A large amount of contributed MySQL software is available.**

MySQL Server has a practical set of features developed in close cooperation with our users. It is very likely that your favorite application or language supports the MySQL Database Server.

The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we do not mind if you pronounce it as “my sequel” or in some other localized way.

1.3.2. The Main Features of MySQL

This section describes some of the important characteristics of the MySQL Database Software. See also [Section 1.5, “MySQL Development History”](#). In most respects, the roadmap applies to all versions of MySQL. For information about features as they are introduced into MySQL on a series-specific basis, see the “In a Nutshell” section of the appropriate Manual:

- MySQL 5.6: [MySQL 5.6 in a Nutshell](#)
- MySQL 5.5: [MySQL 5.5 in a Nutshell](#)
- MySQL 5.1: [MySQL 5.1 in a Nutshell](#)
- MySQL 5.0: [MySQL 5.0 in a Nutshell](#)

Internals and Portability:

- Written in C and C++.
- Tested with a broad range of different compilers.
- Works on many different platforms. See [Section 2.1.1, “Operating Systems Supported by MySQL Community Server”](#).
- For portability, uses `CMake` in MySQL 5.5 and up. Previous series use GNU Automake, Autoconf, and Libtool.
- Tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (<http://developer.kde.org/~sewardj/>).
- Uses multi-layered server design with independent modules.
- Designed to be fully multi-threaded using kernel threads, to easily use multiple CPUs if they are available.
- Provides transactional and nontransactional storage engines.
- Uses very fast B-tree disk tables ([MyISAM](#)) with index compression.
- Designed to make it relatively easy to add other storage engines. This is useful if you want to provide an SQL interface for an in-house database.

- Uses a very fast thread-based memory allocation system.
- Executes very fast joins using an optimized nested-loop join.
- Implements in-memory hash tables, which are used as temporary tables.
- Implements SQL functions using a highly optimized class library that should be as fast as possible. Usually there is no memory allocation at all after query initialization.
- Provides the server as a separate program for use in a client/server networked environment, and as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

Data Types:

- Many data types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, [FLOAT](#), [DOUBLE](#), [CHAR](#), [VARCHAR](#), [BINARY](#), [VARBINARY](#), [TEXT](#), [BLOB](#), [DATE](#), [TIME](#), [DATETIME](#), [TIMESTAMP](#), [YEAR](#), [SET](#), [ENUM](#), and OpenGIS spatial types. See [Chapter 11, Data Types](#).
- Fixed-length and variable-length string types.

Statements and Functions:

- Full operator and function support in the [SELECT](#) list and [WHERE](#) clause of queries. For example:

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
-> FROM citizen
-> WHERE income/dependents > 10000 AND age > 30;
```

- Full support for SQL [GROUP BY](#) and [ORDER BY](#) clauses. Support for group functions ([COUNT\(\)](#) [1204], [AVG\(\)](#) [1203], [STD\(\)](#) [1206], [SUM\(\)](#) [1206], [MAX\(\)](#) [1205], [MIN\(\)](#) [1206], and [GROUP_CONCAT\(\)](#) [1205]).
- Support for [LEFT OUTER JOIN](#) and [RIGHT OUTER JOIN](#) with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- Support for [DELETE](#), [INSERT](#), [REPLACE](#), and [UPDATE](#) to return the number of rows that were changed (affected), or to return the number of rows matched instead by setting a flag when connecting to the server.
- Support for MySQL-specific [SHOW](#) statements that retrieve information about databases, storage engines, tables, and indexes. MySQL 5.0 adds support for the [INFORMATION_SCHEMA](#) database, implemented according to standard SQL.
- An [EXPLAIN](#) statement to show how the optimizer resolves a query.
- Independence of function names from table or column names. For example, [ABS](#) is a valid column name. The only restriction is that for a function call, no spaces are permitted between the function name and the "(" that follows it. See [Section 9.3, "Reserved Words"](#).
- You can refer to tables from different databases in the same statement.

Security:

- A privilege and password system that is very flexible and secure, and that enables host-based verification.
- Password security by encryption of all password traffic when you connect to a server.

Scalability and Limits:

- Support for large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 200,000 tables and about 5,000,000,000 rows.
- Support for up to 64 indexes per table (32 before MySQL 4.1.2). Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 767 bytes for [InnoDB](#) tables, or 1000 for [MyISAM](#); before MySQL 4.1.2, the limit is 500 bytes. An index may use a prefix of a column for [CHAR](#), [VARCHAR](#), [BLOB](#), or [TEXT](#) column types.

Connectivity:

- Clients can connect to MySQL Server using several protocols:
 - Clients can connect using TCP/IP sockets on any platform.
 - On Windows systems in the NT family (NT, 2000, XP, 2003, or Vista), clients can connect using named pipes if the server is started with the `--enable-named-pipe` [\[442\]](#) option. In MySQL 4.1 and higher, Windows servers also support shared-memory connections if started with the `--shared-memory` [\[460\]](#) option. Clients can connect through shared memory by using the `--protocol=memory` option.
 - On Unix systems, clients can connect using Unix domain socket files.
- MySQL client programs can be written in many languages. A client library written in C is available for clients written in C or C++, or for any language that provides C bindings.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available, enabling MySQL clients to be written in many languages. See [Chapter 22, Connectors and APIs](#).
- The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. Connector/ODBC source is available. All ODBC 2.5 functions are supported, as are many others. See [Section 22.1, “MySQL Connector/ODBC”](#).
- The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See [Section 22.3, “MySQL Connector/J”](#).
- MySQL Connector/Net enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. MySQL Connector/Net is a fully managed ADO.NET driver written in 100% pure C#. See [Section 22.2, “MySQL Connector/Net”](#).

Localization:

- The server can provide error messages to clients in many languages. See [Section 10.2, “Setting the Error Message Language”](#).
- Full support for several different character sets, including [latin1](#) (cp1252), [german](#), [big5](#), [ujis](#), and more. For example, the Scandinavian characters “å”, “ä” and “ö” are permitted in table and column names. Unicode support is available as of MySQL 4.1.
- All data is saved in the chosen character set.
- Sorting and comparisons are done according to the chosen character set and collation (using [latin1](#) and Swedish collation by default). It is possible to change this when the MySQL server is started. To see

an example of very advanced sorting, look at the Czech sorting code. MySQL Server supports many different character sets that can be specified at compile time and runtime.

- As of MySQL 4.1, the server time zone can be changed dynamically, and individual clients can specify their own time zone. [Section 10.6, “MySQL Server Time Zone Support”](#).

Clients and Tools:

- MySQL includes several client and utility programs. These include both command-line programs such as `mysqldump` and `mysqladmin`, and graphical programs such as [MySQL Workbench](#).
- MySQL Server has built-in support for SQL statements to check, optimize, and repair tables. These statements are available from the command line through the `mysqlcheck` client. MySQL also includes `myisamchk`, a very fast command-line utility for performing these operations on `MyISAM` tables. See [Chapter 4, MySQL Programs](#).
- MySQL programs can be invoked with the `--help` or `-?` option to obtain online assistance.

1.3.3. History of MySQL

We started out with the intention of using the `mSQL` database system to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing, we came to the conclusion that `mSQL` was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as `mSQL`. This API was designed to enable third-party code that was written for use with `mSQL` to be ported easily for use with MySQL.

MySQL is named after co-founder Monty Widenius's daughter, My.

The name of the MySQL Dolphin (our logo) is “Sakila,” which was chosen from a huge list of names suggested by users in our “Name the Dolphin” contest. The winning name was submitted by Ambrose Twebaze, an Open Source software developer from Swaziland, Africa. According to Ambrose, the feminine name Sakila has its roots in SiSwati, the local language of Swaziland. Sakila is also the name of a town in Arusha, Tanzania, near Ambrose's country of origin, Uganda.

1.4. What Is New in MySQL 5.5

This section summarizes what has been added to and removed from MySQL 5.5.

Added Features

The following features have been added to MySQL 5.5:

- **Thread pooling.** The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. As of MySQL 5.5.16, commercial distributions of MySQL include a thread pool plugin that provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections. For more information, see [Section 8.11.6, “The Thread Pool Plugin”](#).
- **MySQL Enterprise Audit.** MySQL Enterprise Edition now includes MySQL Enterprise Audit, implemented using a server plugin named `audit_log`. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines. When installed, the audit plugin

enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access. For more information, see [Section 6.3.10, “MySQL Enterprise Audit”](#).

- **Pluggable authentication.** MySQL authentication supports two new capabilities, pluggable authentication and proxy users. With pluggable authentication, the server can use plugins to authenticate incoming client connections, and clients can load an authentication plugin that interacts properly with the corresponding server plugin. This capability enables clients to connect to the MySQL server with credentials that are appropriate for authentication methods other than the built-in MySQL authentication based on native MySQL passwords stored in the `mysql.user` table. For example, plugins can be created to use external authentication methods such as LDAP, Kerberos, PAM, or Windows login IDs. Proxy user capability enables a client who connects and authenticates as one user to be treated, for purposes of access control while connected, as having the privileges of a different user. In effect, one user impersonates another. Proxy capability depends on pluggable authentication because it is based on having an authentication plugin return to the server the user name that the connecting user impersonates. See [Section 6.3.6, “Pluggable Authentication”](#), and [Section 6.3.7, “Proxy Users”](#).

As of MySQL 5.5.16, commercial distributions of MySQL include two plugins that enable MySQL Server to use external authentication methods to authenticate MySQL users:

- PAM (Pluggable Authentication Modules) enables a system to access various kinds of authentication methods through a standard interface. A PAM authentication plugin enables MySQL Server to use PAM to authenticate MySQL users.
- Distributions of MySQL for Windows include an authentication plugin that enables MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password.

These authentication plugins enable MySQL Server to accept connections from users defined outside the MySQL grant tables. They also support the MySQL proxy-user capability. Each plugin can return to MySQL a user name different from the login user, which means that the plugin can return the MySQL user that defines the privileges the externally authenticated user should have.

For more information, see [Section 6.3.6.2, “The PAM Authentication Plugin”](#), and [Section 6.3.6.3, “The Windows Native Authentication Plugin”](#).

- **Multi-core scalability.** Scalability on multi-core CPUs is improved. The trend in hardware development now is toward more cores rather than continued increases in CPU clock speeds, which renders “wait until CPUs get faster” a nonviable means of improving database performance. Instead, it is necessary to make better use of multiple cores to maximally exploit the processing cycles they make available. MySQL 5.5 takes advantage of features of SMP systems and tries to eliminate bottlenecks in MySQL architecture that hinder full use of multiple cores. The focus has been on [InnoDB](#), especially locking and memory management. See [Section 1.4.1, “Scalability Improvements”](#).
- **InnoDB I/O subsystem.** [InnoDB](#) I/O subsystem changes enable more effective use of available I/O capacity. See [Section 1.4.2, “InnoDB I/O Subsystem Changes”](#).
- **Diagnostic improvements.** There is better access to execution and performance information. Diagnostic improvements include Performance Schema (a feature for monitoring MySQL Server execution at a low level), DTrace probes, expanded `SHOW ENGINE INNODB STATUS` output, Debug Sync, and a new status variable. See [Section 1.4.3, “Diagnostic and Monitoring Capabilities”](#).
- **Solaris.** Several modifications improve operation of MySQL Server on Solaris. See [Section 1.4.4, “Enhanced Solaris Support”](#).

- **Default storage engine.** The default storage engine for new tables is [InnoDB](#) rather than [MyISAM](#). See [Section 14.3.1, “InnoDB as the Default MySQL Storage Engine”](#).
- **MySQL Cluster.** MySQL Cluster is released as a separate product, with new development for version 7.2 of the [NDB](#) storage engine being based on MySQL 5.5. Clustering support is not available in mainline MySQL Server 5.5 releases. For more information about MySQL Cluster NDB 7.2, see [Chapter 17, MySQL Cluster NDB 7.2](#).

MySQL Cluster releases are identified by a 3-part NDB version number. Currently, MySQL Cluster NDB 7.1 is the most recent GA release series. MySQL Cluster NDB 6.3 and MySQL Cluster NDB 7.0 are also still available. These versions of MySQL Cluster are based on MySQL Server 5.1 and documented in the *MySQL 5.1 Manual*. See [MySQL Cluster NDB 6.1 - 7.1](#).

- **Semisynchronous replication.** A commit performed on the master side blocks before returning to the session that performed the transaction until at least one slave acknowledges that it has received and logged the events for the transaction. Semisynchronous replication is implemented through an optional plugin component. See [Section 16.3.8, “Semisynchronous Replication”](#)
- **Unicode.** Support for supplementary Unicode characters; that is, characters outside the Basic Multilingual Plane (BMP). These new Unicode character sets include supplementary characters: [utf16](#), [utf32](#), and [utf8mb4](#). See [Section 10.1.10, “Unicode Support”](#).
- **Partitioning.** Enhancements to table partitioning:

- Two new types of user-defined partitioning are supported: [RANGE COLUMNS](#) partitioning is an extension to [RANGE](#) partitioning; [LIST COLUMNS](#) partitioning is an extension to [LIST](#) partitioning. Each of these extensions provides two enhancements to MySQL partitioning capabilities:
 - It is possible to define partitioning ranges or lists based on [DATE](#), [DATETIME](#), or string values (such as [CHAR](#) or [VARCHAR](#)).

You can also define ranges or lists based on multiple column values when partitioning tables by [RANGE COLUMNS](#) or [LIST COLUMNS](#), respectively. Such a range or list may refer to up to 16 columns.

- For tables defined using these partitioning types, partition pruning can now optimize queries with [WHERE](#) conditions that use multiple comparisons between (different) column values and constants, such as `a = 10 AND b > 5` or `a < "2005-11-25" AND b = 10 AND c = 50`.

See [Section 18.2.1, “RANGE Partitioning”](#), and [Section 18.2.2, “LIST Partitioning”](#).

- It is now possible to delete all rows from one or more partitions of a partitioned table using the [ALTER TABLE ... TRUNCATE PARTITION](#) statement. Executing the statement deletes rows without affecting the structure of the table. The partitions named in the [TRUNCATE PARTITION](#) clause do not have to be contiguous.
- Key caches are now supported for indexes on partitioned [MyISAM](#) tables, using the [CACHE INDEX](#) and [LOAD INDEX INTO CACHE](#) statements. In addition, a key cache can be defined for and loaded with indexes from an entire partitioned table, or for one or more partitions. In the latter case, the partitions are not required to be contiguous.
- The new [TO_SECONDS\(\)](#) [\[1149\]](#) function converts a date or datetime expression to a number of seconds since the year 0. This is a general-purpose function, but is useful for partitioning. You may use it in partitioning expressions, and partition pruning is supported for tables defined using such expressions.

- **SIGNAL and RESIGNAL.** Support for the SQL standard `SIGNAL` and `RESIGNAL` statements. See [Section 13.6.7, “Condition Handling”](#).
- **Metadata locking.** The server now prevents DDL statements from compromising transaction serializability by using a new class of locks called metadata locks. See [Section 8.10.4, “Metadata Locking Within Transactions”](#).
- **IPv6 support.** MySQL Server can accept TCP/IP connections from clients connecting over IPv6. See [Section 5.1.9, “IPv6 Support”](#).
- **XML.** Enhancements to XML functionality, including a new `LOAD XML INFILE` statement. See [Section 13.2.7, “LOAD XML Syntax”](#).
- **Build configuration.** MySQL releases are now built using `CMake` rather than the GNU autotools. Accordingly, the instructions for installing MySQL from source have been updated to discuss how to build MySQL using `CMake`. See [Section 2.10, “Installing MySQL from Source”](#).

The build process is now similar enough on all platforms, including Windows, that there are no longer sections dedicated to notes for specific platforms.

Removed Features

The following constructs are obsolete and have been removed in MySQL 5.5. Where alternatives are shown, applications should be updated to use them.

- The `language` system variable (use `lc_messages_dir` [\[512\]](#) and `lc_messages` [\[512\]](#)).
- The `log_bin_trust_routine_creators` system variable (use `log_bin_trust_function_creators` [\[514\]](#)).
- The `myisam_max_extra_sort_file_size` system variable.
- The `record_buffer` system variable (use `read_buffer_size` [\[544\]](#)).
- The `sql_log_update` system variable.
- The `Innodb_buffer_pool_read_ahead_rnd` and `Innodb_buffer_pool_read_ahead_seq` status variables (use `Innodb_buffer_pool_read_ahead` [\[600\]](#) and `Innodb_buffer_pool_read_ahead_evicted` [\[600\]](#)).
- The `table_lock_wait_timeout` system variable.
- The `table_type` system variable (use `storage_engine` [\[559\]](#)).
- The `FRAC_SECOND` modifier for the `TIMESTAMPADD()` [\[1147\]](#) function (use `MICROSECOND`).
- The `TYPE` table option to specify the storage engine for `CREATE TABLE` or `ALTER TABLE` (use `ENGINE`).
- The `SHOW TABLE TYPES` SQL statement (use `SHOW ENGINES`).
- The `SHOW INNODB STATUS` and `SHOW MUTEX STATUS` SQL statements (use `SHOW ENGINE INNODB STATUS` and `SHOW ENGINE INNODB MUTEX`).
- The `SHOW PLUGIN` SQL statement (use `SHOW PLUGINS`).
- The `LOAD TABLE ... FROM MASTER` and `LOAD DATA FROM MASTER` SQL statements (use `mysqldump` or `mysqlhotcopy` to dump tables and `mysql` to reload dump files).

- The `BACKUP TABLE` and `RESTORE TABLE` SQL statements (use `mysqldump` or `mysqlhotcopy` to dump tables and `mysql` to reload dump files).
- `TIMESTAMP(N)` data type: The ability to specify a display width of *N* (use without *N*).
- The `--default-character-set` and `--default-collation` server options (use `--character-set-server` [438] and `--collation-server` [438]).
- The `--default-table-type` server option (use `--default-storage-engine` [441]).
- The `--delay-key-write-for-all-tables` server option (use `--delay-key-write=ALL` [441]).
- The `--enable-locking` and `--skip-locking` server options (use `--external-locking` [444] and `--skip-external-locking` [460]).
- The `--log-bin-trust-routine-creators` server option (use `--log-bin-trust-function-creators` [1892]).
- The `--log-long-format` server option.
- The `--log-update` server option.
- The `--master-xxx` server options to set replication parameters (use the `CHANGE MASTER TO` statement instead): `--master-host`, `--master-user`, `--master-password`, `--master-port`, `--master-connect-retry`, `--master-ssl`, `--master-ssl-ca`, `--master-ssl-capath`, `--master-ssl-cert`, `--master-ssl-cipher`, `--master-ssl-key`.
- The `--safe-show-database` server option.
- The `--skip-symlink` and `--use-symbolic-links` server options (use `--skip-symbolic-links` [462] and `--symbolic-links` [462]).
- The `--sql-bin-update-same` server option.
- The `--warnings` server option (use `--log-warnings` [451]).
- The `--no-named-commands` option for `mysql` (use `--skip-named-commands`).
- The `--no-pager` option for `mysql` (use `--skip-pager`).
- The `--no-tee` option for `mysql` (use `--skip-tee`).
- The `--position` option for `mysqlbinlog` (use `--start-position`).
- The `--all` option for `mysqldump` (use `--create-options`).
- The `--first-slave` option for `mysqldump` (use `--lock-all-tables`).
- The `--config-file` option for `mysqld_multi` (use `--defaults-extra-file`).
- The `--set-variable=var_name=value` and `-O var_name=value` general-purpose options for setting program variables (use `--var_name=value`).
- The `--with-pstack` option for `configure` and the `--enable-pstack` option for `mysqld`.

1.4.1. Scalability Improvements

MySQL 5.5 modifications improve performance on SMP systems to increase scalability on multi-core systems. The changes affect InnoDB locking and memory management.

MySQL 5.5 incorporates changes in [InnoDB](#) that improve the performance of RW-locks by using atomic CPU instructions (on platforms where they are available), rather than less scalable mutexes. It is also possible for [InnoDB](#) memory allocation to be disabled and replaced by the normal `malloc` library, or by a different library that implements `malloc` such as `tcmalloc` on Linux or `mtalloc` on Solaris.

The reimplementaion of RW-locks requires atomic instructions. A status variable, `InnoDB_have_atomic_builtins` [601], shows whether the server was built with atomic instructions.

1.4.2. InnoDB I/O Subsystem Changes

MySQL 5.5 changes to the [InnoDB](#) I/O subsystem enable more effective use of available I/O capacity. The changes also provide more control over configuration of the I/O subsystem.

Background I/O Threads

[InnoDB](#) uses background threads to perform I/O for several kinds of activities, two of which are prefetching disk blocks and flushing dirty pages. Previously, [InnoDB](#) used only one thread each to perform these activities, but that can underutilize server capacity. MySQL 5.5 enables use of multiple background read and write threads, making it possible to read and write pages faster.

The patch makes the number of background I/O threads configurable using system variables: `innodb_read_io_threads` [1568] controls the number of threads to use for read prefetch requests. `innodb_write_io_threads` [1575] controls the number of threads to use for writing dirty pages from the buffer cache to disk. The default for both variables is 4.

The ability to increase the number of I/O threads can benefit systems that use multiple disks for [InnoDB](#). However, the type of I/O being done should be considered. On systems that use buffered writes rather than direct writes, increasing the write thread count higher than 1 might yield little benefit because writes will be quick already.

Adjustable I/O Rate

Previously, the number of input/output operations per second (IOPS) that [InnoDB](#) will perform was a compile-time parameter. The rate was chosen to prevent background I/O from exhausting server capacity and the compiled-in value of 100 reflected an assumption that the server can perform 100 IOPS. However, many modern systems can exceed this, so the value is low and unnecessarily restricts I/O utilization.

MySQL 5.5 exposes this I/O rate parameter as a system variable, `innodb_io_capacity` [1559]. This variable can be set at server startup, which enables higher values to be selected for systems capable of higher I/O rates. Having a higher I/O rate can help the server handle a higher rate of row changes because it may be able to increase dirty-page flushing, deleted-row removal, and application of changes to the insert buffer. The default value of `innodb_io_capacity` [1559] is 200. In general, you can increase the value as a function of the number of drives used for [InnoDB](#) I/O.

The ability to raise the I/O limit should be especially beneficial on platforms that support many IOPS. For example, systems that use multiple disks or solid-state disks for [InnoDB](#) are likely to benefit from the ability to control this parameter.

1.4.3. Diagnostic and Monitoring Capabilities

MySQL 5.5 provides improved access to execution and performance information. Diagnostic improvements include Performance Schema, Dtrace probes, expanded `SHOW ENGINE INNODB STATUS` output, Debug Sync, and a new status variable.

Performance Schema

Performance Schema is a feature for monitoring MySQL Server execution at a low level. See [Chapter 21, MySQL Performance Schema](#).

DTrace Support

The DTrace probes work on Solaris, Mac OS X, and FreeBSD. For information on using DTrace in MySQL, see [Section 5.4, “Tracing mysqld Using DTrace”](#).

Enhanced SHOW ENGINE INNODB STATUS Output

The output from `SHOW ENGINE INNODB STATUS` includes more information due to changes made for [InnoDB Plugin](#). A description of revisions to statement output follows.

A new `BACKGROUND THREAD` section has `srv_master_thread` lines that show work done by the main background thread.

```
-----
BACKGROUND THREAD
-----
srv_master_thread loops: 53 1_second, 44 sleeps, 5 10_second, 7 background,
 7 flush
srv_master_thread log flush and writes: 48
```

The `SEMAPHORES` section includes a line to show the number of spinlock rounds per OS wait for a mutex.

```
-----
SEMAPHORES
-----
...
Spin rounds per wait: 0.00 mutex, 20.00 RW-shared, 0.00 RW-excl
```

Debug Sync

The Debug Sync facility provides synchronization points for debugging, see [MySQL Internals: Test Synchronization](#).

New Status Variable

The `InnoDB_have_atomic_builtins` [601] status variable provides information about availability of atomic instructions; see [Section 1.4.1, “Scalability Improvements”](#).

1.4.4. Enhanced Solaris Support

MySQL 5.5 incorporates several modifications for improved operation of MySQL Server on Solaris:

- DTrace support for execution monitoring. See [Section 1.4.3, “Diagnostic and Monitoring Capabilities”](#).
- Atomic instructions, which are needed for the improvements to RW-locking described in [Section 1.4.1, “Scalability Improvements”](#). Atomic instructions now are supported for Sun Studio on SPARC and x86 platforms. This extends their previous availability (supported for `gcc` 4.1 and up on all platforms).
- The SMP improvements described in [Section 1.4.1, “Scalability Improvements”](#), were originally intended for x86 platforms. In MySQL 5.5, these also work on SPARC platforms. Also, Solaris optimizations have been implemented.
- Large page support is enhanced for recent SPARC platforms. Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a “super large pages” feature enables uses of pages up to 256MB. This feature can be enabled or disabled by using the `--super-large-pages` [462] or `--skip-super-large-pages` [462] option.
- Inline handling for `InnoDB` and processor instruction prefetching support, previously not enabled for builds created using Sun Studio, now are supported for that build environment.

1.5. MySQL Development History

This section describes the general MySQL development history, provides an overview about features that have been implemented in previous series and that are new in MySQL 5.5, the release series covered in this manual. The maturity level this release series is general availability. Information about maturity levels can be found in [Section 2.1.2.1, “Choosing Which Version of MySQL to Install”](#).

Before upgrading from one release series to the next, please see the notes in [Section 2.12.1, “Upgrading MySQL”](#).

The most requested features and the versions in which they were implemented are summarized in the following table.

Feature	MySQL Series
Unions	4.0
Subqueries	4.1
R-trees	4.1 (for the MyISAM storage engine)
Stored procedures and functions	5.0
Views	5.0
Cursors	5.0
XA transactions	5.0
Triggers	5.0 and 5.1
Event scheduler	5.1
Partitioning	5.1
Pluggable storage engine API	5.1
Plugin API	5.1
InnoDB Plugin	5.1
Row-based replication	5.1
Server log tables	5.1
Scalability and performance improvements	5.1 (with InnoDB Plugin)
DTrace support	5.5
InnoDB as default storage engine	5.5
Semisynchronous replication	5.5
SIGNAL/RESIGNAL support in stored routines	5.5
Performance Schema	5.5
Supplementary Unicode characters	5.5

1.6. MySQL Information Sources

This section lists sources of additional information that you may find helpful, such as the MySQL mailing lists and user forums, and Internet Relay Chat.

1.6.1. MySQL Mailing Lists

This section introduces the MySQL mailing lists and provides guidelines as to how the lists should be used. When you subscribe to a mailing list, you receive all postings to the list as email messages. You can also send your own questions and answers to the list.

To subscribe to or unsubscribe from any of the mailing lists described in this section, visit <http://lists.mysql.com/>. For most of them, you can select the regular version of the list where you get individual messages, or a digest version where you get one large message per day.

Please *do not* send messages about subscribing or unsubscribing to any of the mailing lists, because such messages are distributed automatically to thousands of other users.

Your local site may have many subscribers to a MySQL mailing list. If so, the site may have a local mailing list, so that messages sent from lists.mysql.com to your site are propagated to the local list. In such cases, please contact your system administrator to be added to or dropped from the local MySQL list.

To have traffic for a mailing list go to a separate mailbox in your mail program, set up a filter based on the message headers. You can use either the `List-ID:` or `Delivered-To:` headers to identify list messages.

The MySQL mailing lists are as follows:

- [announce](#)

The list for announcements of new versions of MySQL and related programs. This is a low-volume list to which all MySQL users should subscribe.

- [mysql](#)

The main list for general MySQL discussion. Please note that some topics are better discussed on the more-specialized lists. If you post to the wrong list, you may not get an answer.

- [bugs](#)

The list for people who want to stay informed about issues reported since the last release of MySQL or who want to be actively involved in the process of bug hunting and fixing. See [Section 1.7, “How to Report Bugs or Problems”](#).

- [internals](#)

The list for people who work on the MySQL code. This is also the forum for discussions on MySQL development and for posting patches.

- [mysqldoc](#)

The list for people who work on the MySQL documentation.

- [benchmarks](#)

The list for anyone interested in performance issues. Discussions concentrate on database performance (not limited to MySQL), but also include broader categories such as performance of the kernel, file system, disk system, and so on.

- [packagers](#)

The list for discussions on packaging and distributing MySQL. This is the forum used by distribution maintainers to exchange ideas on packaging MySQL and on ensuring that MySQL looks and feels as similar as possible on all supported platforms and operating systems.

- [java](#)

The list for discussions about the MySQL server and Java. It is mostly used to discuss JDBC drivers such as MySQL Connector/J.

- [win32](#)

The list for all topics concerning the MySQL software on Microsoft operating systems, such as Windows 9x, Me, NT, 2000, XP, and 2003.

- [myodbc](#)

The list for all topics concerning connecting to the MySQL server with ODBC.

- [gui-tools](#)

The list for all topics concerning MySQL graphical user interface tools such as MySQL Workbench.

- [cluster](#)

The list for discussion of MySQL Cluster.

- [dotnet](#)

The list for discussion of the MySQL server and the .NET platform. It is mostly related to MySQL Connector/Net.

- [plusplus](#)

The list for all topics concerning programming with the C++ API for MySQL.

- [perl](#)

The list for all topics concerning Perl support for MySQL with `DBD::mysql`.

If you're unable to get an answer to your questions from a MySQL mailing list or forum, one option is to purchase support from Oracle. This puts you in direct contact with MySQL developers.

The following MySQL mailing lists are in languages other than English. These lists are not operated by Oracle.

- [<mysql-france-subscribe@yahoogroups.com>](mailto:mysql-france-subscribe@yahoogroups.com)

A French mailing list.

- [<list@tinc.net>](mailto:list@tinc.net)

A Korean mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

- [<mysql-de-request@lists.4t2.com>](mailto:mysql-de-request@lists.4t2.com)

A German mailing list. To subscribe, email `subscribe mysql-de your@email.address` to this list. You can find information about this mailing list at <http://www.4t2.com/mysql/>.

- [<mysql-br-request@listas.linkway.com.br>](mailto:mysql-br-request@listas.linkway.com.br)

A Portuguese mailing list. To subscribe, email `subscribe mysql-br your@email.address` to this list.

- [<mysql-alta@elistas.net>](mailto:mysql-alta@elistas.net)

A Spanish mailing list. To subscribe, email subscribe_mysql_your@email.address to this list.

1.6.1.1. Guidelines for Using the Mailing Lists

Please do not post mail messages from your browser with HTML mode turned on. Many users do not read mail with a browser.

When you answer a question sent to a mailing list, if you consider your answer to have broad interest, you may want to post it to the list instead of replying directly to the individual who asked. Try to make your answer general enough that people other than the original poster may benefit from it. When you post to the list, please make sure that your answer is not a duplication of a previous answer.

Try to summarize the essential part of the question in your reply. Do not feel obliged to quote the entire original message.

When answers are sent to you individually and not to the mailing list, it is considered good etiquette to summarize the answers and send the summary to the mailing list so that others may have the benefit of responses you received that helped you solve your problem.

1.6.2. MySQL Community Support at the MySQL Forums

The forums at <http://forums.mysql.com> are an important community resource. Many forums are available, grouped into these general categories:

- Migration
- MySQL Usage
- MySQL Connectors
- Programming Languages
- Tools
- 3rd-Party Applications
- Storage Engines
- MySQL Technology
- SQL Standards
- Business

1.6.3. MySQL Community Support on Internet Relay Chat (IRC)

In addition to the various MySQL mailing lists and forums, you can find experienced community people on Internet Relay Chat (IRC). These are the best networks/channels currently known to us:

freenode (see <http://www.freenode.net/> for servers)

- [#mysql](#) is primarily for MySQL questions, but other database and general SQL questions are welcome. Questions about PHP, Perl, or C in combination with MySQL are also common.

If you are looking for IRC client software to connect to an IRC network, take a look at [xChat](#) (<http://www.xchat.org/>). X-Chat (GPL licensed) is available for Unix as well as for Windows platforms (a free Windows build of X-Chat is available at <http://www.silverex.org/download/>).

1.6.4. MySQL Enterprise

Oracle offers technical support in the form of MySQL Enterprise. For organizations that rely on the MySQL DBMS for business-critical production applications, MySQL Enterprise is a commercial subscription offering which includes:

- MySQL Enterprise Server
- MySQL Enterprise Monitor
- Monthly Rapid Updates and Quarterly Service Packs
- MySQL Knowledge Base
- 24x7 Technical and Consultative Support

MySQL Enterprise is available in multiple tiers, giving you the flexibility to choose the level of service that best matches your needs. For more information, see [MySQL Enterprise](#).

1.7. How to Report Bugs or Problems

Before posting a bug report about a problem, please try to verify that it is a bug and that it has not been reported already:

- Start by searching the MySQL online manual at <http://dev.mysql.com/doc/>. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. In addition, the release notes accompanying the manual can be particularly useful since it is quite possible that a newer version contains a solution to your problem. The release notes are available at the location just given for the manual.
- If you get a parse error for an SQL statement, please check your syntax closely. If you cannot find something wrong with it, it is extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual doesn't cover the syntax that you are using, MySQL Server doesn't support your statement.

If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server.

- For solutions to some common problems, see [Section C.5, "Problems and Common Errors"](#).
- Search the bugs database at <http://bugs.mysql.com/> to see whether the bug has been reported and fixed.
- Search the MySQL mailing list archives at <http://lists.mysql.com/>. See [Section 1.6.1, "MySQL Mailing Lists"](#).
- You can also use <http://www.mysql.com/search/> to search all the Web pages (including the manual) that are located at the MySQL Web site.

If you cannot find an answer in the manual, the bugs database, or the mailing list archives, check with your local MySQL expert. If you still cannot find an answer to your question, please use the following guidelines for reporting the bug.

The normal way to report bugs is to visit <http://bugs.mysql.com/>, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports.

Bugs posted in the bugs database at <http://bugs.mysql.com/> that are corrected for a given release are noted in the release notes.

If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to [<secalert_us@oracle.com>](mailto:secalert_us@oracle.com). Exception: Support customers should report all problems, including security bugs, to Oracle Support at <http://support.oracle.com/>.

To discuss problems with other users, you can use one of the MySQL mailing lists. [Section 1.6.1, “MySQL Mailing Lists”](#).

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release. This section helps you write your report correctly so that you do not waste your time doing things that may not help us much or at all. Please read this section carefully and make sure that all the information described here is included in your report.

Preferably, you should test the problem using the latest production or development version of MySQL Server before posting. Anyone should be able to repeat the bug by just using `mysql test <script_file` on your test case or by running the shell or Perl script that you include in the bug report. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself. The best reports are those that include a full example showing how to reproduce the bug or problem. See [MySQL Internals: Porting to Other Systems](#).

Remember that it is possible for us to respond to a report containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details do not matter. A good principle to follow is that if you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution that you use, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). These are highly relevant pieces of information, and in 99 cases out of 100, the bug report is useless without them. Very often we get questions like, “Why doesn't this work for me?” Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Errors often are platform-dependent. In such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If you compiled MySQL from source, remember also to provide information about your compiler if it is related to the problem. Often people find bugs in compilers and think the problem is MySQL-related. Most compilers are under development all the time and become better version by version. To determine whether your problem depends on your compiler, we need to know what compiler you used. Note that every compiling problem should be regarded as a bug and reported accordingly.

If a program produces an error message, it is very important to include the message in your report. If we try to search for something from the archives, it is better that the error message reported exactly matches the one that the program produces. (Even the lettercase should be observed.) It is best to copy and paste the entire error message into your report. You should never try to reproduce the message from memory.

If you have a problem with Connector/ODBC (MyODBC), please try to generate a trace file and send it with your report. See [Section 22.1.8.2, “How to Report Connector/ODBC Problems or Bugs”](#).

If your report includes long query output lines from test cases that you run with the `mysql` command-line tool, you can make the output more readable by using the `--vertical` [282] option or the `\G` statement terminator. The `EXPLAIN SELECT` example later in this section demonstrates the use of `\G`.

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 5.0.19). You can find out which version you are running by executing `mysqladmin version`. The `mysqladmin` program can be found in the `bin` directory under your MySQL installation directory.
- The manufacturer and model of the machine on which you experience the problem.
- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the “Help/About Windows” menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.
- If you are using a source distribution of the MySQL software, include the name and version number of the compiler that you used. If you have a binary distribution, include the distribution name.
- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.
- If `mysqld` died, you should also report the statement that crashed `mysqld`. You can usually get this information by running `mysqld` with query logging enabled, and then looking in the log after `mysqld` crashes. See [MySQL Internals: Porting to Other Systems](#).
- If a database table is related to the problem, include the output from the `SHOW CREATE TABLE db_name.tbl_name` statement in the bug report. This is a very easy way to get the definition of any table in a database. The information helps us create a situation matching the one that you have experienced.
- The SQL mode in effect when the problem occurred can be significant, so please report the value of the `sql_mode` [556] system variable. For stored procedure, stored function, and trigger objects, the relevant `sql_mode` [556] value is the one in effect when the object was created. For a stored procedure or function, the `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION` statement shows the relevant SQL mode, or you can query `INFORMATION_SCHEMA` for the information:

```
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.ROUTINES;
```

For triggers, you can use this statement:

```
SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.TRIGGERS;
```

- For performance-related bugs or problems with `SELECT` statements, you should always include the output of `EXPLAIN SELECT . . .`, and at least the number of rows that the `SELECT` statement produces. You should also include the output from `SHOW CREATE TABLE tbl_name` for each table that is involved. The more information you provide about your situation, the more likely it is that someone can help you.

The following is an example of a very good bug report. The statements are run using the `mysql` command-line tool. Note the use of the `\G` statement terminator for statements that would otherwise provide very long output lines that are difficult to read.

```
mysql> SHOW VARIABLES;
```

```
mysql> SHOW COLUMNS FROM ...\G
<output from SHOW COLUMNS>
mysql> EXPLAIN SELECT ...\G
<output from EXPLAIN>
mysql> FLUSH STATUS;
mysql> SELECT ...;
<A short version of the output from SELECT,
including the time taken to run the query>
mysql> SHOW STATUS;
<output from SHOW STATUS>
```

- If a bug or problem occurs while running `mysqld`, try to provide an input script that reproduces the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better. If you can make a reproducible test case, you should upload it to be attached to the bug report.

If you cannot provide a script, you should at least include the output from `mysqladmin variables extended-status processlist` in your report to provide some information on how your system is performing.

- If you cannot produce a test case with only a few rows, or if the test table is too big to be included in the bug report (more than 10 rows), you should dump your tables using `mysqldump` and create a `README` file that describes your problem. Create a compressed archive of your files using `tar` and `gzip` or `zip`. After you initiate a bug report for our bugs database at <http://bugs.mysql.com/>, click the Files tab in the bug report for instructions on uploading the archive to the bugs database.
- If you believe that the MySQL server produces a strange result from a statement, include not only the result, but also your opinion of what the result should be, and an explanation describing the basis for your opinion.
- When you provide an example of the problem, it is better to use the table names, variable names, and so forth that exist in your actual situation than to come up with new names. The problem could be related to the name of a table or variable. These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation, and it is by all means better for us. If you have data that you do not want to be visible to others in the bug report, you can upload it using the Files tab as previously described. If the information is really top secret and you do not want to show it even to us, go ahead and provide an example using other names, but please regard this as the last choice.
- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the `mysqld` server, as well as the options that you use to run any MySQL client programs. The options to programs such as `mysqld` and `mysql`, and to the `configure` script, are often key to resolving problems and are very relevant. It is never a bad idea to include them. If your problem involves a program written in a language such as Perl or PHP, please include the language processor's version number, as well as the version for any modules that the program uses. For example, if you have a Perl script that uses the `DBI` and `DBD: :mysql` modules, include the version numbers for Perl, `DBI`, and `DBD: :mysql`.
- If your question is related to the privilege system, please include the output of `mysqlaccess`, the output of `mysqladmin reload`, and all the error messages you get when trying to connect. When you test your privileges, you should first run `mysqlaccess`. After this, execute `mysqladmin reload version` and try to connect with the program that gives you trouble. `mysqlaccess` can be found in the `bin` directory under your MySQL installation directory.
- If you have a patch for a bug, do include it. But do not assume that the patch is all we need, or that we can use it, if you do not provide some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all. If so, we cannot use it.

If we cannot verify the exact purpose of the patch, we will not use it. Test cases help us here. Show that the patch handles all the situations that may occur. If we find a borderline case (even a rare one) where the patch will not work, it may be useless.

- Guesses about what the bug is, why it occurs, or what it depends on are usually wrong. Even the MySQL team cannot guess such things without first using a debugger to determine the real cause of a bug.
- Indicate in your bug report that you have checked the reference manual and mail archive so that others know you have tried to solve the problem yourself.
- If your data appears corrupt or you get errors when you access a particular table, first check your tables with `CHECK TABLE`. If that statement reports any errors:
 - The `InnoDB` crash recovery mechanism handles cleanup when the server is restarted after being killed, so in typical operation there is no need to “repair” tables. If you encounter an error with `InnoDB` tables, restart the server and see whether the problem persists, or whether the error affected only cached data in memory. If data is corrupted on disk, consider restarting with the `innodb_force_recovery [1558]` option enabled so that you can dump the affected tables.
 - For non-transactional tables, try to repair them with `REPAIR TABLE` or with `myisamchk`. See [Chapter 5, MySQL Server Administration](#).

If you are running Windows, please verify the value of `lower_case_table_names [518]` using the `SHOW VARIABLES LIKE 'lower_case_table_names'` statement. This variable affects how the server handles lettercase of database and table names. Its effect for a given value should be as described in [Section 9.2.2, “Identifier Case Sensitivity”](#).

- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the `.err` suffix in the name.) See [Section 5.2.2, “The Error Log”](#). Please include any relevant information from this file in your bug report. Normally `mysqld` should *never* crash a table if nothing killed it in the middle of an update. If you can find the cause of `mysqld` dying, it is much easier for us to provide you with a fix for the problem. See [Section C.5.1, “How to Determine What Is Causing a Problem”](#).
- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software thoroughly tested and should work without problems. We believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See [Section 2.1.2, “Choosing Which MySQL Distribution to Install”](#).

1.8. MySQL Standards Compliance

This section describes how MySQL relates to the ANSI/ISO SQL standards. MySQL Server has many extensions to the SQL standard, and here you can find out what they are and how to use them. You can also find information about functionality missing from MySQL Server, and how to work around some of the differences.

The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, “SQL:2003” refers to the standard released in 2003, and “SQL:2008” refers to the most recent version of the standard, released in 2008. We use the phrase “the SQL standard” or “standard SQL” to mean the current version of the SQL Standard at any time.

One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability. We are not afraid to add extensions to SQL or support for non-

SQL features if this greatly increases the usability of MySQL Server for a large segment of our user base. The `HANDLER` interface is an example of this strategy. See [Section 13.2.4, “HANDLER Syntax”](#).

We continue to support transactional and nontransactional databases to satisfy both mission-critical 24/7 usage and heavy Web or logging usage.

MySQL Server was originally designed to work with medium-sized databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-sized databases, but the code can also be compiled in a reduced version suitable for hand-held and embedded devices. The compact design of the MySQL server makes development in both directions possible without any conflicts in the source tree.

Currently, we are not targeting real-time support, although MySQL replication capabilities offer significant functionality.

MySQL supports high-availability database clustering using the `NDBCLUSTER` storage engine. See [Chapter 17, MySQL Cluster NDB 7.2](#).

We are implementing XML functionality beginning in MySQL 5.1, which supports most of the W3C XPath standard. We plan to increase support for XML as part of future MySQL development. See [Section 12.11, “XML Functions”](#).

1.8.1. What Standards MySQL Follows

Our aim is to support the full ANSI/ISO SQL standard, but without making concessions to speed and quality of the code.

ODBC levels 0 to 3.51.

1.8.2. Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differentially for different clients. This capability enables each application to tailor the server's operating mode to its own requirements.

SQL modes control aspects of server operation such as what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting `mysqld` with the `--sql-mode="mode_value"` [\[464\]](#) option. You can also change the mode at runtime by setting the `sql_mode` [\[556\]](#) system variable with a `SET [GLOBAL|SESSION] sql_mode='mode_value'` statement.

For more information on setting the SQL mode, see [Section 5.1.7, “Server SQL Modes”](#).

1.8.3. Running MySQL in ANSI Mode

You can tell `mysqld` to run in ANSI mode with the `--ansi` [\[434\]](#) startup option. Running the server in ANSI mode is the same as starting it with the following options:

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

You can achieve the same effect at runtime by executing these two statements:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SET GLOBAL sql_mode = 'ANSI';
```

You can see that setting the `sql_mode` [556] system variable to `'ANSI'` enables all SQL mode options that are relevant for ANSI mode as follows:

```
mysql> SET GLOBAL sql_mode='ANSI';
mysql> SELECT @@global.sql_mode;
-> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

Running the server in ANSI mode with `--ansi` [434] is not quite the same as setting the SQL mode to `'ANSI'`. The `--ansi` [434] option affects the SQL mode and also sets the transaction isolation level. Setting the SQL mode to `'ANSI'` has no effect on the isolation level.

See [Section 5.1.3, “Server Command Options”](#), and [Section 1.8.2, “Selecting SQL Modes”](#).

1.8.4. MySQL Extensions to Standard SQL

MySQL Server supports some extensions that you probably won't find in other SQL DBMSs. Be warned that if you use them, your code won't be portable to other SQL servers. In some cases, you can write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the “!” character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

MySQL Server maps each database to a directory under the MySQL data directory, and maps tables within a database to file names in the database directory. This has a few implications:

- Database and table names are case sensitive in MySQL Server on operating systems that have case-sensitive file names (such as most Unix systems). See [Section 9.2.2, “Identifier Case Sensitivity”](#).
- You can use standard system commands to back up, rename, move, delete, and copy tables that are managed by the `MyISAM` storage engine. For example, it is possible to rename a `MyISAM` table by renaming the `.MYD`, `.MYI`, and `.frm` files to which the table corresponds. (Nevertheless, it is preferable to use `RENAME TABLE` or `ALTER TABLE ... RENAME` and let the server rename the files.)

- General language syntax

- By default, strings can be enclosed by either “” or ‘’, not just by ‘’. (If the `ANSI_QUOTES` [612] SQL mode is enabled, strings can be enclosed only by “” and the server interprets strings enclosed by “” as identifiers.)
- “\” is the escape character in strings.

- In SQL statements, you can access tables from different databases with the `db_name.tbl_name` syntax. Some SQL servers provide the same functionality but call this `User space`. MySQL Server doesn't support tablespaces such as used in statements like this: `CREATE TABLE ralph.my_table ... IN my_tablespace`.
- SQL statement syntax
 - The `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
 - The `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE` statements. See [Section 13.1.10](#), “`CREATE DATABASE Syntax`”, [Section 13.1.21](#), “`DROP DATABASE Syntax`”, and [Section 13.1.1](#), “`ALTER DATABASE Syntax`”.
 - The `DO` statement.
 - `EXPLAIN SELECT` to obtain a description of how tables are processed by the query optimizer.
 - The `FLUSH` and `RESET` statements.
 - The `SET` statement. See [Section 13.7.4](#), “`SET Syntax`”.
 - The `SHOW` statement. See [Section 13.7.5](#), “`SHOW Syntax`”. The information produced by many of the MySQL-specific `SHOW` statements can be obtained in more standard fashion by using `SELECT` to query `INFORMATION_SCHEMA`. See [Chapter 20](#), [INFORMATION_SCHEMA Tables](#).
 - Use of `LOAD DATA INFILE`. In many cases, this syntax is compatible with Oracle's `LOAD DATA INFILE`. See [Section 13.2.6](#), “`LOAD DATA INFILE Syntax`”.
 - Use of `RENAME TABLE`. See [Section 13.1.32](#), “`RENAME TABLE Syntax`”.
 - Use of `REPLACE` instead of `DELETE` plus `INSERT`. See [Section 13.2.8](#), “`REPLACE Syntax`”.
 - Use of `CHANGE col_name`, `DROP col_name`, or `DROP INDEX`, `IGNORE` or `RENAME` in `ALTER TABLE` statements. Use of multiple `ADD`, `ALTER`, `DROP`, or `CHANGE` clauses in an `ALTER TABLE` statement. See [Section 13.1.7](#), “`ALTER TABLE Syntax`”.
 - Use of index names, indexes on a prefix of a column, and use of `INDEX` or `KEY` in `CREATE TABLE` statements. See [Section 13.1.17](#), “`CREATE TABLE Syntax`”.
 - Use of `TEMPORARY` or `IF NOT EXISTS` with `CREATE TABLE`.
 - Use of `IF EXISTS` with `DROP TABLE` and `DROP DATABASE`.
 - The capability of dropping multiple tables with a single `DROP TABLE` statement.
 - The `ORDER BY` and `LIMIT` clauses of the `UPDATE` and `DELETE` statements.
 - `INSERT INTO tbl_name SET col_name = ...` syntax.
 - The `DELAYED` clause of the `INSERT` and `REPLACE` statements.
 - The `LOW_PRIORITY` clause of the `INSERT`, `REPLACE`, `DELETE`, and `UPDATE` statements.
 - Use of `INTO OUTFILE` or `INTO DUMPFILE` in `SELECT` statements. See [Section 13.2.9](#), “`SELECT Syntax`”.
 - Options such as `STRAIGHT_JOIN` or `SQL_SMALL_RESULT` in `SELECT` statements.

- You don't need to name all selected columns in the `GROUP BY` clause. This gives better performance for some very specific, but quite normal queries. See [Section 12.16, “Functions and Modifiers for Use with GROUP BY Clauses”](#).
- You can specify `ASC` and `DESC` with `GROUP BY`, not just with `ORDER BY`.
- The ability to set variables in a statement with the `:=` assignment operator. See [Section 9.4, “User-Defined Variables”](#).
- Data types
 - The `MEDIUMINT`, `SET`, and `ENUM` data types, and the various `BLOB` and `TEXT` data types.
 - The `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED`, and `ZEROFILL` data type attributes.
- Functions and operators
 - To make it easier for users who migrate from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.
 - MySQL Server understands the `||` [1095] and `&&` [1094] operators to mean logical OR and AND, as in the C programming language. In MySQL Server, `||` [1095] and `OR` [1095] are synonyms, as are `&&` [1094] and `AND` [1094]. Because of this nice syntax, MySQL Server doesn't support the standard SQL `||` [1095] operator for string concatenation; use `CONCAT()` [1102] instead. Because `CONCAT()` [1102] takes any number of arguments, it is easy to convert use of the `||` [1095] operator to MySQL Server.
 - Use of `COUNT(DISTINCT value_list)` [1204] where `value_list` has more than one element.
 - String comparisons are case-insensitive by default, with sort ordering determined by the collation of the current character set, which is `latin1` (cp1252 West European) by default. If you don't like this, you should declare your columns with the `BINARY` attribute or use the `BINARY` cast, which causes comparisons to be done using the underlying character code values rather than a lexical ordering.
 - The `%` [1127] operator is a synonym for `MOD()` [1127]. That is, `N % M` is equivalent to `MOD(N,M)` [1127]. `%` [1127] is supported for C programmers and for compatibility with PostgreSQL.
 - The `=` [1089], `<>` [1090], `<=` [1090], `<` [1090], `>=` [1090], `>` [1090], `<<` [1183], `>>` [1183], `<=>` [1090], `AND` [1094], `OR` [1095], or `LIKE` [1111] operators may be used in expressions in the output column list (to the left of the `FROM`) in `SELECT` statements. For example:


```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```
 - The `LAST_INSERT_ID()` [1194] function returns the most recent `AUTO_INCREMENT` value. See [Section 12.14, “Information Functions”](#).
 - `LIKE` [1111] is permitted on numeric values.
 - The `REGEXP` [1114] and `NOT REGEXP` [1114] extended regular expression operators.
 - `CONCAT()` [1102] or `CHAR()` [1101] with one argument or more than two arguments. (In MySQL Server, these functions can take a variable number of arguments.)
 - The `BIT_COUNT()` [1183], `CASE` [1097], `ELT()` [1102], `FROM_DAYS()` [1140], `FORMAT()` [1103], `IF()` [1097], `PASSWORD()` [1188], `ENCRYPT()` [1187], `MD5()` [1187], `ENCODE()` [1187],

`DECODE()` [1186], `PERIOD_ADD()` [1144], `PERIOD_DIFF()` [1144], `TO_DAYS()` [1148], and `WEEKDAY()` [1152] functions.

- Use of `TRIM()` [1109] to trim substrings. Standard SQL supports removal of single characters only.
- The `GROUP BY` functions `STD()` [1206], `BIT_OR()` [1204], `BIT_AND()` [1204], `BIT_XOR()` [1204], and `GROUP_CONCAT()` [1205]. See Section 12.16, “Functions and Modifiers for Use with `GROUP BY` Clauses”.

1.8.5. MySQL Differences from Standard SQL

We try to make MySQL Server follow the ANSI SQL standard and the ODBC SQL standard, but MySQL Server performs operations differently in some cases:

- There are several differences between the MySQL and standard SQL privilege systems. For example, in MySQL, privileges for a table are not automatically revoked when you delete a table. You must explicitly issue a `REVOKE` statement to revoke privileges for a table. For more information, see Section 13.7.1.5, “`REVOKE` Syntax”.
- The `CAST()` [1168] function does not support cast to `REAL` or `BIGINT`. See Section 12.10, “Cast Functions and Operators”.

1.8.5.1. `SELECT INTO TABLE` Differences

MySQL Server doesn't support the `SELECT ... INTO TABLE` Sybase SQL extension. Instead, MySQL Server supports the `INSERT INTO ... SELECT` standard SQL syntax, which is basically the same thing. See Section 13.2.5.1, “`INSERT ... SELECT` Syntax”. For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Alternatively, you can use `SELECT ... INTO OUTFILE` or `CREATE TABLE ... SELECT`.

You can use `SELECT ... INTO` with user-defined variables. The same syntax can also be used inside stored routines using cursors and local variables. See Section 13.2.9.1, “`SELECT ... INTO` Syntax”.

1.8.5.2. `UPDATE` Differences

If you access a column from the table to be updated in an expression, `UPDATE` uses the current value of the column. The second assignment in the following statement sets `col2` to the current (updated) `col1` value, not the original `col1` value. The result is that `col1` and `col2` have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

1.8.5.3. Transaction and Atomic Operation Differences

MySQL Server (version 3.23-max and all versions 4.0 and above) supports transactions with the `InnoDB` transactional storage engine. `InnoDB` provides *full* ACID compliance. See Chapter 14, *Storage Engines*. For information about `InnoDB` differences from standard SQL with regard to treatment of transaction errors, see Section 14.3.13, “`InnoDB` Error Handling”.

The other nontransactional storage engines in MySQL Server (such as `MyISAM`) follow a different paradigm for data integrity called “atomic operations.” In transactional terms, `MyISAM` tables effectively always operate in `autocommit = 1` [484] mode. Atomic operations often offer comparable integrity with higher performance.

Because MySQL Server supports both paradigms, you can decide whether your applications are best served by the speed of atomic operations or the use of transactional features. This choice can be made on a per-table basis.

As noted, the tradeoff for transactional versus nontransactional storage engines lies mostly in performance. Transactional tables have significantly higher memory and disk space requirements, and more CPU overhead. On the other hand, transactional storage engines such as [InnoDB](#) also offer many significant features. MySQL Server's modular design enables the concurrent use of different storage engines to suit different requirements and deliver optimum performance in all situations.

Workarounds for Reliability with Non-Transactional Tables

But how do you use the features of MySQL Server to maintain integrity even with the nontransactional [MyISAM](#) tables, and how do these features compare with the transactional storage engines?

- If your applications are written in a way that is dependent on being able to call [ROLLBACK](#) rather than [COMMIT](#) in critical situations, transactions are more convenient. Transactions also ensure that unfinished updates or corrupting activities are not committed to the database; the server is given the opportunity to do an automatic rollback and your database is saved.

If you use nontransactional tables, you must resolve potential problems at the application level by including simple checks before updates and by running simple scripts that check the databases for inconsistencies and automatically repair or warn if such an inconsistency occurs. You can normally fix tables perfectly with no data integrity loss just by using the MySQL log or even adding one extra log.

- More often than not, critical transactional updates can be rewritten to be atomic. Generally speaking, all integrity problems that transactions solve can be done with [LOCK TABLES](#) or atomic updates, ensuring that there are no automatic aborts from the server, which is a common problem with transactional database systems.
- To be safe with MySQL Server, regardless of what kinds of tables you use, make regular backups and have binary logging turned on. It is always good to have backups, regardless of which database system you use.

The transactional paradigm has its advantages and disadvantages. Many users and application developers depend on the ease with which they can code around problems where an abort appears to be necessary, or is necessary. However, even if you are new to the atomic operations paradigm, or more familiar with transactions, do consider the speed benefit that nontransactional tables can offer on the order of three to five times the speed of the fastest and most optimally tuned transactional tables.

In situations where integrity is of highest importance, MySQL Server offers transaction-level reliability and integrity even for nontransactional tables. If you lock tables with [LOCK TABLES](#), all updates stall until integrity checks are made. If you obtain a [READ LOCAL](#) lock (as opposed to a write lock) for a table that enables concurrent inserts at the end of the table, reads are permitted, as are inserts by other clients. The newly inserted records are not be seen by the client that has the read lock until it releases the lock. With [INSERT DELAYED](#), you can write inserts that go into a local queue until the locks are released, without having the client wait for the insert to complete. See [Section 8.10.3, "Concurrent Inserts"](#), and [Section 13.2.5.2, "INSERT DELAYED Syntax"](#).

"Atomic," in the sense that we mean it, is nothing magical. It only means that you can be sure that while each specific update is running, no other user can interfere with it, and there can never be an automatic rollback (which can happen with transactional tables if you are not very careful). MySQL Server also guarantees that there are no dirty reads.

Following are some techniques for working with nontransactional tables:

- Loops that need transactions normally can be coded with the help of [LOCK TABLES](#), and you don't need cursors to update records on the fly.
- To avoid using [ROLLBACK](#), you can employ the following strategy:
 1. Use [LOCK TABLES](#) to lock all the tables you want to access.
 2. Test the conditions that must be true before performing the update.
 3. Update if the conditions are satisfied.
 4. Use [UNLOCK TABLES](#) to release your locks.



Note

This solution does not handle the situation when someone kills the threads in the middle of an update. In that case, all locks are released but some of the updates may not have been executed.

- You can also use functions to update records in a single operation, using the following techniques:
 - Modify columns relative to their current value. This makes the update correct even if another client has changed the column values in the meantime.
 - Update only those columns that actually have changed. This is a good database practice in general.
- When managing unique identifiers, you can avoid statements such as [LOCK TABLES](#) or [ROLLBACK](#) by using an [AUTO_INCREMENT](#) column and either the [LAST_INSERT_ID\(\) \[1194\]](#) SQL function or the [mysql_insert_id\(\)](#) C API function. See [Section 12.14, "Information Functions"](#), and [Section 22.8.3.37, "mysql_insert_id\(\)"](#).

For situations that require row-level locking, use [InnoDB](#) tables. Otherwise, with [MyISAM](#) tables, you can use a flag column in the table and do something like the following:

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID;
```

MySQL returns `1` for the number of affected rows if the row was found and `row_flag` wasn't `1` in the original row. You can think of this as though MySQL Server changed the preceding statement to:

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
```

1.8.5.4. Foreign Key Differences

The [InnoDB](#) storage engine supports checking of foreign key constraints, including [CASCADE](#), [ON DELETE](#), and [ON UPDATE](#). See [Section 14.3.5.4, "FOREIGN KEY Constraints"](#).

For storage engines other than [InnoDB](#), MySQL Server parses the [FOREIGN KEY](#) syntax in [CREATE TABLE](#) statements, but does not use or store it. In the future, the implementation will be extended to store this information in the table specification file so that it may be retrieved by [mysqldump](#) and ODBC. At a later stage, foreign key constraints will be implemented for [MyISAM](#) tables as well.

Foreign key enforcement offers several benefits to database developers:

- Assuming proper design of the relationships, foreign key constraints make it more difficult for a programmer to introduce an inconsistency into the database.
- Centralized checking of constraints by the database server makes it unnecessary to perform these checks on the application side. This eliminates the possibility that different applications may not all check the constraints in the same way.

- Using cascading updates and deletes can simplify the application code.
- Properly designed foreign key rules aid in documenting relationships between tables.

Do keep in mind that these benefits come at the cost of additional overhead for the database server to perform the necessary checks. Additional checking by the server affects performance, which for some applications may be sufficiently undesirable as to be avoided if possible. (Some major commercial applications have coded the foreign key logic at the application level for this reason.)

MySQL gives database developers the choice of which approach to use. If you don't need foreign keys and want to avoid the overhead associated with enforcing referential integrity, you can choose another storage engine instead, such as `MyISAM`. (For example, the `MyISAM` storage engine offers very fast performance for applications that perform only `INSERT` and `SELECT` operations. In this case, the table has no holes in the middle and the inserts can be performed concurrently with retrievals. See [Section 8.10.3, “Concurrent Inserts”](#).)

If you choose not to take advantage of referential integrity checks, keep the following considerations in mind:

- In the absence of server-side foreign key relationship checking, the application itself must handle relationship issues. For example, it must take care to insert rows into tables in the proper order, and to avoid creating orphaned child records. It must also be able to recover from errors that occur in the middle of multiple-record insert operations.
- If `ON DELETE` is the only referential integrity capability an application needs, you can achieve a similar effect as of MySQL Server 4.0 by using multiple-table `DELETE` statements to delete rows from many tables with a single statement. See [Section 13.2.2, “DELETE Syntax”](#).
- A workaround for the lack of `ON DELETE` is to add the appropriate `DELETE` statements to your application when you delete records from a table that has a foreign key. In practice, this is often as quick as using foreign keys and is more portable.

Be aware that the use of foreign keys can sometimes lead to problems:

- Foreign key support addresses many referential integrity issues, but it is still necessary to design key relationships carefully to avoid circular rules or incorrect combinations of cascading deletes.
- It is not uncommon for a DBA to create a topology of relationships that makes it difficult to restore individual tables from a backup. (MySQL alleviates this difficulty by enabling you to temporarily disable foreign key checks when reloading a table that depends on other tables. See [Section 14.3.5.4, “FOREIGN KEY Constraints”](#). As of MySQL 4.1.1, `mysqldump` generates dump files that take advantage of this capability automatically when they are reloaded.)

Foreign keys in SQL are used to check and enforce referential integrity, not to join tables. If you want to get results from multiple tables from a `SELECT` statement, you do this by performing a join between them:

```
SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

See [Section 13.2.9.2, “JOIN Syntax”](#), and [Section 3.6.6, “Using Foreign Keys”](#).

The `FOREIGN KEY` syntax without `ON DELETE ...` is often used by ODBC applications to produce automatic `WHERE` clauses.

1.8.5.5. '--' as the Start of a Comment

Standard SQL uses the C syntax `/* this is a comment */` for comments, and MySQL Server supports this syntax as well. MySQL also support extensions to this syntax that enable MySQL-specific SQL to be embedded in the comment, as described in [Section 9.6, “Comment Syntax”](#).

Standard SQL uses “--” as a start-comment sequence. MySQL Server uses “#” as the start comment character. MySQL Server 3.23.3 and up also supports a variant of the “--” comment style. That is, the “--” start-comment sequence must be followed by a space (or by a control character such as a newline). The space is required to prevent problems with automatically generated SQL queries that use constructs such as the following, where we automatically insert the value of the payment for `payment`:

```
UPDATE account SET credit=credit-payment
```

Consider about what happens if `payment` has a negative value such as `-1`:

```
UPDATE account SET credit=credit--1
```

`credit--1` is a valid expression in SQL, but “--” is interpreted as the start of a comment, part of the expression is discarded. The result is a statement that has a completely different meaning than intended:

```
UPDATE account SET credit=credit
```

The statement produces no change in value at all. This illustrates that permitting comments to start with “--” can have serious consequences.

Using our implementation requires a space following the “--” for it to be recognized as a start-comment sequence in MySQL Server 3.23.3 and newer. Therefore, `credit--1` is safe to use.

Another safe feature is that the `mysql` command-line client ignores lines that start with “--”.

The following information is relevant only if you are running a MySQL version earlier than 3.23.3:

If you have an SQL script in a text file that contains “--” comments, you should use the `replace` utility as follows to convert the comments to use “#” characters before executing the script:

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \
| mysql db_name
```

That is safer than executing the script in the usual way:

```
shell> mysql db_name < text-file-with-funny-comments.sql
```

You can also edit the script file “in place” to change the “--” comments to “#” comments:

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

Change them back with this command:

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

See [Section 4.8.2, “replace — A String-Replacement Utility”](#).

1.8.6. How MySQL Deals with Constraints

MySQL enables you to work both with transactional tables that permit rollback and with nontransactional tables that do not. Because of this, constraint handling is a bit different in MySQL than in other DBMSs. We must handle the case when you have inserted or updated a lot of rows in a nontransactional table for which changes cannot be rolled back when an error occurs.

The basic philosophy is that MySQL Server tries to produce an error for anything that it can detect while parsing a statement to be executed, and tries to recover from any errors that occur while executing the statement. We do this in most cases, but not yet for all.

The options MySQL has when an error occurs are to stop the statement in the middle or to recover as well as possible from the problem and continue. By default, the server follows the latter course. This means, for example, that the server may coerce invalid values to the closest valid values.

Several SQL mode options are available to provide greater control over handling of bad data values and whether to continue statement execution or abort when errors occur. Using these options, you can configure MySQL Server to act in a more traditional fashion that is like other DBMSs that reject improper input. The SQL mode can be set globally at server startup to affect all clients. Individual clients can set the SQL mode at runtime, which enables each client to select the behavior most appropriate for its requirements. See [Section 5.1.7, “Server SQL Modes”](#).

The following sections describe how MySQL Server handles different types of constraints.

1.8.6.1. PRIMARY KEY and UNIQUE Index Constraints

Normally, errors occur for data-change statements (such as `INSERT` or `UPDATE`) that would violate primary-key, unique-key, or foreign-key constraints. If you are using a transactional storage engine such as `InnoDB`, MySQL automatically rolls back the statement. If you are using a nontransactional storage engine, MySQL stops processing the statement at the row for which the error occurred and leaves any remaining rows unprocessed.

MySQL supports an `IGNORE` keyword for `INSERT`, `UPDATE`, and so forth. If you use it, MySQL ignores primary-key or unique-key violations and continues processing with the next row. See the section for the statement that you are using ([Section 13.2.5, “INSERT Syntax”](#), [Section 13.2.11, “UPDATE Syntax”](#), and so forth).

You can get information about the number of rows actually inserted or updated with the `mysql_info()` C API function. You can also use the `SHOW WARNINGS` statement. See [Section 22.8.3.35, “mysql_info\(\)”](#), and [Section 13.7.5.41, “SHOW WARNINGS Syntax”](#).

Currently, only `InnoDB` tables support foreign keys. See [Section 14.3.5.4, “FOREIGN KEY Constraints”](#).

1.8.6.2. Constraints on Invalid Data

By default, MySQL is forgiving of invalid or improper data values and coerces them to valid values for data entry. However, you can change the server SQL mode to select more traditional treatment of bad values such that the server rejects them and aborts the statement in which they occur. See [Section 5.1.7, “Server SQL Modes”](#).

This section describes the default (forgiving) behavior of MySQL, as well as the strict SQL mode and how it differs.

If you are not using strict mode, then whenever you insert an “incorrect” value into a column, such as a `NULL` into a `NOT NULL` column or a too-large numeric value into a numeric column, MySQL sets the column to the “best possible value” instead of producing an error: The following rules describe in more detail how this works:

- If you try to store an out of range value into a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value, whichever is closest to the invalid value.
- For strings, MySQL stores either the empty string or as much of the string as can be stored in the column.
- If you try to store a string that doesn't start with a number into a numeric column, MySQL Server stores 0.

- Invalid values for `ENUM` and `SET` columns are handled as described in [Section 1.8.6.3, “ENUM and SET Constraints”](#).
- MySQL enables you to store certain incorrect date values into `DATE` and `DATETIME` columns (such as `'2000-02-31'` or `'2000-02-00'`). The idea is that it is not the job of the SQL server to validate dates. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special “zero” date value `'0000-00-00'` is stored in the column instead.
- If you try to store `NULL` into a column that doesn't take `NULL` values, an error occurs for single-row `INSERT` statements. For multiple-row `INSERT` statements or for `INSERT INTO ... SELECT` statements, MySQL Server stores the implicit default value for the column data type. In general, this is `0` for numeric types, the empty string (`' '`) for string types, and the “zero” value for date and time types. Implicit default values are discussed in [Section 11.5, “Data Type Default Values”](#).
- If an `INSERT` statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit `DEFAULT` clause. If the definition has no such `DEFAULT` clause, MySQL inserts the implicit default value for the column data type.

The reason for using the preceding rules in nonstrict mode is that we can't check these conditions until the statement has begun executing. We can't just roll back if we encounter a problem after updating a few rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in this case, the update would be “half done,” which is probably the worst possible scenario. In this case, it is better to “do the best you can” and then continue as if nothing happened.

In MySQL 5.0.2 and up, you can select stricter treatment of input values by using the `STRICT_TRANS_TABLES` [\[617\]](#) or `STRICT_ALL_TABLES` [\[616\]](#) SQL modes:

```
SET sql_mode = 'STRICT_TRANS_TABLES';
SET sql_mode = 'STRICT_ALL_TABLES';
```

`STRICT_TRANS_TABLES` [\[617\]](#) enables strict mode for transactional storage engines, and also to some extent for nontransactional engines. It works like this:

- For transactional storage engines, bad data values occurring anywhere in a statement cause the statement to abort and roll back.
- For nontransactional storage engines, a statement aborts if the error occurs in the first row to be inserted or updated. (When the error occurs in the first row, the statement can be aborted to leave the table unchanged, just as for a transactional table.) Errors in rows after the first do not abort the statement, because the table has already been changed by the first row. Instead, bad data values are adjusted and result in warnings rather than errors. In other words, with `STRICT_TRANS_TABLES` [\[617\]](#), a wrong value causes MySQL to roll back all updates done so far, if that can be done without changing the table. But once the table has been changed, further errors result in adjustments and warnings.

For even stricter checking, enable `STRICT_ALL_TABLES` [\[616\]](#). This is the same as `STRICT_TRANS_TABLES` [\[617\]](#) except that for nontransactional storage engines, errors abort the statement even for bad data in rows following the first row. This means that if an error occurs partway through a multiple-row insert or update for a nontransactional table, a partial update results. Earlier rows are inserted or updated, but those from the point of the error on are not. To avoid this for nontransactional tables, either use single-row statements or else use `STRICT_TRANS_TABLES` [\[617\]](#) if conversion warnings rather than errors are acceptable. To avoid problems in the first place, do not use MySQL to check column content. It is safest (and often faster) to let the application ensure that it passes only valid values to the database.

With either of the strict mode options, you can cause errors to be treated as warnings by using `INSERT IGNORE` or `UPDATE IGNORE` rather than `INSERT` or `UPDATE` without `IGNORE`.

1.8.6.3. ENUM and SET Constraints

ENUM and SET columns provide an efficient way to define columns that can contain only a given set of values. See [Section 11.4.4, “The ENUM Type”](#), and [Section 11.4.5, “The SET Type”](#). However, before MySQL 5.0.2, ENUM and SET columns do not provide true constraints on entry of invalid data:

- ENUM columns always have a default value. If you specify no default value, then it is NULL for columns that can have NULL, otherwise it is the first enumeration value in the column definition.
- If you insert an incorrect value into an ENUM column or if you force a value into an ENUM column with IGNORE, it is set to the reserved enumeration value of 0, which is displayed as an empty string in string context.
- If you insert an incorrect value into a SET column, the incorrect value is ignored. For example, if the column can contain the values 'a', 'b', and 'c', an attempt to assign 'a,x,b,y' results in a value of 'a,b'.

As of MySQL 5.0.2, you can configure the server to use strict SQL mode. See [Section 5.1.7, “Server SQL Modes”](#). With strict mode enabled, the definition of a ENUM or SET column does act as a constraint on values entered into the column. An error occurs for values that do not satisfy these conditions:

- An ENUM value must be one of those listed in the column definition, or the internal numeric equivalent thereof. The value cannot be the error value (that is, 0 or the empty string). For a column defined as ENUM('a', 'b', 'c'), values such as '', 'd', or 'ax' are invalid and are rejected.
- A SET value must be the empty string or a value consisting only of the values listed in the column definition separated by commas. For a column defined as SET('a', 'b', 'c'), values such as 'd' or 'a,b,c,d' are invalid and are rejected.

Errors for invalid values can be suppressed in strict mode if you use INSERT IGNORE or UPDATE IGNORE. In this case, a warning is generated rather than an error. For ENUM, the value is inserted as the error member (0). For SET, the value is inserted as given except that any invalid substrings are deleted. For example, 'a,x,b,y' results in a value of 'a,b'.

1.9. Credits

The following sections list developers, contributors, and supporters that have helped to make MySQL what it is today.

1.9.1. Contributors to MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the [MySQL server](#) and the [MySQL manual](#), we wish to recognize those who have made contributions of one kind or another to the [MySQL distribution](#). Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola <qwertg@mbx.vol.it> or <qwertg@tin.it>

The initial port to Win32/NT.

- Per Eric Olsson

For constructive criticism and real testing of the dynamic record format.

- Irena Pancirov <irena@mail.yacc.it>

Win32 port with Borland compiler. [mysqlshutdown.exe](#) and [mysqlwatch.exe](#).

- David J. Hughes

For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with `mSQL`, but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. `mysqladmin` and `mysql` client are programs that were largely influenced by their `mSQL` counterparts. We have put a lot of effort into making the MySQL syntax a superset of `mSQL`. Many of the API's ideas are borrowed from `mSQL` to make it easy to port free `mSQL` programs to the MySQL API. The MySQL software doesn't contain any code from `mSQL`. Two files in the distribution (`client/insert_test.c` and `client/select_test.c`) are based on the corresponding (noncopyrighted) files in the `mSQL` distribution, but are modified as examples showing the changes necessary to convert code from `mSQL` to MySQL Server. (`mSQL` is copyrighted David J. Hughes.)

- Patrick Lynch

For helping us acquire <http://www.mysql.com/>.

- Fred Lindberg

For setting up gmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko <igor@frog.kiev.ua>

`mysqldump` (previously `msqldump`, but ported and enhanced by Monty).

- Yuri Dario

For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

Author of `mysqlhotcopy`.

- Zarko Mocnik <zarko.mocnik@dem.si>

Sorting for Slovenian language.

- "TAMITO" <tommy@valley.ne.jp>

The `_MB` character set macros and the `ujis` and `sjis` character sets.

- Joshua Chamas <joshua@chamas.com>

Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier <Yves.Carlier@rug.ac.be>

`mysqlaccess`, a program to show the access rights for a user.

- Rhys Jones <rhys@wales.com> (And GWE Technologies Limited)

For one of the early JDBC drivers.

- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>

Further development of one of the early JDBC drivers and other MySQL-related Java tools.

- James Cooper <pixel@organic.com>

For setting up a searchable mailing list archive at his site.

- Rick Mehalick <Rick_Mehalick@i-o.com>

For [xmysql](#), a graphical X client for MySQL Server.

- Doug Sisk <sisk@wix.com>

For providing RPM packages of MySQL for Red Hat Linux.

- Diemand Alexander V. <axeld@vial.ethz.ch>

For providing RPM packages of MySQL for Red Hat Linux-Alpha.

- Antoni Pamies Olive <toni@readysoft.es>

For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

- Jay Bloodworth <jay@pathways.sde.state.sc.us>

For providing RPM versions for MySQL 3.21.

- David Sacerdote <davids@secnet.com>

Ideas for secure checking of DNS host names.

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

Some support for Chinese(BIG5) characters.

- Wei He <hewei@mail.ied.ac.cn>

A lot of functionality for the Chinese(GBK) character set.

- Jan Pazdziora <adelton@fi.muni.cz>

Czech sorting order.

- Zeev Suraski <bourbon@netvision.net.il>

[FROM_UNIXTIME\(\)](#) [1140] time formatting, [ENCRYPT\(\)](#) [1187] functions, and [bison](#) advisor. Active mailing list member.

- Luuk de Boer <luuk@wxs.nl>

Ported (and extended) the benchmark suite to [DBI/DBD](#). Have been of great help with [crash-me](#) and running benchmarks. Some new date functions. The [mysql_setpermission](#) script.

- Alexis Mikhailov <root@medinf.chuvashia.su>

User-defined functions (UDFs); [CREATE FUNCTION](#) and [DROP FUNCTION](#).

- Andreas F. Bobak <bobak@relog.ch>

The [AGGREGATE](#) extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>
The `libmysql.dll` library.
- James Pereria <jpereira@iafrica.com>
Mysqmanager, a Win32 GUI tool for administering MySQL Servers.
- Curt Sampson <cjs@portal.ca>
Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.
- Martin Ramsch <m.ramsch@computer.org>
Examples in the MySQL Tutorial.
- Steve Harvey
For making `mysqlaccess` more secure.
- Konark IA-64 Centre of Persistent Systems Private Limited
<http://www.pspl.co.in/konark/>. Help with the Win64 port of the MySQL server.
- Albert Chin-A-Young.
Configure updates for Tru64, large file support and better TCP wrappers support.
- John Birrell
Emulation of `pthread_mutex()` for OS/2.
- Benjamin Pflugmann
Extended `MERGE` tables to handle `INSERTS`. Active member on the MySQL mailing lists.
- Jocelyn Fournier
Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).
- Marc Liyanage
Maintaining the Mac OS X packages and providing invaluable feedback on how to create Mac OS X packages.
- Robert Rutherford
Providing invaluable information and feedback about the QNX port.
- Previous developers of NDB Cluster
Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to mention here. Notable name is Ataulah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architectural foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.
- Google Inc.

We wish to recognize Google Inc. for contributions to the MySQL distribution: Mark Callaghan's SMP Performance patches and other patches.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, <jehamby@lightside>, <psmith@BayNetworks.com>, <duane@connect.com.au>, Ted Deppner <ted@psyber.com>, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch <dkoch@amcity.com>

Irix setup.

- Luuk de Boer <luuk@wxs.nl>

Benchmark questions.

- Tim Sailer <tps@users.buoy.com>

DBD: :mysql questions.

- Boyd Lynn Gerber <gerberb@zenez.com>

SCO-related questions.

- Richard Mehalick <RM186061@shellus.com>

xmysql-related questions and basic installation questions.

- Zeev Suraski <bourbon@netvision.net.il>

Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

- Francesc Guasch <frankie@citel.upc.es>

General questions.

- Jonathan J Smith <jsmith@wtp.net>

Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar <sklar@student.net>

Using MySQL from PHP and Perl.

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

Is flexible and can handle Linux and perhaps HP-UX.

- John Lyon <jlyon@imag.net>

Questions about installing MySQL on Linux systems, using either .rpm files or compiling from source.

- Lorvid Ltd. <lorvid@WOLFENET.com>

Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

DBD, Linux, some SQL syntax questions.

1.9.2. Documenters and translators

The following people have helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded (<http://mysql.hitstar.com/>) versions were based. [Personal home page at linuxdb.yeah.net](#).

- Jay Flaherty <fty@mediapulse.com>

Big parts of the Perl DBI/DBD section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

French error messages.

- Petr Snajdr, <snajdr@pvt.net>

Czech error messages.

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

Polish error messages.

- Miguel Angel Fernandez Roiz

Spanish error messages.

- Roy-Magne Mo <rmo@www.hivolda.no>
Norwegian error messages and testing of MySQL 3.21.xx.
- Timur I. Bakeyev <root@timur.tatarstan.ru>
Russian error messages.
- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>
Italian error messages.
- Dirk Munzinger <dirk@trinity.saar.de>
German error messages.
- Billik Stefan <billik@sun.uniag.sk>
Slovak error messages.
- Stefan Saroiu <tzoompy@cs.washington.edu>
Romanian error messages.
- Peter Feher
Hungarian error messages.
- Roberto M. Serqueira
Portuguese error messages.
- Carsten H. Pedersen
Danish error messages.
- Arjen Lentz
Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

1.9.3. Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We cannot list every possible package here because the list would then be way to hard to maintain. For other packages, please refer to the software portal at <http://solutions.mysql.com/software/>.

- Tim Bunce, Alligator Descartes
For the `DBD` (Perl) interface.
- Andreas Koenig <a.koenig@mind.de>
For the Perl interface for MySQL Server.
- Jochen Wiedmann <wiedmann@neckar-alb.de>
For maintaining the Perl `DBD::mysql` module.

- Eugene Chan <eugene@acenet.com.sg>
For porting PHP for MySQL Server.
- Georg Richter
MySQL 4.1 testing and bug hunting. New PHP 5.0 [mysqli](#) extension (API) for use with MySQL 4.1 and up.
- Giovanni Maruzzelli <maruzz@matrice.it>
For porting iODBC (Unix ODBC).
- Xavier Leroy <Xavier.Leroy@inria.fr>
The author of LinuxThreads (used by the MySQL Server on Linux).

1.9.4. Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation
From whom we got an excellent compiler ([gcc](#)), an excellent debugger ([gdb](#) and the [libc](#) library (from which we have borrowed [strto.c](#) to get some code working in Linux).
- Free Software Foundation & The XEmacs development team
For a really great editor/environment.
- Julian Seward
Author of [valgrind](#), an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.
- Dorothea Lütkehaus and Andreas Zeller
For [DDD](#) (The Data Display Debugger) which is an excellent graphical front end to [gdb](#).

1.9.5. Supporters of MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the [MySQL server](#) and the [MySQL manual](#), we wish to recognize the following companies, which helped us finance the development of the [MySQL server](#), such as by paying us for developing a new feature or giving us hardware for development of the [MySQL server](#).

- VA Linux / Andover.net
Funded replication.
- NuSphere
Editing of the MySQL manual.
- Stork Design studio
The MySQL Web site in use between 1998-2000.

- Intel
Contributed to development on Windows and Linux platforms.
- Compaq
Contributed to Development on Linux/Alpha.
- SWSOft
Development on the embedded `mysqld` version.
- FutureQuest
The `--skip-show-database` [\[463\]](#) option.

Chapter 2. Installing and Upgrading MySQL

Table of Contents

2.1. General Installation Guidance	47
2.1.1. Operating Systems Supported by MySQL Community Server	48
2.1.2. Choosing Which MySQL Distribution to Install	49
2.1.3. How to Get MySQL	52
2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG	52
2.1.5. Installation Layouts	55
2.1.6. Compiler-Specific Build Characteristics	56
2.2. Installing MySQL from Generic Binaries on Unix/Linux	56
2.3. Installing MySQL on Microsoft Windows	58
2.3.1. MySQL Installation Layout on Microsoft Windows	60
2.3.2. Choosing An Installation Package	61
2.3.3. Installing MySQL on Microsoft Windows Using MySQL Installer	61
2.3.4. Installing MySQL on Microsoft Windows Using an MSI Package	78
2.3.5. MySQL Notifier for Microsoft Windows	85
2.3.6. MySQL Server Instance Configuration Wizard	89
2.3.7. Installing MySQL on Microsoft Windows Using a <code>noinstall</code> Zip Archive	105
2.3.8. Troubleshooting a Microsoft Windows MySQL Server Installation	113
2.3.9. Upgrading MySQL on Windows	114
2.3.10. Windows Postinstallation Procedures	115
2.4. Installing MySQL on Mac OS X	117
2.4.1. General Notes on Installing MySQL on Mac OS X	117
2.4.2. Installing MySQL on Mac OS X Using Native Packages	119
2.4.3. Installing the MySQL Startup Item	121
2.4.4. Installing and Using the MySQL Preference Pane	124
2.4.5. Using the Bundled MySQL on Mac OS X Server	126
2.5. Installing MySQL on Linux	127
2.5.1. Installing MySQL from RPM Packages on Linux	127
2.5.2. Installing MySQL on Linux using Native Package Manager	131
2.6. Installing MySQL Using Unbreakable Linux Network (ULN)	135
2.7. Installing MySQL on Solaris and OpenSolaris	135
2.7.1. Installing MySQL on Solaris using a Solaris <code>PKG</code>	136
2.7.2. Installing MySQL on OpenSolaris using IPS	137
2.8. Installing MySQL on HP-UX	138
2.8.1. General Notes on Installing MySQL on HP-UX	138
2.8.2. Installing MySQL on HP-UX using DEPOT	138
2.9. Installing MySQL on FreeBSD	139
2.10. Installing MySQL from Source	140
2.10.1. MySQL Layout for Source Installation	142
2.10.2. Installing MySQL from a Standard Source Distribution	142
2.10.3. Installing MySQL from a Development Source Tree	146
2.10.4. MySQL Source-Configuration Options	147
2.10.5. Dealing with Problems Compiling MySQL	158
2.10.6. MySQL Configuration and Third-Party Tools	159
2.11. Postinstallation Setup and Testing	159
2.11.1. Unix Postinstallation Procedures	160
2.11.2. Securing the Initial MySQL Accounts	171
2.12. Upgrading or Downgrading MySQL	175
2.12.1. Upgrading MySQL	175

2.12.2. Downgrading MySQL	187
2.12.3. Checking Whether Tables or Indexes Must Be Rebuilt	188
2.12.4. Rebuilding or Repairing Tables or Indexes	190
2.12.5. Copying MySQL Databases to Another Machine	191
2.13. Environment Variables	192
2.14. Perl Installation Notes	194
2.14.1. Installing Perl on Unix	194
2.14.2. Installing ActiveState Perl on Windows	195
2.14.3. Problems Using the Perl DBI/DBD Interface	195

This chapter describes how to obtain and install MySQL. A summary of the procedure follows and later sections provide the details. If you plan to upgrade an existing version of MySQL to a newer version rather than install MySQL for the first time, see [Section 2.12.1, “Upgrading MySQL”](#), for information about upgrade procedures and about issues that you should consider before upgrading.

If you are interested in migrating to MySQL from another database system, you may wish to read [Section B.8, “MySQL 5.5 FAQ: Migration”](#), which contains answers to some common questions concerning migration issues.

1. Determine whether MySQL runs and is supported on your platform.

Please note that not all platforms are equally suitable for running MySQL, and that not all platforms on which MySQL is known to run are officially supported by Oracle Corporation:

2. Choose which distribution to install.

Several versions of MySQL are available, and most are available in several distribution formats. You can choose from pre-packaged distributions containing binary (precompiled) programs or source code. When in doubt, use a binary distribution. We also provide public access to our current source tree for those who want to see our most recent developments and help us test new code. To determine which version and type of distribution you should use, see [Section 2.1.2, “Choosing Which MySQL Distribution to Install”](#).

3. Download the distribution that you want to install.

For instructions, see [Section 2.1.3, “How to Get MySQL”](#). To verify the integrity of the distribution, use the instructions in [Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”](#).

4. Install the distribution.

To install MySQL from a binary distribution, use the instructions in [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#).

To install MySQL from a source distribution or from the current development source tree, use the instructions in [Section 2.10, “Installing MySQL from Source”](#).

5. Perform any necessary postinstallation setup.

After installing MySQL, read [Section 2.11, “Postinstallation Setup and Testing”](#). This section contains important information about making sure the MySQL server is working properly. It also describes how to secure the initial MySQL user accounts, *which have no passwords* until you assign passwords. The section applies whether you install MySQL using a binary or source distribution.

6. If you want to run the MySQL benchmark scripts, Perl support for MySQL must be available. See [Section 2.14, “Perl Installation Notes”](#).

Instructions for installing MySQL on different platforms and environments is available on a platform by platform basis:

- **Unix, Linux, FreeBSD**

For instructions on installing MySQL on most Linux and Unix platforms using a generic binary (for example, a `.tar.gz` package), see [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#).

For information on building MySQL entirely from the source code distributions or the source code repositories, see [Section 2.10, “Installing MySQL from Source”](#)

For specific platform help on installation, configuration, and building from source see the corresponding platform section:

- Linux, including notes on distribution specific methods, see [Section 2.5, “Installing MySQL on Linux”](#).
- Solaris and OpenSolaris, including PKG and IPS formats, see [Section 2.7, “Installing MySQL on Solaris and OpenSolaris”](#).
- IBM AIX, see [Section 2.7, “Installing MySQL on Solaris and OpenSolaris”](#).
- Hewlett-Packard HP-UX, including the DEPOT package format, see [Section 2.8, “Installing MySQL on HP-UX”](#).
- FreeBSD, see [Section 2.9, “Installing MySQL on FreeBSD”](#).

- **Microsoft Windows**

For instructions on installing MySQL on Microsoft Windows, using either a Zipped binary or an MSI package, see [Section 2.3, “Installing MySQL on Microsoft Windows”](#).

For information on using the MySQL Server Instance Config Wizard, see [Section 2.3.6, “MySQL Server Instance Configuration Wizard”](#).

For details and instructions on building MySQL from source code using Microsoft Visual Studio, see [Section 2.10, “Installing MySQL from Source”](#).

- **Mac OS X**

For installation on Mac OS X, including using both the binary package and native PKG formats, see [Section 2.4, “Installing MySQL on Mac OS X”](#).

For information on making use of the MySQL Startup Item to automatically start and stop MySQL, see [Section 2.4.3, “Installing the MySQL Startup Item”](#).

For information on the MySQL Preference Pane, see [Section 2.4.4, “Installing and Using the MySQL Preference Pane”](#).

2.1. General Installation Guidance

The immediately following sections contain the information necessary to choose, download, and verify your distribution. The instructions in later sections of the chapter describe how to install the distribution that you choose. For binary distributions, see the instructions at [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#) or the corresponding section for your platform if available. To build MySQL from source, use the instructions in [Section 2.10, “Installing MySQL from Source”](#).

2.1.1. Operating Systems Supported by MySQL Community Server

This section lists the operating systems on which MySQL Community Server is known to run.



Important

Oracle Corporation does not necessarily provide official support for all the platforms listed in this section. For information about those platforms that are officially supported, see <http://www.mysql.com/support/supportedplatforms.html> on the MySQL Web site.

We use `CMake`, so it is possible to port MySQL to all modern systems that have a C++ compiler and a working implementation of POSIX threads. (Thread support is needed for the server. To compile only the client code, the only requirement is a C++ compiler.)

MySQL has been reported to compile successfully on the following combinations of operating system and thread package.

- FreeBSD 5.x and up with native threads. See [Section 2.9, “Installing MySQL on FreeBSD”](#).
- HP-UX 11.x with the native threads. See [Section 2.8, “Installing MySQL on HP-UX”](#).
- Linux. Builds on all recent Linux distributions based on the 2.6 kernel. See [Section 2.5, “Installing MySQL on Linux”](#).
- Mac OS X. See [Section 2.4, “Installing MySQL on Mac OS X”](#).
- Solaris 2.8 on SPARC and x86, including support for native threads. See [Section 2.7, “Installing MySQL on Solaris and OpenSolaris”](#).
- Windows XP, Windows Vista, Windows Server 2003, and Windows Server 2008. See [Section 2.3, “Installing MySQL on Microsoft Windows”](#).

MySQL has also been known to run on other systems in the past. See [Section 2.1, “General Installation Guidance”](#). Some porting effort might be required for current versions of MySQL on these systems.

Not all platforms are equally well-suited for running MySQL. How well a certain platform is suited for a high-load mission-critical MySQL server is determined by the following factors:

- General stability of the thread library. A platform may have an excellent reputation otherwise, but MySQL is only as stable as the thread library it calls, even if everything else is perfect.
- The capability of the kernel and the thread library to take advantage of symmetric multi-processor (SMP) systems. In other words, when a process creates a thread, it should be possible for that thread to run on a CPU different from the original process.
- The capability of the kernel and the thread library to run many threads that acquire and release a mutex over a short critical region frequently without excessive context switches. If the implementation of `pthread_mutex_lock()` is too anxious to yield CPU time, this hurts MySQL tremendously. If this issue is not taken care of, adding extra CPUs actually makes MySQL slower.
- General file system stability and performance.
- Table size. If your tables are large, performance is affected by the ability of the file system to deal with large files and dealing with them efficiently.
- Our level of expertise here at Oracle Corporation with the platform. If we know a platform well, we enable platform-specific optimizations and fixes at compile time. We can also provide advice on configuring your system optimally for MySQL.

- The amount of testing we have done internally for similar configurations.
- The number of users that have run MySQL successfully on the platform in similar configurations. If this number is high, the likelihood of encountering platform-specific surprises is much smaller.

2.1.2. Choosing Which MySQL Distribution to Install

When preparing to install MySQL, you should decide which version to use. MySQL development occurs in several release series, and you can pick the one that best fits your needs. After deciding which version to install, you can choose a distribution format. Releases are available in binary or source format.

2.1.2.1. Choosing Which Version of MySQL to Install

The first decision to make is whether you want to use a production (stable) release or a development release. In the MySQL development process, multiple release series co-exist, each at a different stage of maturity.

Production Releases

- MySQL 5.5: Latest General Availability (Production) release
- MySQL 5.1: Previous stable (production-quality) release
- MySQL 5.0: Older stable release nearing the end of the product lifecycle

Development Release

- MySQL 5.6: Current release under development (pre-Production)

MySQL 4.1, 4.0, and 3.23 are old releases that are no longer supported.

See <http://www.mysql.com/about/legal/lifecycle/> for information about support policies and schedules.

Normally, if you are beginning to use MySQL for the first time or trying to port it to some system for which there is no binary distribution, use the most recent General Availability series listed in the preceding descriptions. All MySQL releases, even those from development series, are checked with the MySQL benchmarks and an extensive test suite before being issued.

If you are running an older system and want to upgrade, but do not want to take the chance of having a nonseamless upgrade, you should upgrade to the latest version in the same release series you are using (where only the last part of the version number is newer than yours). We have tried to fix only fatal bugs and make only small, relatively “safe” changes to that version.

If you want to use new features not present in the production release series, you can use a version from a development series. Be aware that development releases are not as stable as production releases.

We do not use a complete code freeze because this prevents us from making bugfixes and other fixes that must be done. We may add small things that should not affect anything that currently works in a production release. Naturally, relevant bugfixes from an earlier series propagate to later series.

If you want to use the very latest sources containing all current patches and bugfixes, you can use one of our source code repositories (see [Section 2.10.3, “Installing MySQL from a Development Source Tree”](#)). These are not “releases” as such, but are available as previews of the code on which future releases are to be based.

The naming scheme in MySQL 5.5 uses release names that consist of three numbers and a suffix; for example, **mysql-5.5.6-m3**. The numbers within the release name are interpreted as follows:

- The first number (**5**) is the major version and describes the file format. All MySQL 5 releases have the same file format.
- The second number (**5**) is the release level. Taken together, the major version and release level constitute the release series number.
- The third number (**6**) is the version number within the release series. This is incremented for each new release. Usually you want the latest version for the series you have chosen.

For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.

Release names also include a suffix to indicate the stability level of the release. Releases within a series progress through a set of suffixes to indicate how the stability level improves. The possible suffixes are:

- **mN** (for example, **m1**, **m2**, **m3**, ...) indicate a milestone number. MySQL development uses a milestone model, in which each milestone proceeds through a small number of versions with a tight focus on a small subset of thoroughly tested features. Following the releases for one milestone, development proceeds with another small number of releases that focuses on the next small set of features, also thoroughly tested. Features within milestone releases may be considered to be of pre-production quality.
- **rc** indicates a Release Candidate. Release candidates are believed to be stable, having passed all of MySQL's internal testing, and with all known fatal runtime bugs fixed. However, the release has not been in widespread use long enough to know for sure that all bugs have been identified. Only minor fixes are added.
- If there is no suffix, it indicates that the release is a General Availability (GA) or Production release. GA releases are stable, having successfully passed through all earlier release stages and are believed to be reliable, free of serious bugs, and suitable for use in production systems. Only critical bugfixes are applied to the release.

All releases of MySQL are run through our standard tests and benchmarks to ensure that they are relatively safe to use. Because the standard tests are extended over time to check for all previously found bugs, the test suite keeps getting better.

All releases have been tested at least with these tools:

- **An internal test suite.** The `mysql-test` directory contains an extensive set of test cases. We run these tests for every server binary. See [Section 23.1.2, "The MySQL Test Suite"](#), for more information about this test suite.
- **The MySQL benchmark suite.** This suite runs a range of common queries. It is also a test to determine whether the latest batch of optimizations actually made the code faster. See [Section 8.12.2, "The MySQL Benchmark Suite"](#).

We also perform additional integration and nonfunctional testing of the latest MySQL version in our internal production environment. Integration testing is done with different connectors, storage engines, replication modes, backup, partitioning, stored programs, and so forth in various combinations. Additional nonfunctional testing is done in areas of performance, concurrency, stress, high volume, upgrade and downgrade.

2.1.2.2. Choosing a Distribution Format

After choosing which version of MySQL to install, you should decide whether to use a binary distribution or a source distribution. In most cases, you should probably use a binary distribution, if one exists for your

platform. Binary distributions are available in native format for many platforms, such as RPM files for Linux or PKG package installers for Mac OS X or Solaris. Distributions also are available as Zip archives or compressed `tar` files.

Reasons to choose a binary distribution include the following:

- Binary distributions generally are easier to install than source distributions.
- To satisfy different user requirements, we provide several servers in binary distributions. `mysqld` is an optimized server that is a smaller, faster binary. `mysqld-debug` is compiled with debugging support.

Each of these servers is compiled from the same source distribution, though with different configuration options. All native MySQL clients can connect to servers from either MySQL version.

Under some circumstances, you may be better off installing MySQL from a source distribution:

- You want to install MySQL at some explicit location. The standard binary distributions are ready to run at any installation location, but you might require even more flexibility to place MySQL components where you want.
- You want to configure `mysqld` to ensure that features are available that might not be included in the standard binary distributions. Here is a list of the most common extra options that you may want to use to ensure feature availability:
 - `-DWITH_LIBWRAP=1` [155] for TCP wrappers support.
 - `-DWITH_ZLIB={system|bundled}` [156] for features that depend on compression
 - `-DWITH_DEBUG=1` [155] for debugging support
- You want to configure `mysqld` without some features that are included in the standard binary distributions. For example, distributions normally are compiled with support for all character sets. If you want a smaller MySQL server, you can recompile it with support for only the character sets you need.
- You want to use the latest sources from one of the Bazaar repositories to have access to all current bugfixes. For example, if you have found a bug and reported it to the MySQL development team, the bugfix is committed to the source repository and you can access it there. The bugfix does not appear in a release until a release actually is issued.
- You want to read (or modify) the C and C++ code that makes up MySQL. For this purpose, you should get a source distribution, because the source code is always the ultimate manual.
- Source distributions contain more tests and examples than binary distributions.

2.1.2.3. How and When Updates Are Released

MySQL is evolving quite rapidly and we want to share new developments with other MySQL users. We try to produce a new release whenever we have new and useful features that others also seem to have a need for.

We also try to help users who request features that are easy to implement. We take note of what our licensed users want, and we especially take note of what our support customers want and try to help them in this regard.

No one is *required* to download a new release. The [Release Notes](#) help you determine whether the new release has something you really want.

We use the following policy when updating MySQL:

- Enterprise Server releases are meant to appear every 18 months, supplemented by quarterly service packs and monthly rapid updates. Community Server releases are meant to appear 2 to 3 times per year.
- Releases are issued within each series. For each release, the last number in the version is one more than the previous release within the same series.
- Binary distributions for some platforms are made by us for major releases. Other people may make binary distributions for other systems, but probably less frequently.
- We make fixes available as soon as we have identified and corrected small or noncritical but annoying bugs. The fixes are available in source form immediately from our public Bazaar repositories, and are included in the next release.
- If by any chance a security vulnerability or critical bug is found in a release, our policy is to fix it in a new release as soon as possible. (We would like other companies to do this, too!)

2.1.3. How to Get MySQL

Check our downloads page at <http://dev.mysql.com/downloads/> for information about the current version of MySQL and for downloading instructions. For a complete up-to-date list of MySQL download mirror sites, see <http://dev.mysql.com/downloads/mirrors.html>. You can also find information there about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

To obtain the latest development source, see [Section 2.10.3, “Installing MySQL from a Development Source Tree”](#).

2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG

After you have downloaded the MySQL package that suits your needs and before you attempt to install it, you should make sure that it is intact and has not been tampered with. There are three means of integrity checking:

- MD5 checksums
- Cryptographic signatures using [GnuPG](#), the GNU Privacy Guard
- For RPM packages, the built-in RPM integrity verification mechanism

The following sections describe how to use these methods.

If you notice that the MD5 checksum or GPG signatures do not match, first try to download the respective package one more time, perhaps from another mirror site.

2.1.4.1. Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that you can verify with the following command, where *package_name* is the name of the package you downloaded:

```
shell> md5sum package_name
```

Example:

```
shell> md5sum mysql-standard-5.5.31-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945  mysql-standard-5.5.31-linux-i686.tar.gz
```

You should verify that the resulting checksum (the string of hexadecimal digits) matches the one displayed on the download page immediately below the respective package.



Note

Make sure to verify the checksum of the *archive file* (for example, the `.zip` or `.tar.gz` file) and not of the files that are contained inside of the archive.

Note that not all operating systems support the `md5sum` command. On some, it is simply called `md5`, and others do not ship it at all. On Linux, it is part of the **GNU Text Utilities** package, which is available for a wide range of platforms. You can download the source code from <http://www.gnu.org/software/textutils/> as well. If you have OpenSSL installed, you can use the command `openssl md5 package_name` instead. A Windows implementation of the `md5` command line utility is available from <http://www.fourmilab.ch/md5/winMd5Sum> is a graphical MD5 checking tool that can be obtained from <http://www.nullriver.com/index/products/winmd5sum>.

2.1.4.2. Signature Checking Using GnuPG

Another method of verifying the integrity and authenticity of a package is to use cryptographic signatures. This is more reliable than using MD5 checksums, but requires more work.

We sign MySQL downloadable packages with **GnuPG** (GNU Privacy Guard). **GnuPG** is an Open Source alternative to the well-known Pretty Good Privacy (**PGP**) by Phil Zimmermann. See <http://www.gnupg.org/> for more information about **GnuPG** and how to obtain and install it on your system. Most Linux distributions ship with **GnuPG** installed by default. For more information about **GnuPG**, see <http://www.openpgp.org/>.

To verify the signature for a specific package, you first need to obtain a copy of our public GPG build key, which you can download from <http://pgp.mit.com/>. The key that you want to obtain is named `mysql-build@oss.oracle.com`. Alternatively, you can cut and paste the key directly from the following text:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGP Universal 2.9.1 (Build 347)

mQGiBD4+owwRBAC14GIfUfCyEDSIEpVew3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQRyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBDck96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRwVArNYJddRT+rF2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQC7fJYLV
K2HtkrPbP72vwsEKMYhr0eKcBtLGFls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTtkamEnPcia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd81H0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvzh6//62zQJ7Q2TXlTUUwgUmVs
ZWFzZSBFbmdpbmVlcmluZyA8bXlzcWwtYnVpbGRab3NzLm9yYWNsZS5jb20+igYE
ExECACYCGyMGcWkIBwMCBBUCCAMEFgIDAQIEAQIXGAUCTnc+KgUJE/sCFQAKRCRM
cY07UHLh9SbMAJ41l+qBz2BZNSGCzwwA6YbhGPC7FwCgp8z5TzIw4YQuL5NGU/sy
0oSazqmJASIEAECAAwFAk53QS4FAwAsdQAACgkQlxC4m8pXrXwJ8Qf/be/UO9mq
foc2sMyhwMpn4/fdBWwfLkA12FXQDOQMvH9HsmEjnfUgYKXschZRI+DuHXelP7l
8G2aQLubhBsQf9eJkVrFTzuWmQkdIq+6Koulxv6ofkCcv3d1xtO2W7nb5yxcPVBp
rRfGFGeBjvZa58DymCngyGtAU6AOz4veavNmI2+GIDQsY66+tYDvZ+CxwzdYu+HD
V9HmrJfc6deM0mnBn7SRjzqxJpgoTQhithTav6q/R5/2p5NvQ/H84OgS6GjosfGc2
duUDzCP/kheMRkfzuyKCOHQPtJuIj8++gfpHtEU7IDUX1So3c9n0PdpeBvclsDbp
RnCNxQWU4mBot7kCDQQ+PgMdEAgA7+GJfxbMdY4wslPnjH9rF4N2qfWsEN/lxaZo
JYc3a6M02WcNhl6ahT2/tBK2w1QI4YFteR47gCvtgb601JHffOo2HfLmRDRiRjd1
DTCHqeyX7CHhcgj/dNRlW2Z015QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAWzE7zaD
5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqXlDY7LwoPEb/O9Rkbf4fmLe1l
EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D81lbf5vSYHbuE5p/loI
Dznkg/p8kW+3FxuWrycciqFTcNz215yyX39LXFnlLzKUB/F5GwADBQf+Lwqqa8CG
rRfsOAJxim63CHfty5mUc5rUSnTs1GYEIOCR1BeQauyPzBpDSD9MZ1ZaSafanFv
```

```
wFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGoTbOWI39E
cyzgs1zBdC++MPjCQTcA7p6JUvSP6oAB3FQWg54tuUo0Ec8bsM8b3Ev42LmuQT5N
dKHGwHsXTpt10klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkKht926s/y
mfdF5HkdQ1cyvsz5tryVI3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUOetdZWhe7
0YGNPwlyjWJT1IhUBBgRAGAMBQJodz3tBQkT+wG4ABIHZUdQRwABAQkQjHGNO1By
4fUUmwCbBYr2+bBen/L2BOcnw9Z/QFWuhRMAoKVgCFm5fadQ3Afi+UQ1AcOphrnJ
=Eto8
-----END PGP PUBLIC KEY BLOCK-----
```

To import the build key into your personal public GPG keyring, use `gpg --import`. For example, if you have saved the key in a file named `mysql_pubkey.asc`, the import command looks like this:

```
shell> gpg --import mysql_pubkey.asc
gpg: key 5072E1F5: public key "MySQL Release Engineering
<mysql-build@oss.oracle.com>" imported
gpg: Total number processed: 1
gpg:             imported: 1
gpg: no ultimately trusted keys found
```

You can also download the key from the public keyserver using the public key id, `5072E1F5`:

```
shell> gpg --recv-keys 5072E1F5
gpg: requesting key 5072E1F5 from hkp server keys.gnupg.net
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
1 new user ID
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
53 new signatures
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:             new user IDs: 1
gpg:             new signatures: 53
```

If you want to import the key into your RPM configuration to validate RPM install packages, you should be able to import the key directly:

```
shell> rpm --import mysql_pubkey.asc
```

If you experience problems, try exporting the key from `gpg` and importing:

```
shell> gpg --export -a 5072e1f5 > 5072e1f5.asc
shell> rpm --import 5072e1f5.asc
```

Alternatively, `rpm` also supports loading the key directly from a URL, and you can use this manual page:

```
shell> rpm --import http://dev.mysql.com/doc/refman/5.5/en/checking-gpg-signature.html
```

After you have downloaded and imported the public build key, download your desired MySQL package and the corresponding signature, which also is available from the download page. The signature file has the same name as the distribution file with an `.asc` extension, as shown by the examples in the following table.

Table 2.1. MySQL Package and Signature Files

File Type	File Name
Distribution file	<code>mysql-standard-5.5.31-linux-i686.tar.gz</code>
Signature file	<code>mysql-standard-5.5.31-linux-i686.tar.gz.asc</code>

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file:

```
shell> gpg --verify package_name.asc
```

Example:

```
shell> gpg --verify mysql-standard-5.5.31-linux-i686.tar.gz.asc
gpg: Signature made Tue 01 Feb 2011 02:38:30 AM CST using DSA key ID 5072E1F5
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
```

The `Good signature` message indicates that everything is all right. You can ignore any `insecure memory` warning you might obtain.

See the GPG documentation for more information on how to work with public keys.

2.1.4.3. Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages have a built-in GPG signature and MD5 checksum. You can verify a package by running the following command:

```
shell> rpm --checksig package_name.rpm
```

Example:

```
shell> rpm --checksig MySQL-server-5.5.31-0.glibc23.i386.rpm
MySQL-server-5.5.31-0.glibc23.i386.rpm: md5 gpg OK
```



Note

If you are using RPM 4.1 and it complains about `(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5)`, even though you have imported the MySQL public build key into your own GPG keyring, you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (or GPG itself). Rather, RPM maintains a separate keyring because it is a system-wide application and a user's GPG public keyring is a user-specific file. To import the MySQL public key into the RPM keyring, first obtain the key as described in [Section 2.1.4.2, "Signature Checking Using GnuPG"](#). Then use `rpm --import` to import the key. For example, if you have saved the public key in a file named `mysql_pubkey.asc`, import it using this command:

```
shell> rpm --import mysql_pubkey.asc
```

If you need to obtain the MySQL public key, see [Section 2.1.4.2, "Signature Checking Using GnuPG"](#).

2.1.5. Installation Layouts

The installation layout differs for different installation types (for example, native packages, binary tarballs, and source tarballs), which can lead to confusion when managing different systems or using different installation sources. The individual layouts are given in the corresponding installation type or platform chapter, as described following. Note that the layout of installations from vendors other than Oracle may differ from these layouts.

- [Section 2.3.1, "MySQL Installation Layout on Microsoft Windows"](#)
- [Section 2.10.1, "MySQL Layout for Source Installation"](#)
- [Table 2.2, "MySQL Installation Layout for Generic Unix/Linux Binary Package"](#)
- [Table 2.11, "MySQL Installation Layout for Linux RPM"](#)

- [Table 2.8, “MySQL Installation Layout on Mac OS X”](#)

2.1.6. Compiler-Specific Build Characteristics

In some cases, the compiler used to build MySQL affects the features available for use. The notes in this section apply for binary distributions provided by Oracle Corporation or that you compile yourself from source.

icc (Intel C++ Compiler) Builds

A server built with `icc` has these characteristics:

- SSL support is not included.

2.2. Installing MySQL from Generic Binaries on Unix/Linux

Oracle provides a set of binary distributions of MySQL. These include binary distributions in the form of compressed `tar` files (files with a `.tar.gz` extension) for a number of platforms, as well as binaries in platform-specific package formats for selected platforms.

This section covers the installation of MySQL from a compressed `tar` file binary distribution. For other platform-specific package formats, see the other platform-specific sections. For example, for Windows distributions, see [Section 2.3, “Installing MySQL on Microsoft Windows”](#).

To obtain MySQL, see [Section 2.1.3, “How to Get MySQL”](#).

MySQL compressed `tar` file binary distributions have names of the form `mysql-VERSION-OS.tar.gz`, where `VERSION` is a number (for example, `5.5.31`), and `OS` indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

To install MySQL from a compressed `tar` file binary distribution, your system must have GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it. If your `tar` program supports the `z` option, it can both uncompress and unpack the file.

GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU `tar`. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from <http://www.gnu.org/software/tar/>.



Warning

If you have previously installed MySQL using your operating system native package management system, such as `yum` or `apt-get`, you may experience problems installing using a native binary. Make sure your previous MySQL installation has been removed entirely (using your package management system), and that any additional files, such as old versions of your data files, have also been removed. You should also check the existence of configuration files such as `/etc/my.cnf` or the `/etc/mysql` directory have been deleted.

If you run into problems and need to file a bug report, please use the instructions in [Section 1.7, “How to Report Bugs or Problems”](#).

On Unix, to install a compressed `tar` file binary distribution, unpack it at the installation location you choose (typically `/usr/local/mysql`). This creates the directories shown in the following table.

Table 2.2. MySQL Installation Layout for Generic Unix/Linux Binary Package

Directory	Contents of Directory
<code>bin</code>	Client programs and the <code>mysqld</code> server
<code>data</code>	Log files, databases
<code>docs</code>	Manual in Info format
<code>man</code>	Unix manual pages
<code>include</code>	Include (header) files
<code>lib</code>	Libraries
<code>scripts</code>	<code>mysql_install_db</code>
<code>share</code>	Miscellaneous support files, including error messages, sample configuration files, SQL for database installation
<code>sql-bench</code>	Benchmarks

Debug versions of the `mysqld` binary are available as `mysqld-debug`. To compile your own debug version of MySQL from a source distribution, use the appropriate configuration options to enable debugging support. For more information on compiling from source, see [Section 2.10, “Installing MySQL from Source”](#).

To install and use a MySQL binary distribution, the basic command sequence looks like this:

```
shell> groupadd mysql
shell> useradd -r -g mysql mysql
shell> cd /usr/local
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
# Next command is optional
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

A more detailed version of the preceding description for installing a binary distribution follows.



Note

This procedure assumes that you have `root` (administrator) access to your system. Alternatively, you can prefix each command using the `sudo` (Linux) or `pfexec` (OpenSolaris) command.

The procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to [Section 2.11, “Postinstallation Setup and Testing”](#).

Create a `mysql` User and Group

If your system does not already have a user and group for `mysqld` to run as, you may need to create one. The following commands add the `mysql` group and the `mysql` user. You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following instructions. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

```
shell> groupadd mysql
shell> useradd -r -g mysql mysql
```

**Note**

Because the user is required only for ownership purposes, not login purposes, the `useradd` command uses the `-r` option to create a user that does not have login permissions to your server host. Omit this option to permit logins for the user (or if your `useradd` does not support the option).

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it. The example here unpacks the distribution under `/usr/local`. The instructions, therefore, assume that you have permission to create files and directories in `/usr/local`. If that directory is protected, you must perform the installation as `root`.

```
shell> cd /usr/local
```

Obtain a distribution file using the instructions in [Section 2.1.3, “How to Get MySQL”](#). For a given release, binary distributions for all platforms are built from the same MySQL source distribution.

Unpack the distribution, which creates the installation directory. Then create a symbolic link to that directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

The `tar` command creates a directory named `mysql-VERSION-OS`. The `ln` command makes a symbolic link to that directory. This enables you to refer more easily to the installation directory as `/usr/local/mysql`.

If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it. Replace the preceding `tar` command with the following alternative command to uncompress and extract the distribution:

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
```

Perform Postinstallation Setup

The remainder of the installation process involves setting up the configuration file, creating the core databases, and starting the MySQL server. For instructions, see [Section 2.11, “Postinstallation Setup and Testing”](#).

**Note**

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.11, “Postinstallation Setup and Testing”](#).

2.3. Installing MySQL on Microsoft Windows

A native Windows distribution of MySQL has been available since version 3.21 and represents a sizable percentage of the daily downloads of MySQL. This section describes the process for installing MySQL on Windows.

**Note**

If you are upgrading MySQL from an existing installation older than MySQL 4.1.5, you must first perform the procedure described in [Section 2.3.9, “Upgrading MySQL on Windows”](#).

To run MySQL on Windows, you need the following:

- A Windows operating system such as Windows 2000, Windows XP, Windows Vista, Windows Server 2003, or Windows Server 2008. Both 32-bit and 64-bit versions are supported.

A Windows operating system permits you to run the MySQL server as a service. See [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

Generally, you should install MySQL on Windows using an account that has administrator rights. Otherwise, you may encounter problems with certain operations such as editing the `PATH` environment variable or accessing the `Service Control Manager`. Once installed, MySQL does not need to be executed using a user with Administrator privileges.

- TCP/IP protocol support.
- Enough space on the hard drive to unpack, install, and create the databases in accordance with your requirements (generally a minimum of 200 megabytes is recommended.)

For a list of limitations within the Windows version of MySQL, see [Section E.10.5, “Windows Platform Limitations”](#).

There may also be other requirements, depending on how you plan to use MySQL:

- If you plan to connect to the MySQL server using ODBC, you need a Connector/ODBC driver. See [Section 22.1, “MySQL Connector/ODBC”](#).
- If you plan to use MySQL server with ADO.NET applications, you need the Connector/Net driver. See [Section 22.2, “MySQL Connector/Net”](#).
- If you need tables with a size larger than 4GB, install MySQL on an NTFS or newer file system. Do not forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See [Section 13.1.17, “CREATE TABLE Syntax”](#).

MySQL for Windows is available in several distribution formats:

- Binary distributions are available that contain a setup program that installs everything you need so that you can start the server immediately. Another binary distribution format contains an archive that you simply unpack in the installation location and then configure yourself. For details, see [Section 2.3.2, “Choosing An Installation Package”](#).
- The source distribution contains all the code and support files for building the executables using the Visual Studio compiler system.

Generally speaking, you should use a binary distribution that includes an installer. It is simpler to use than the others, and you need no additional tools to get MySQL up and running. The installer for the Windows version of MySQL, combined with a GUI Configuration Wizard, automatically installs MySQL, creates an option file, starts the server, and secures the default user accounts.

**Caution**

Using virus scanning software such as Norton/Symantec Anti-Virus on directories containing MySQL data and temporary tables can cause issues, both in terms of the

performance of MySQL and the virus-scanning software mis-identifying the contents of the files as containing spam. This is because of the fingerprinting mechanism used by the virus scanning software, and the way in which MySQL rapidly updates different files, which may be identified as a potential security risk.

After installing MySQL Server, it is recommended that you disable virus scanning on the main directory (`datadir` [492]) being used to store your MySQL table data. There is usually a system built into the virus scanning software to enable certain directories to be specifically ignored during virus scanning.

In addition, by default, MySQL creates temporary files in the standard Windows temporary directory. To prevent the temporary files also being scanned, you should configure a separate temporary directory for MySQL temporary files and add this to the virus scanning exclusion list. To do this, add a configuration option for the `tmpdir` [466] parameter to your `my.ini` configuration file. For more information, see Section 2.3.7.2, “Creating an Option File”.

The following section describes how to install MySQL on Windows using a binary distribution. To use an installation package that does not include an installer, follow the procedure described in Section 2.3.7, “Installing MySQL on Microsoft Windows Using a `noinstall` Zip Archive”. To install using a source distribution, see Section 2.10, “Installing MySQL from Source”.

MySQL distributions for Windows can be downloaded from <http://dev.mysql.com/downloads/>. See Section 2.1.3, “How to Get MySQL”.

2.3.1. MySQL Installation Layout on Microsoft Windows

For MySQL 5.5 on Windows, the default installation directory is `C:\Program Files\MySQL\MySQL Server 5.5`. Some Windows users prefer to install in `C:\mysql`, the directory that formerly was used as the default. However, the layout of the subdirectories remains the same.

All of the files are located within this parent directory, using the structure shown in the following table.

Table 2.3. MySQL Installation Layout for Windows

Directory	Contents of Directory
<code>bin</code>	Client programs and the <code>mysqld</code> server
<code>C:\Documents and Settings\All Users\Application Data\MySQL</code>	Log files, databases (Windows XP, Windows Server 2003)
<code>C:\ProgramData\MySQL</code>	Log files, databases (Windows 7, Windows Server 2008)
<code>examples</code>	Example programs and scripts
<code>include</code>	Include (header) files
<code>lib</code>	Libraries
<code>scripts</code>	Utility scripts
<code>share</code>	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation

If you install MySQL using a Windows MSI package, this package creates and sets up the data directory that the installed server will use, but as of MySQL 5.5.5, it also creates a pristine “template” data directory named `data` under the installation directory. This directory can be useful when the machine will be used

to run multiple instances of MySQL: After an installation has been performed using an MSI package, the template data directory can be copied to set up additional MySQL instances. See [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#).

2.3.2. Choosing An Installation Package

For MySQL 5.5, there are installation package formats to choose from when installing MySQL on Windows:



Note

MySQL Installer and the "Complete Package" methods for installing MySQL are similar, but different. The MySQL Installer is the newer and more advanced option, and it includes all functionality found within the "Complete Package."

- **MySQL Installer:** This package has a file name similar to `mysql-installer-community-5.5.31.0.msi` and utilizes MSI's to automatically install MySQL server and other products. It will download and apply updates to itself, and for each of the installed products. It also configures the additional non-server products, such as MySQL Workbench.

The installed products are configurable, and this includes: documentation with samples and examples, connectors (such as C, C++, J, NET, and ODBC), MySQL Workbench, and the MySQL Server with its components.

MySQL Installer will run on all Windows platforms that are [supported by MySQL](#). Note: because MySQL is not a native component of Windows, and depends on .NET, it will not work on minimal installation options like the "Server Core" version of Windows Server 2008.

- **The Complete Package:** This package has a file name similar to `mysql-5.5.31-win32.msi` and contains all files needed for a complete Windows installation, including the Configuration Wizard. This package includes optional components such as the embedded server and benchmark suite.
- **The Noinstall Archive:** This package has a file name similar to `mysql-5.5.31-win32.zip` and contains all the files found in the Complete install package, with the exception of the Configuration Wizard. This package does not include an automated installer, and must be manually installed and configured.

MySQL Installer is recommended for most users. Both MySQL Installer and the alternative "Complete distribution" versions are available as `.msi` files for use with installations on Windows. The Noinstall distribution is packaged as a Zip archive. To use a Zip archive, you must have a tool that can unpack `.zip` files.

Your choice of install package affects the installation process you must follow. If you choose to install using MySQL Installer, see [Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”](#). If you choose to install a standard MSI package, see [Section 2.3.4, “Installing MySQL on Microsoft Windows Using an MSI Package”](#). If you choose to install a Noinstall archive, see [Section 2.3.7, “Installing MySQL on Microsoft Windows Using a noinstall Zip Archive”](#).

2.3.3. Installing MySQL on Microsoft Windows Using MySQL Installer

MySQL Installer is an application that simplifies the installation and updating process for a wide range of MySQL products, including [MySQL Notifier for Microsoft Windows](#), [MySQL Workbench](#), and [MySQL for Excel](#). From this central application, you can see which MySQL products are already installed, configure them, and update or remove them if necessary. The installer can also install plugins, documentation, tutorials, and example databases. The MySQL Installer is only available for Microsoft Windows, and includes both a GUI and command-line interface.

Download MySQL Installer at <http://dev.mysql.com/downloads/installer/>

For release notes detailing the changes in each release of MySQL Installer, see [MySQL Installer Release Notes](#).

MySQL Installer is compatible with pre-existing installations, and adds them to its list of installed components. And while the MySQL Installer is bundled with a specific version of MySQL Server, a single MySQL Installer instance can install and manage multiple MySQL Server versions. For example, a single MySQL Installer instance can install versions 5.1, 5.5, and 5.6. It can also manage both commercial and community versions of the MySQL Server.

MySQL Installer handles the initial configuration and setup of the applications. For example:

1. It will create MySQL Server connections in MySQL Workbench.
2. It creates the configuration file (`my.ini`) that is used to configure the MySQL Server.
3. It imports example databases.
4. It creates MySQL Server user accounts with configurable permissions based on general roles, such as DB Administrator, DB Designer, and Backup Admin. It optionally creates a Windows user named `MysqlSys` with limited privileges, which would then run the MySQL Server.

This feature is only available during the initial installation of the MySQL Server, and not during future updates. User accounts may also be added with MySQL Workbench.

5. If the "Advanced Configuration" option is checked, then the **Logging Options** are also configured. This includes defining file paths for the error log, general log, slow query log (including the configuration of seconds it requires to execute a query), and the binary log.

MySQL Installer can optionally check for updated components and download them for you automatically.

2.3.3.1. MySQL Installer GUI

After installation of the GUI version, the installer will have add its own Start Menu item under MySQL.

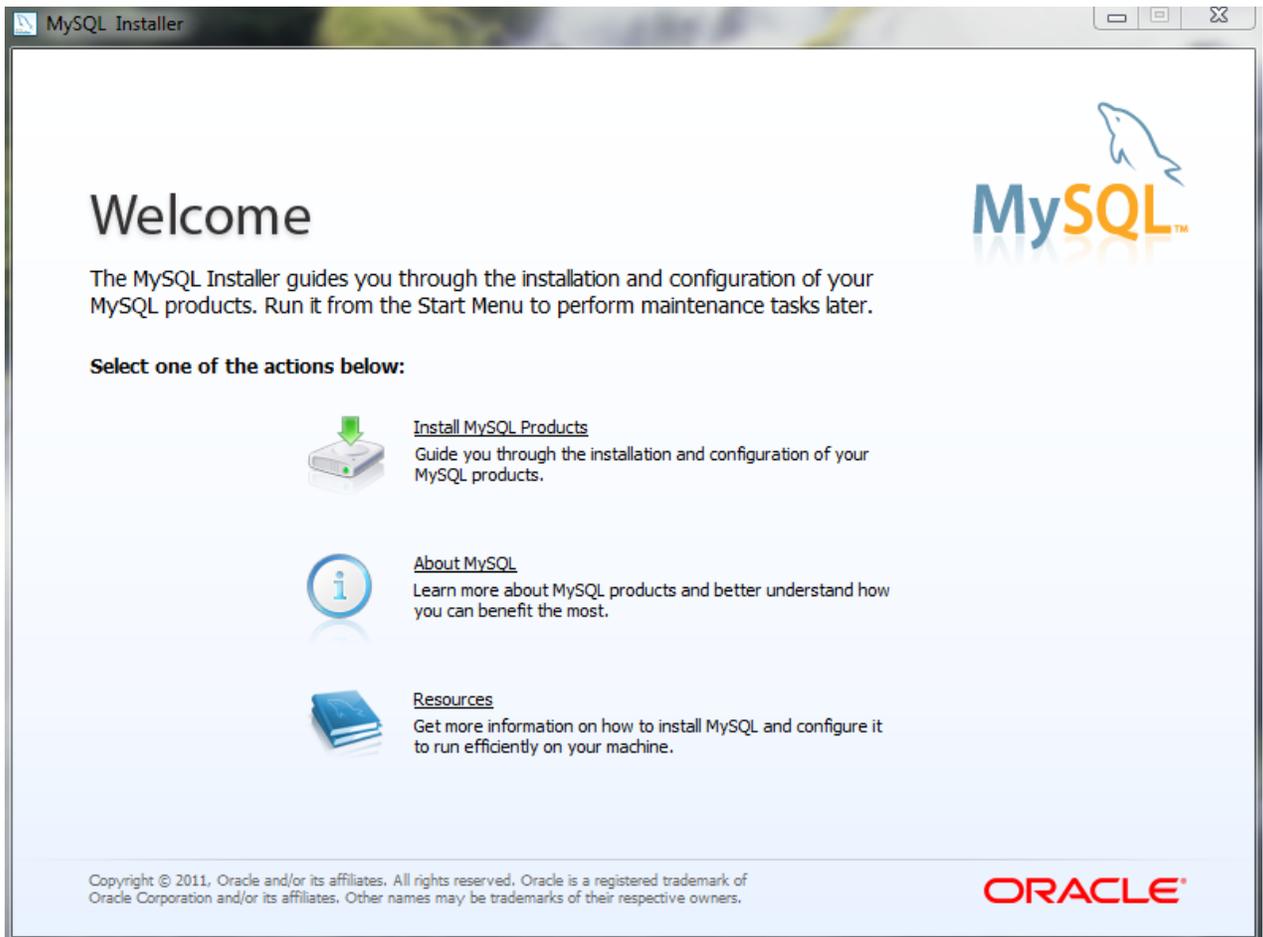


Note

Files that are generated by MySQL Installer grant full permissions to the user that executes MySQL Installer, including `my.ini`. This does not apply to files and directories for specific products such as the MySQL Server data directory in `ProgramData`, that is owned by `SYSTEM`.

After the installer itself has been installed and started, the following screen is displayed:

Figure 2.1. MySQL Installer - Welcome Screen

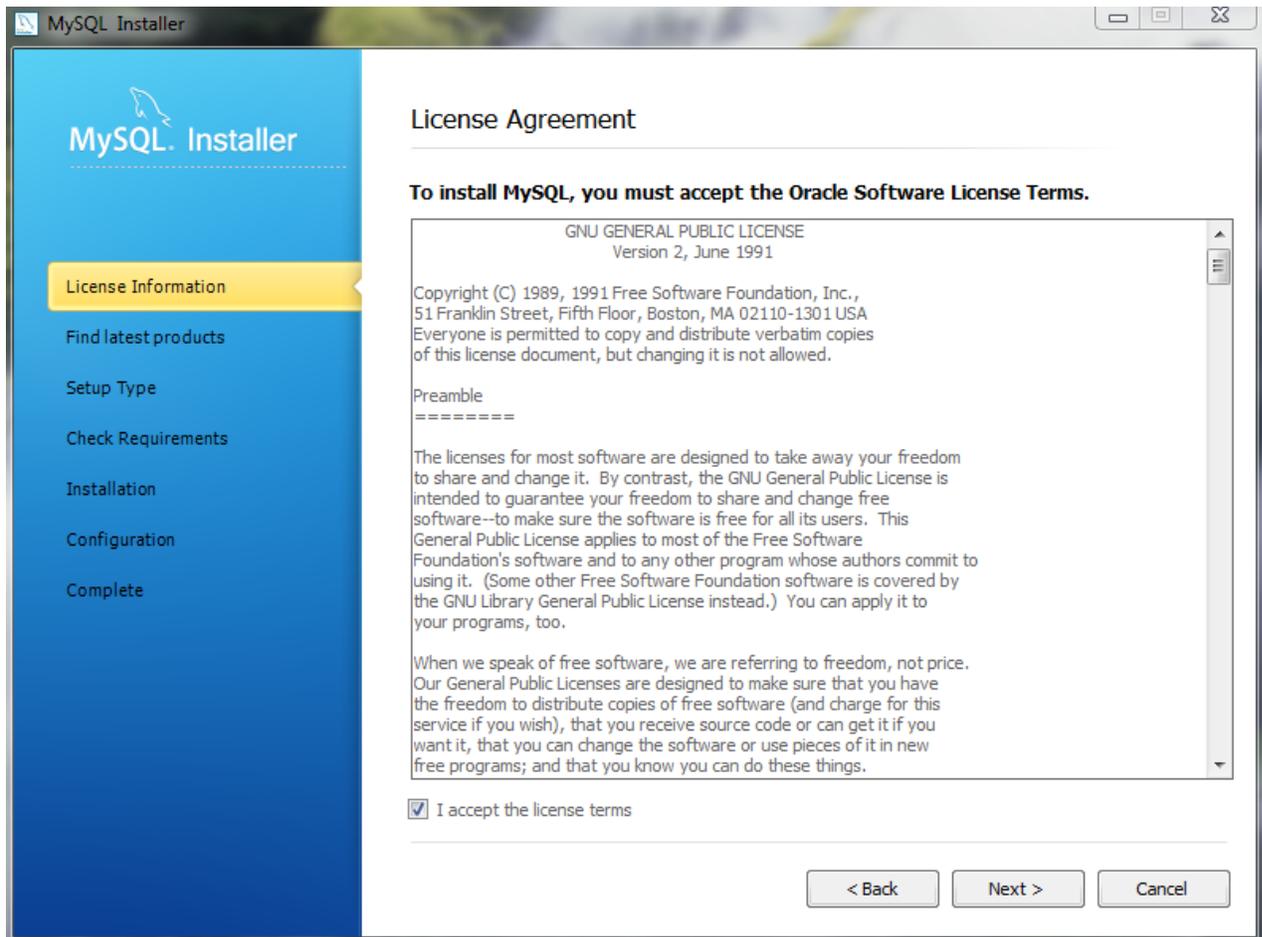


There are three main options:

1. Install MySQL Products - The Installation Wizard.
2. About MySQL - Learn about MySQL products and features.
3. Resources - Information to help install and configure MySQL.

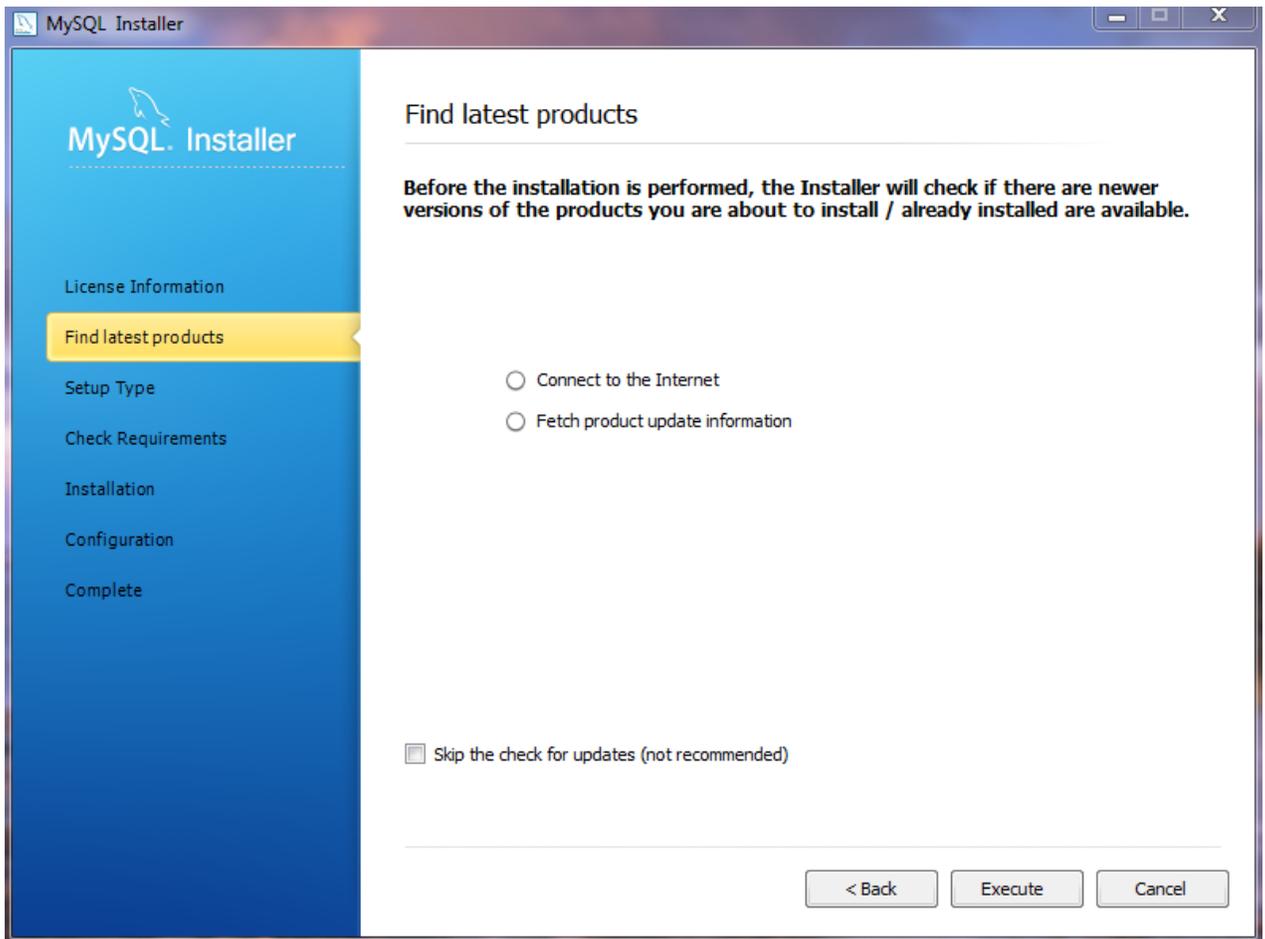
To **Install MySQL Products** after executing MySQL Installer for the first time, you must accept the license agreement before proceeding with the installation process.

Figure 2.2. MySQL Installer - License Agreement



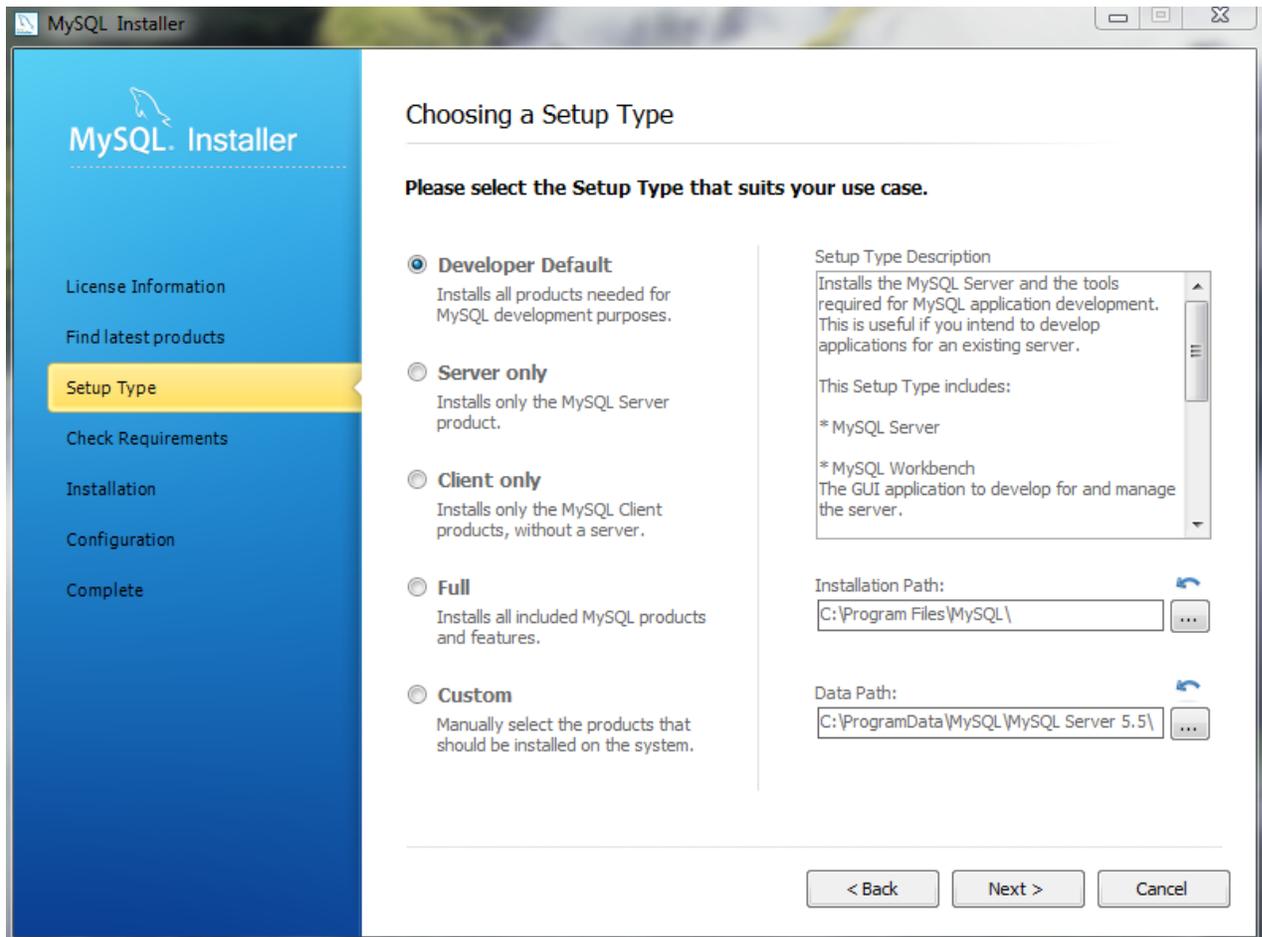
If you are connected to the Internet, then the Installer will search for the latest MySQL components and add them to the installation bundle. Click [Connect to the Internet](#) to complete this step, or otherwise check the [Skip](#) checkbox and then [Continue](#).

Figure 2.3. MySQL Installer - Find latest products



If you chose "Connect to the Internet," the next page will show the progress of MySQL Installer's search for available updates. When the search is complete (or if you opted to skip the search), you will be taken to the **Choose Setup Type** page:

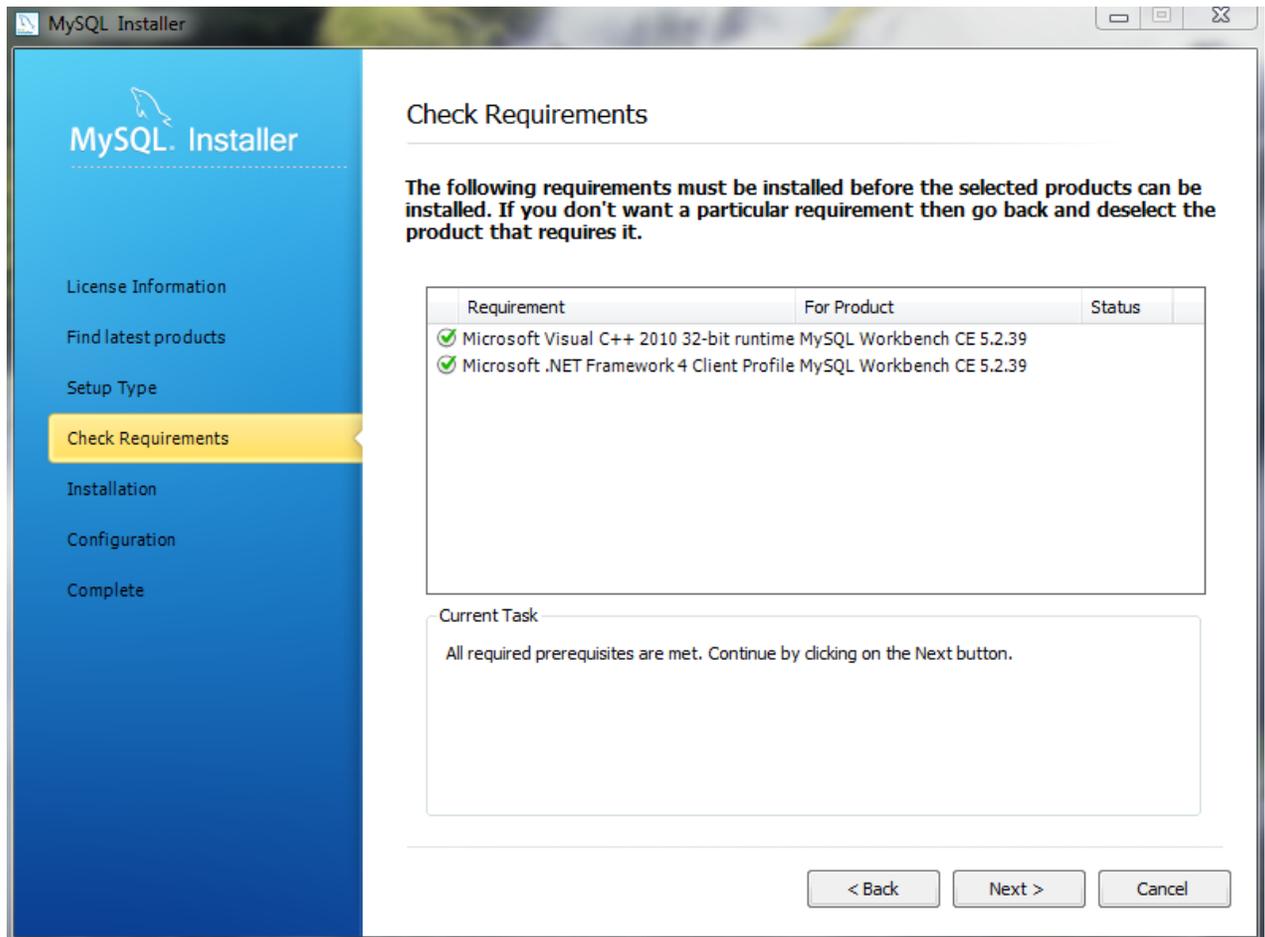
Figure 2.4. MySQL Installer - Choose Setup Type



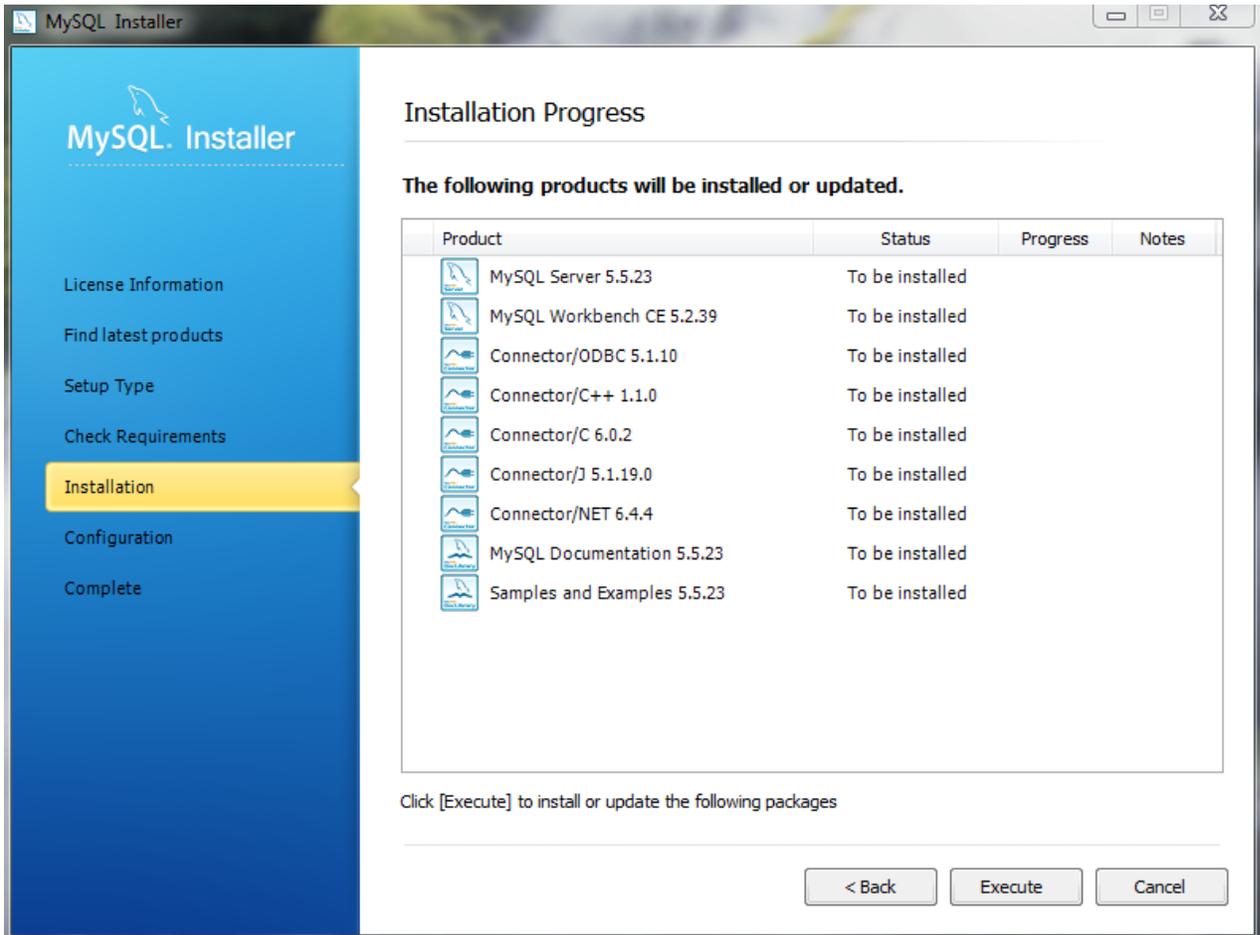
Each type contains a description, so read the [Setup Type Description](#) to determine the option most compatible with your preferences.

The Installation and Data paths are also defined here, and a caution flag will notify you if the data path you define already exists.

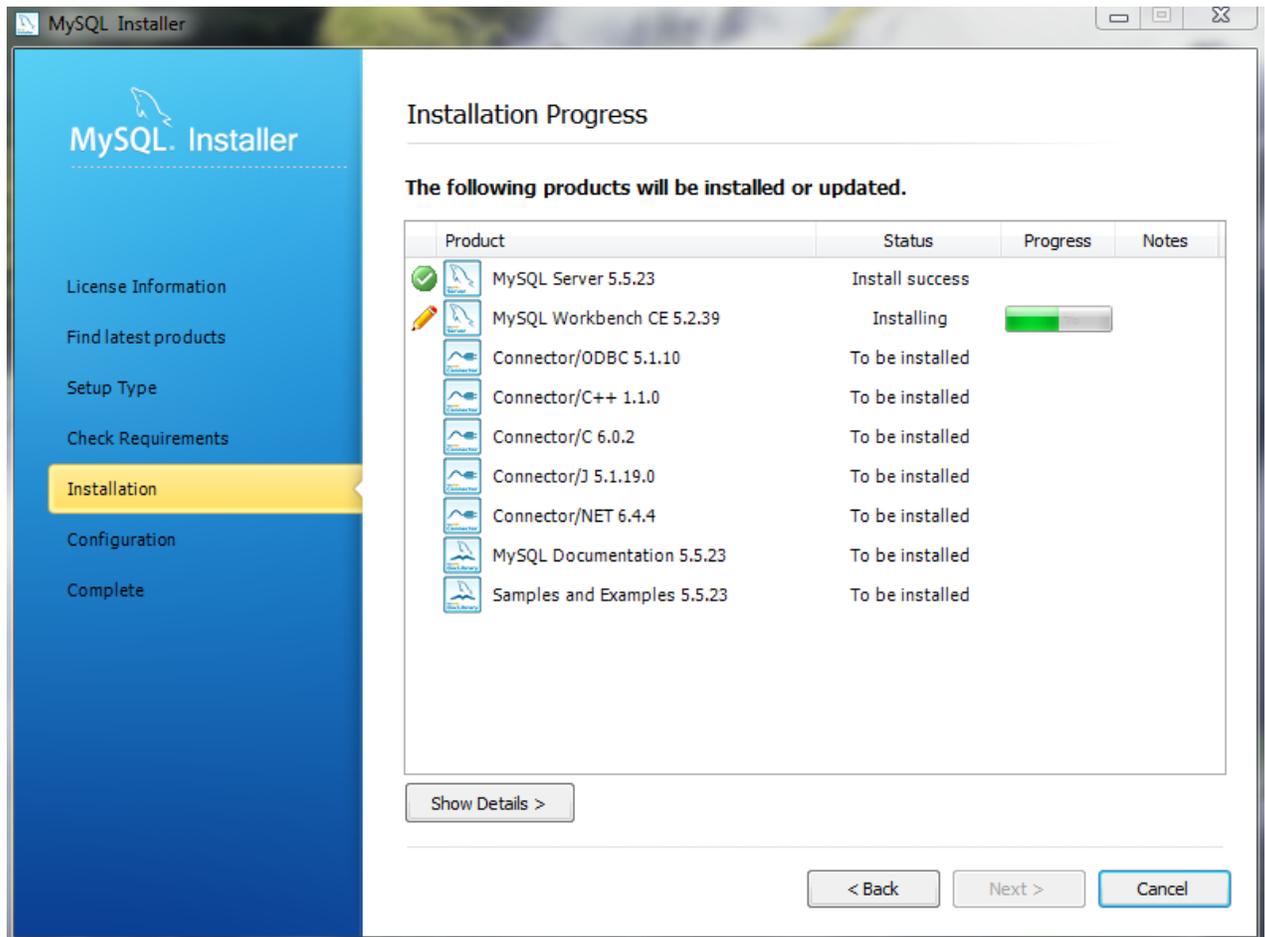
After you select a setup type, the MySQL Installer will check your system for the necessary external requirements and download then install missing components onto your system.

Figure 2.5. MySQL Installer - Check Requirements

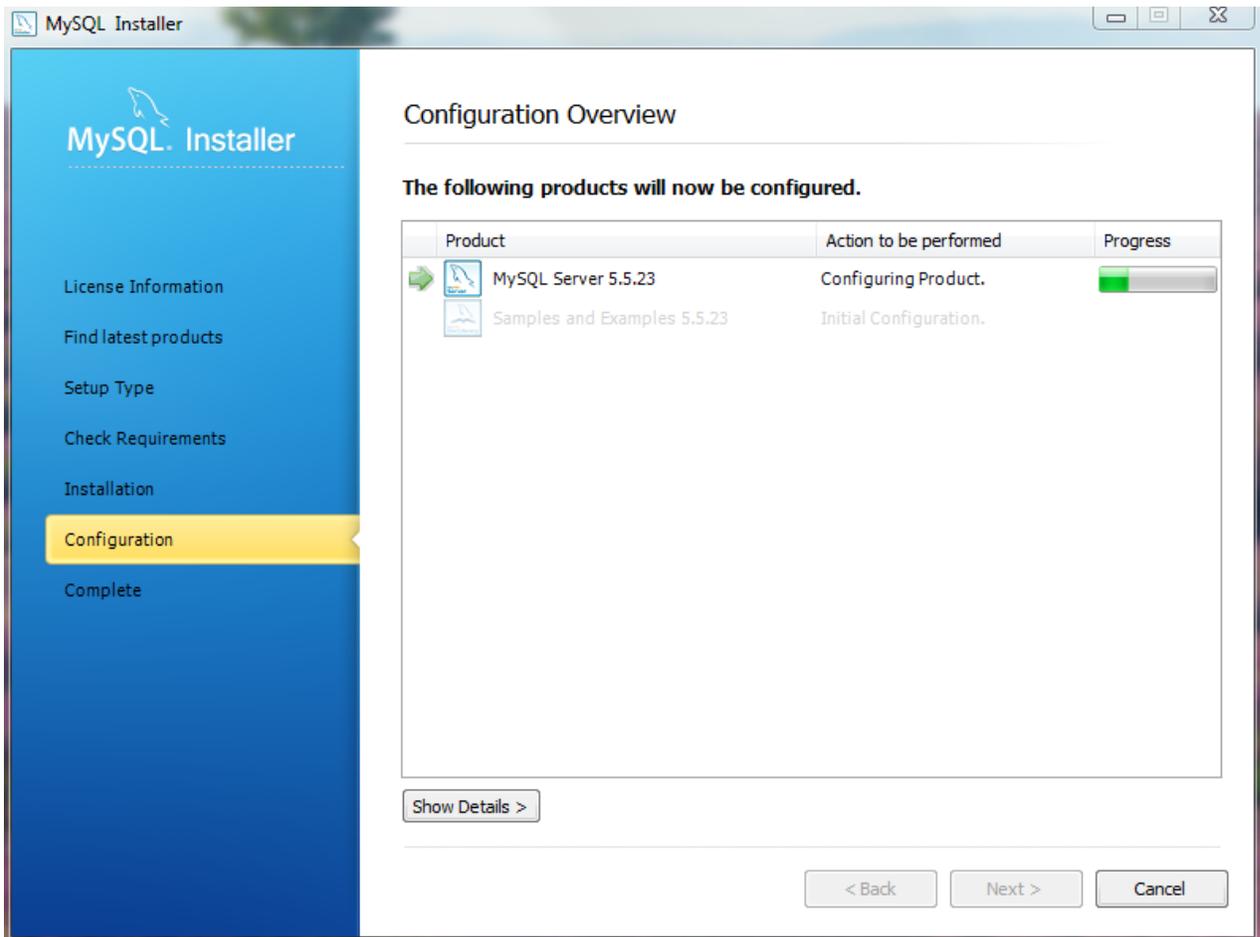
The next window lists the MySQL components that are scheduled to be installed:

Figure 2.6. MySQL Installer - Installation Progress

As components are installed, you'll see their status change from "to be installed" to "install success."

Figure 2.7. MySQL Installer - Installation Progress status

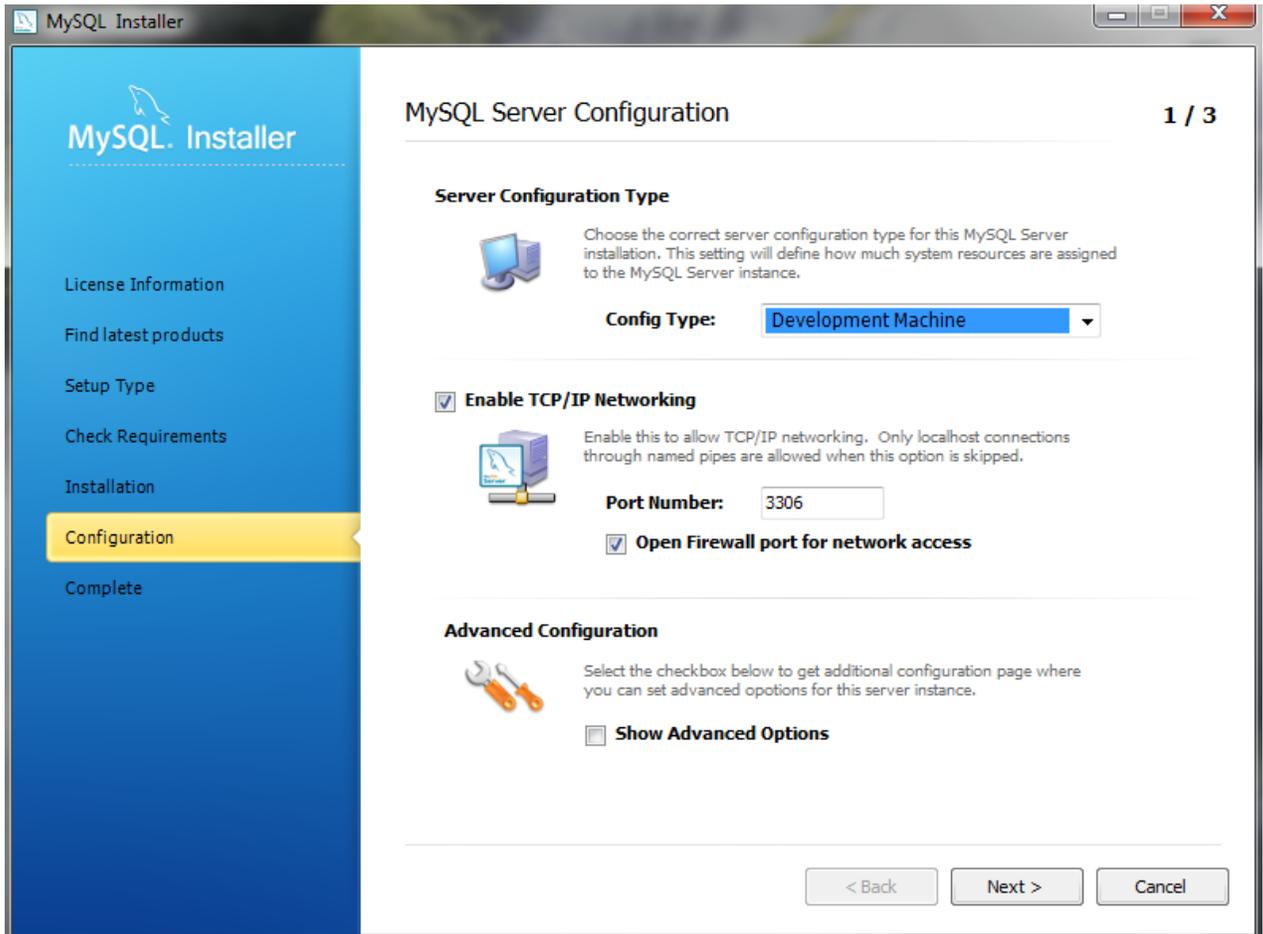
After all components are installed, the next step involves configuring the products. The [Configuration Overview](#) window displays the progress and then loads a configuration window if it is required.

Figure 2.8. MySQL Installer - Configuration Overview

The ideal MySQL Server configuration depends on your intended use, as explained in the next window. Choose the description that most closely applies to your machine.

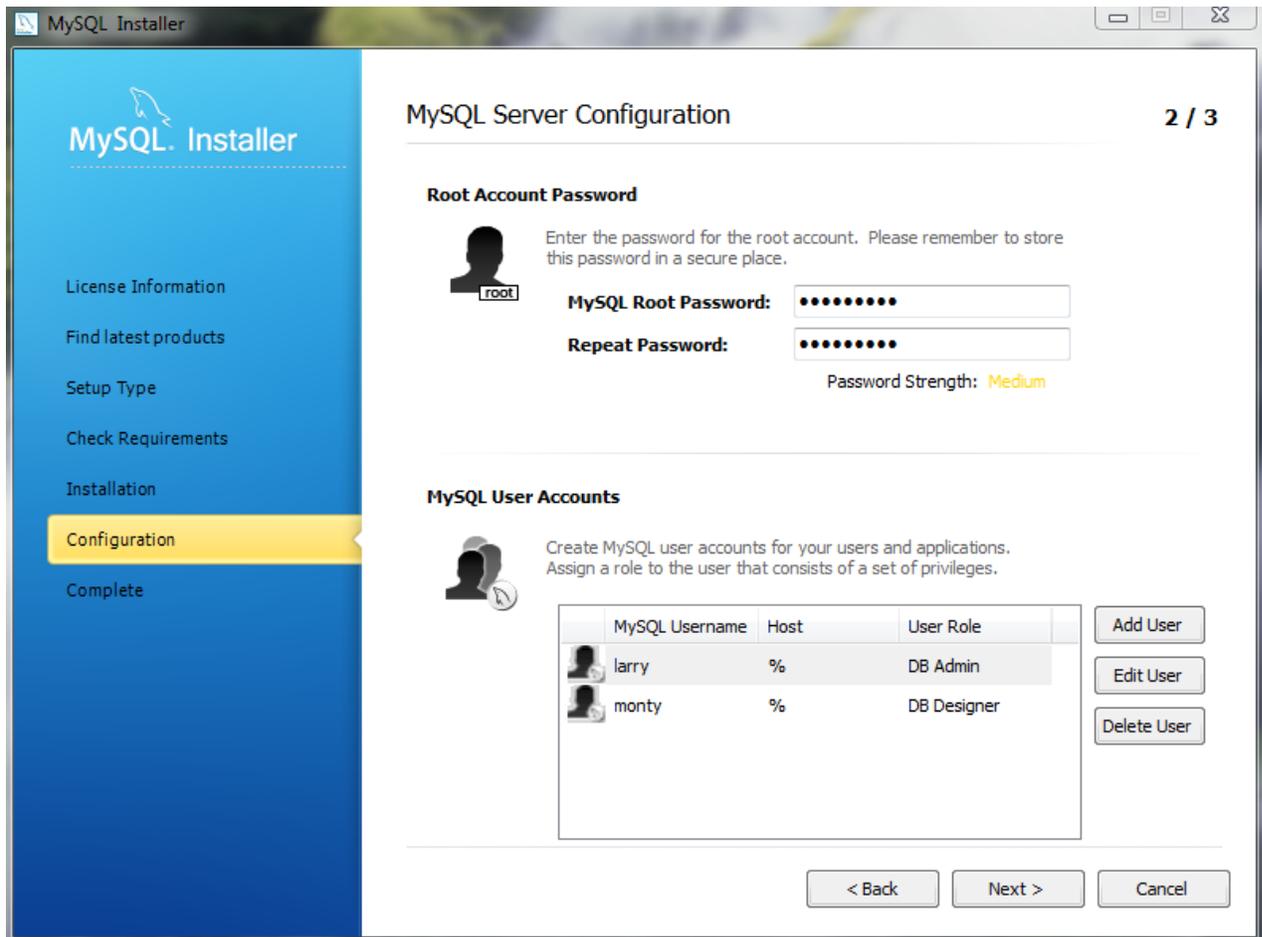
You may enable TCP/IP Networking here as otherwise only localhost connections are allowed.

Checking the "Advanced Configuration" option provides additional **Logging Options** to configure. This includes defining file paths for the error log, general log, slow query log (including the configuration of seconds it requires to execute a query), and the binary log.

Figure 2.9. MySQL Installer - MySQL Server Configuration: Define platform, networking, and logging options

Next, choose your account information. Defining a root password is required, whereas it's optional to create additional users. There are several different predefined user roles that each have different permission levels. For example, a "DB Admin" will have more privileges than a "DB Designer."

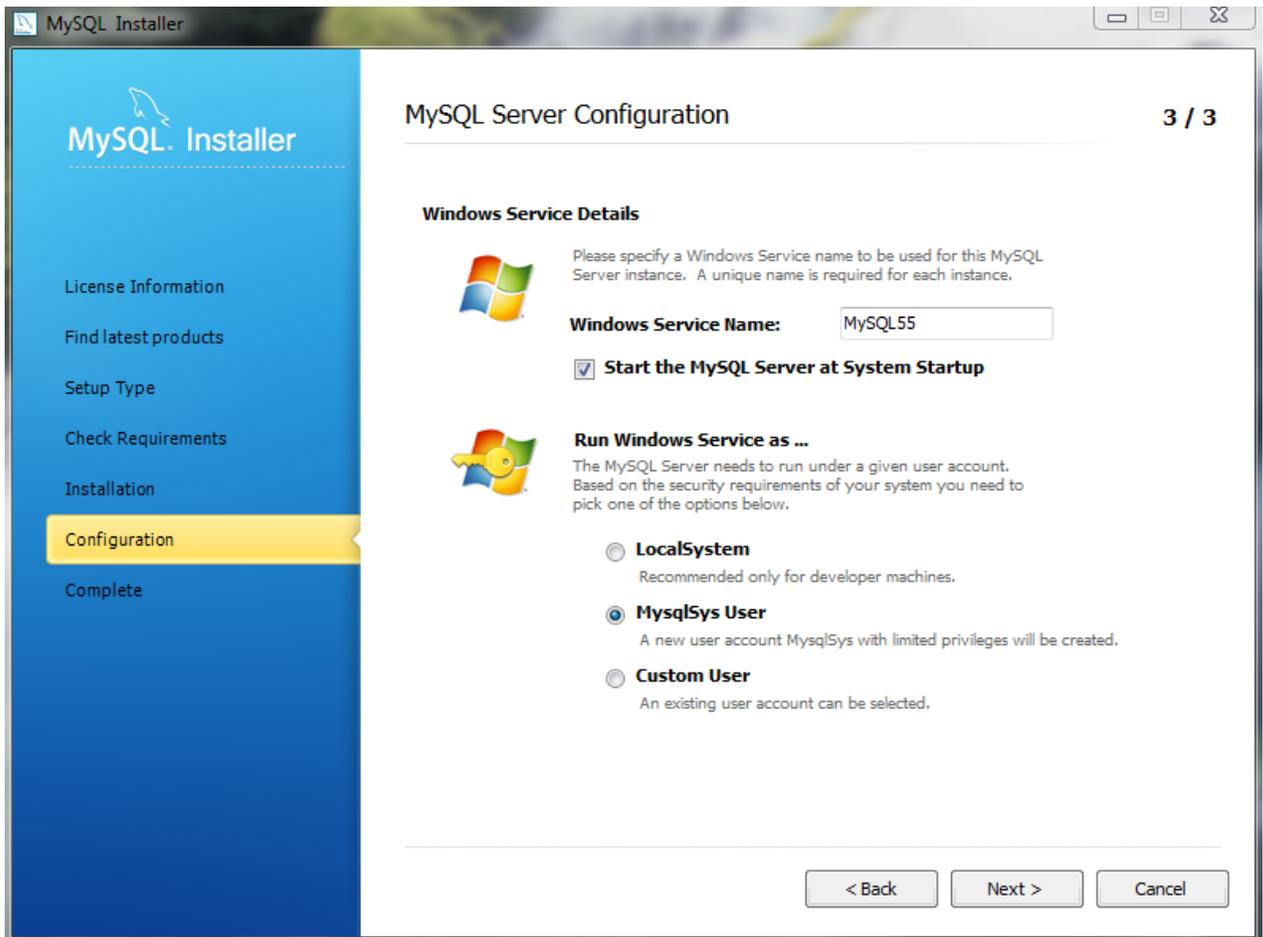
Figure 2.10. MySQL Installer - MySQL Server Configuration: User accounts

**Note**

If the MySQL Server is already installed, then the [Current Root Password](#) will also be needed.

Next, configure the **Windows Service Details**. This includes the name, whether MySQL Server should be loaded at startup, and how the Windows Service for MySQL Server will be run. Choosing **MysqSys User** will create a MySQL Windows user with limited privileges that will run the MySQL Server.

Figure 2.11. MySQL Installer - MySQL Server Configuration: Windows service details

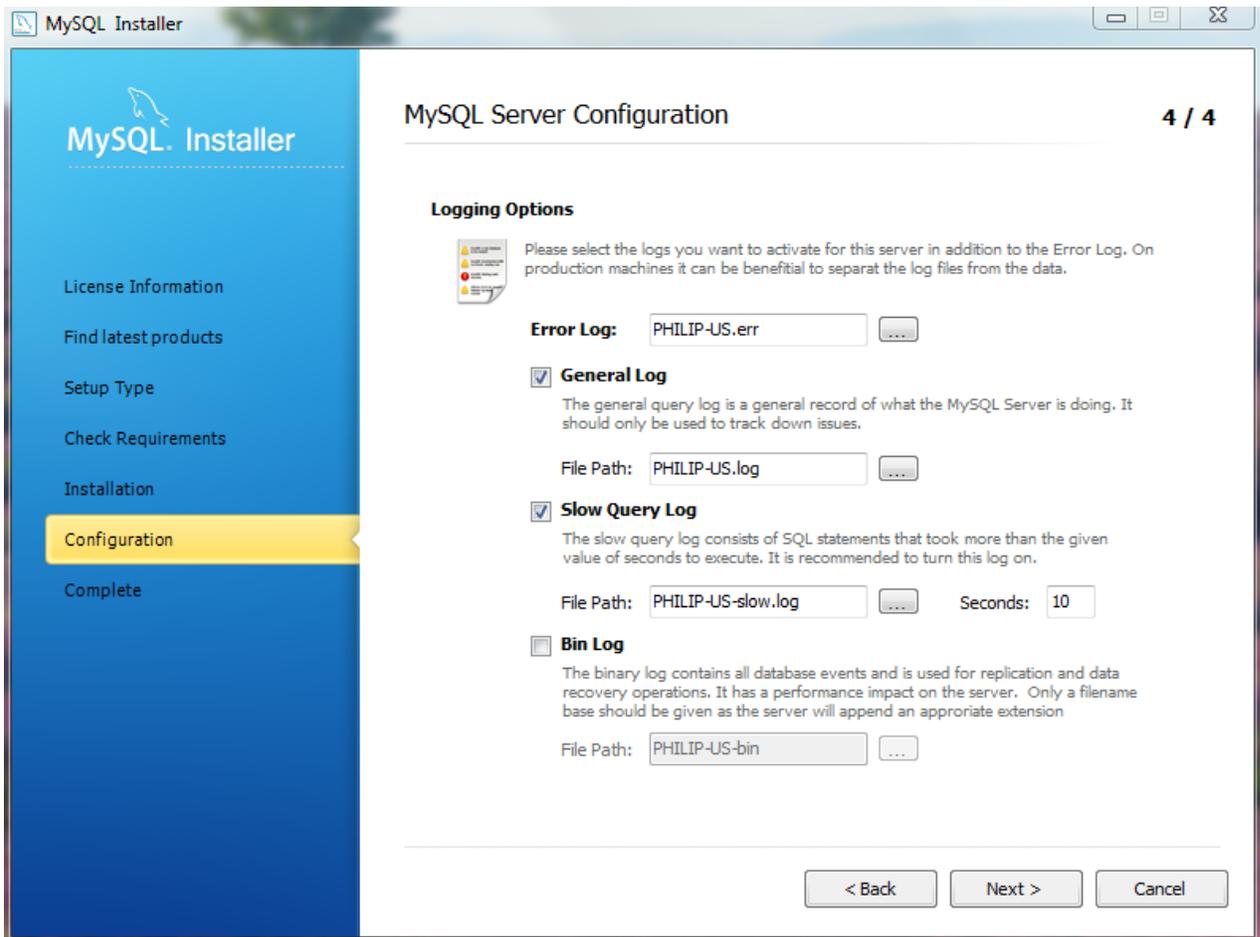


Note

When configuring **Run Windows Services as ...** using a **Custom User**, the custom user must have privileges to log on to Windows as a service. And the **Next** button will be disabled until this user is given these user rights.

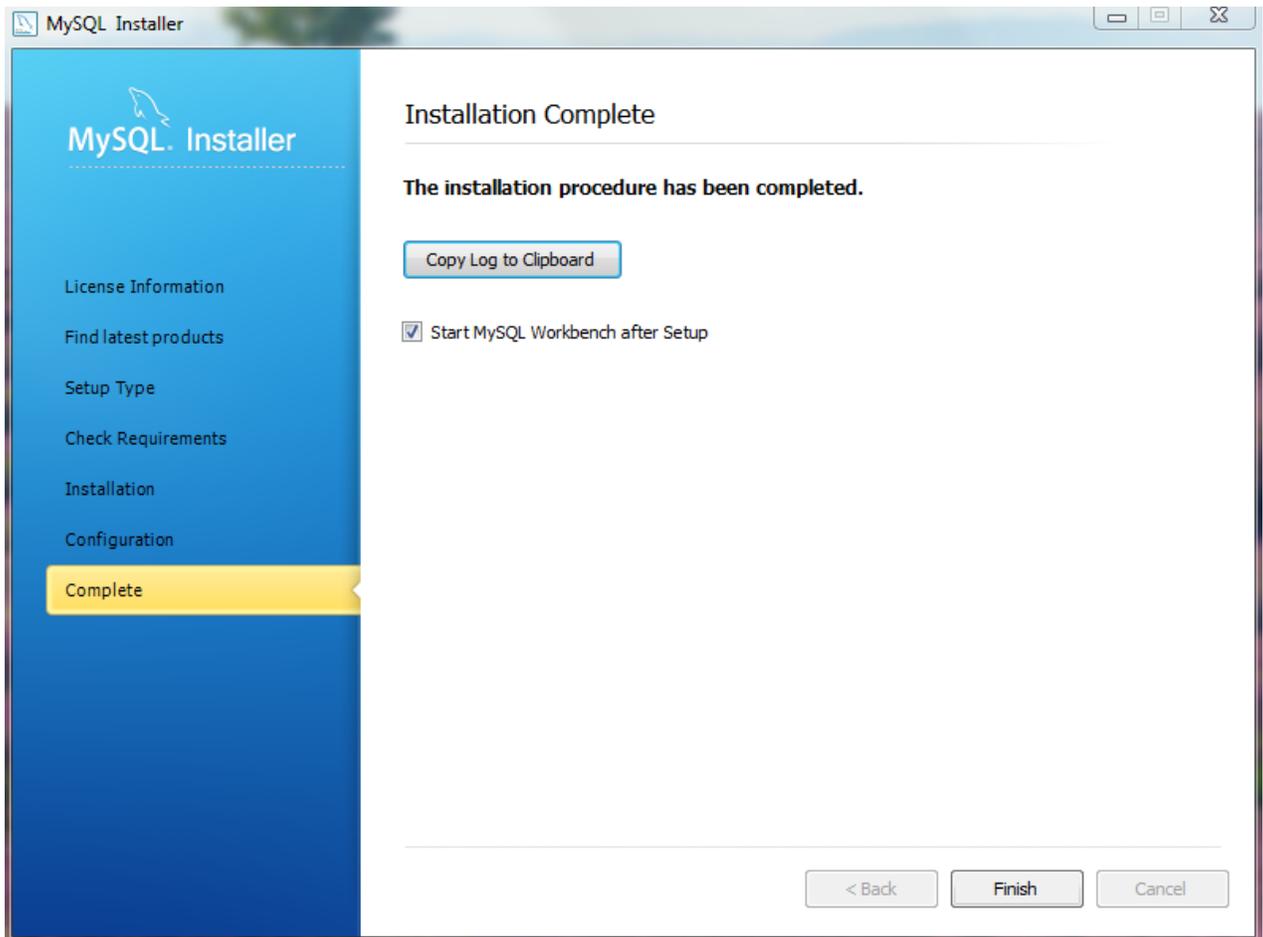
On Microsoft Windows 7, this is configured by loading the [Start Menu](#), [Control Panel](#), [Administrative Tools](#), [Local Security Policy](#), [Local Policies](#), [User Rights Assignment](#), then [Log On As A Service](#). Choose [Add User or Group](#) here to add the custom user, and then **OK**, **OK** to save.

The final configuration step is available if the **Advanced Configuration** option was checked, and it includes configuration options related to log file names:

Figure 2.12. MySQL Installer - MySQL Server Configuration: Logging options

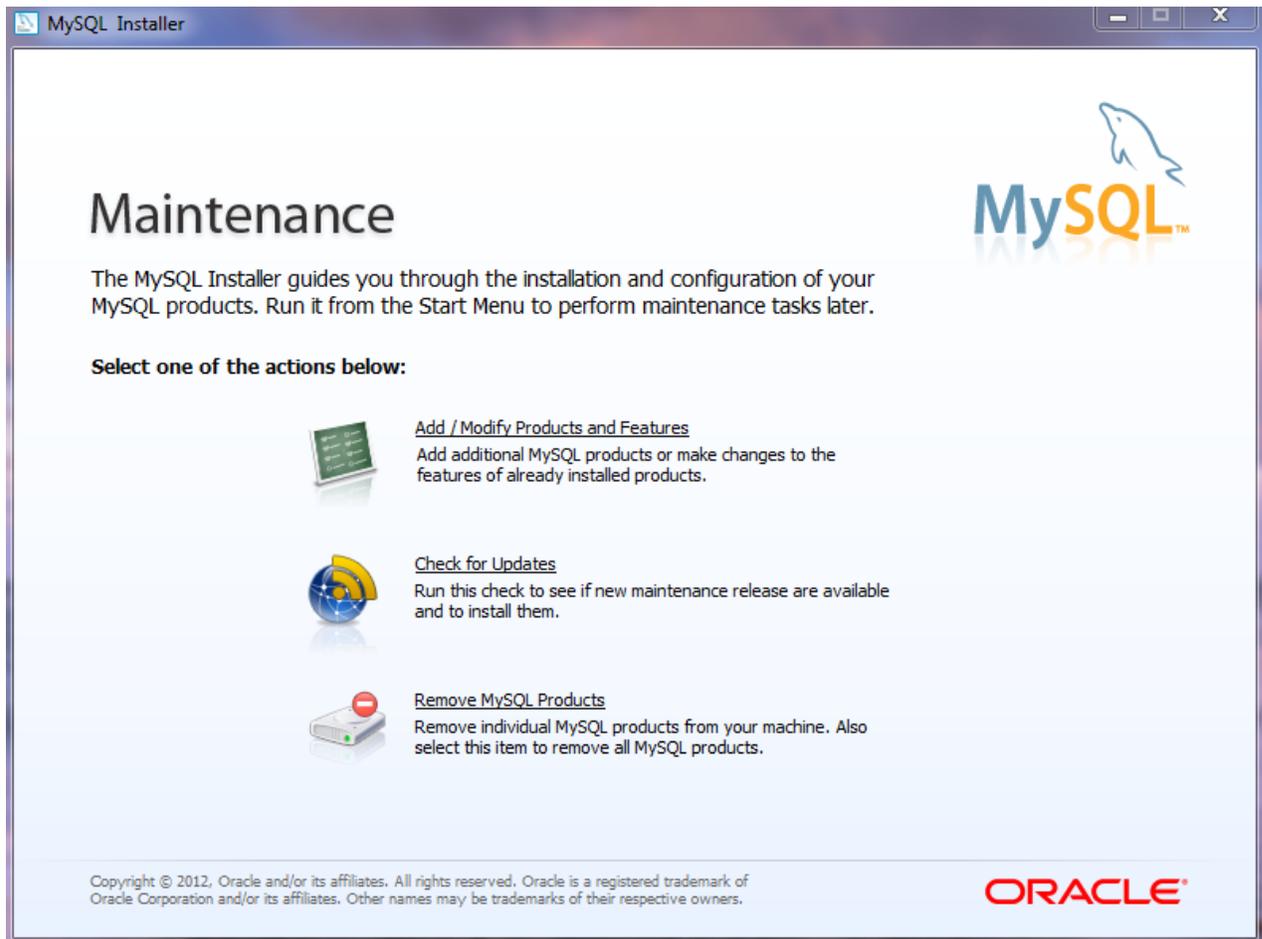
After the MySQL Installer configuration process is completed, you may save the installation log, and then load MySQL Workbench if the **Start MySQL Workbench after Setup** option is checked:

Figure 2.13. MySQL Installer - Installation Complete



You can now open MySQL Installer from the Microsoft Windows Start menu under the [MySQL](#) group, which will load the MySQL Installer [Maintenance Screen](#). This is used to add, update, and remove features.

Figure 2.14. MySQL Installer - Maintenance Screen



Note

An [Update Screen](#) screen is shown if MySQL Installer is used on a machine with older products installed, as opposed to the [Maintenance Screen](#) shown above. However, the functionality remains the same.

See also the MySQL Workbench documentation (<http://dev.mysql.com/doc/workbench/en/>).

2.3.3.2. MySQL Installer Console

`MySQLInstallerConsole` provides functionality similar to the GUI version of MySQL Installer, but from the command-line. It is installed when MySQL Installer is initially executed, and then available within the `MySQL Installer` directory. Typically that is in `C:\Program Files (x86)\MySQL\MySQL Installer\`, and the console must be executed with administrative privileges.

To use, invoke the Command Prompt with administrative privileges by choosing **Start**, **Accessories**, then right-click on **Command Prompt** and choose **Run as administrator**. And from the command-line, optionally change the directory to where `MySQLInstallerConsole` is located:

```
C:\> cd "C:\Program Files (x86)\MySQL\MySQL Installer"
```

`MySQLInstallerConsole` supports the following options, which are specified on the command line:

- `--help, -h, or -?`

Displays a help message with usage examples, and then exits.

```
C:\> MySQLInstallerConsole --help
```

- `--updates (or -u)`

Checks for new products before any further action is taken. Disabled by default.

- `--nowait`

Skips the final pause when the program finishes. Otherwise, a "Press Enter to continue." dialogue is generated. It is used in conjunction with other options.

- `--catalog=catalog_name (or -c)`

Sets the default catalog. Use `--list` to view a list of available catalogs.

- `--type=installation_type (or -t)`

Sets the installation type.

The possible values for `installation_type` are: developer, server, client, full, and custom.

- `--action=action_name`

The action being performed.

The possible values are: install, remove, upgrade, list, and status.

- **install:** Installs a product or products, as defined by `--products`
- **upgrade:** Upgrades a product or products, as defined by `--products`.
- **remove:** Removes a product or products, as defined by `--products`.
- **list:** Lists the product manifest, both installed and available products.
- **status:** Shows the status after another action is performed.

- `--product=product_name[:feature1],[feature2], [...]` (or `-p`)

Set the feature list of a product. Use `--list` to view available products, or pass in `--product=*` (an asterisk) to install all available products.

- `--config=product_name:passwd=root_password[;parameter1=value], [;parameter2=value], ...`

The configuration parameters for the most recently listed products.

- `--user=product_name:name=username,host:hostname,role=rolename,password=password` or `--user=product_name:name=username,host:hostname,role=rolename,tokens=tokens`

Creates a new user.

Requires: name, host, role, and the password or tokens. Tokens are separated by pipe (|) characters.

2.3.4. Installing MySQL on Microsoft Windows Using an MSI Package

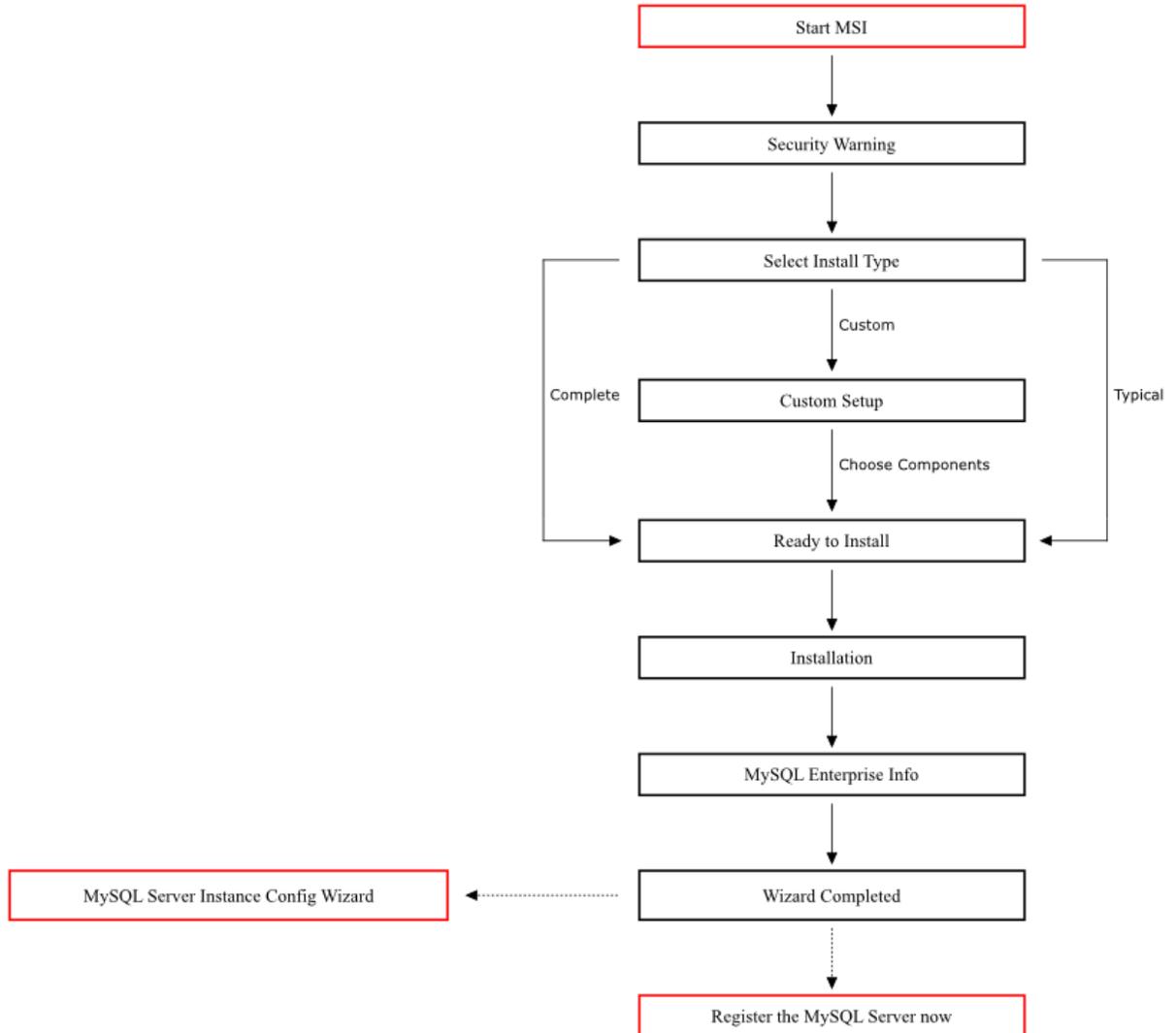
The MSI package is designed to install and configure MySQL in such a way that you can immediately get started using MySQL.

The MySQL Installation Wizard and MySQL Configuration Wizard are available in the Complete install package, which is recommended for most standard MySQL installations. Exceptions include users who need to install multiple instances of MySQL on a single server host and advanced users who want complete control of server configuration.

- For information on installing using the GUI MSI installer process, see [Section 2.3.4.1, “Using the MySQL Installation Wizard”](#).
- For information on installing using the command line using the MSI package, see [Section 2.3.4.2, “Automating MySQL Installation on Microsoft Windows using the MSI Package”](#).
- If you have previously installed MySQL using the MSI package and want to remove MySQL, see [Section 2.3.4.3, “Removing MySQL When Installed from the MSI Package”](#).

The workflow sequence for using the installer is shown in the figure below:

Figure 2.15. Installation Workflow for Windows using MSI Installer



Note

Microsoft Windows XP and later include a firewall which specifically blocks ports. If you plan on using MySQL through a network port then you should open and create an exception for this port before performing the installation. To check and if necessary add an exception to the firewall settings:

1. First ensure that you are logged in as an Administrator or a user with Administrator privileges.
2. Go to the **Control Panel**, and double click the **Windows Firewall** icon.
3. Choose the **Allow a program through Windows Firewall** option and click the **Add port** button.
4. Enter **MySQL** into the **Name** text box and **3306** (or the port of your choice) into the **Port number** text box.

5. Also ensure that the **TCP** protocol radio button is selected.
6. If you wish, you can also limit access to the MySQL server by choosing the **Change scope** button.
7. Confirm your choices by clicking the **OK** button.

Additionally, when running the MySQL Installation Wizard on Windows Vista or newer, ensure that you are logged in as a user with administrative rights.



Note

When using Windows Vista or newer, you may want to disable User Account Control (UAC) before performing the installation. If you do not do so, then MySQL may be identified as a security risk, which will mean that you need to enable MySQL. You can disable the security checking by following these instructions:

1. Open **Control Panel**.
2. Under the **User Accounts and Family Safety**, select **Add or remove user accounts**.
3. Click the **Got to the main User Accounts page** link.
4. Click on **Turn User Account Control on or off**. You may be prompted to provide permission to change this setting. Click **Continue**.
5. Deselect or uncheck the check box next to **Use User Account Control (UAC) to help protect your computer**. Click **OK** to save the setting.

You will need to restart to complete the process. Click **Restart Now** to reboot the machine and apply the changes. You can then follow the instructions below for installing Windows.

2.3.4.1. Using the MySQL Installation Wizard

MySQL Installation Wizard is an installer for the MySQL server that uses the latest installer technologies for Microsoft Windows. The MySQL Installation Wizard, in combination with the MySQL Configuration Wizard, enables a user to install and configure a MySQL server that is ready for use immediately after installation.

The MySQL Installation Wizard is the standard installer for all MySQL server distributions, version 4.1.5 and higher. Users of previous versions of MySQL need to shut down and remove their existing MySQL installations manually before installing MySQL with the MySQL Installation Wizard. See [Section 2.3.4.1.6, "Upgrading MySQL with the Installation Wizard"](#), for more information on upgrading from a previous version.

Microsoft has included an improved version of their Microsoft Windows Installer (MSI) in the recent versions of Windows. MSI has become the de-facto standard for application installations on Windows 2000, Windows XP, and Windows Server 2003. The MySQL Installation Wizard makes use of this technology to provide a smoother and more flexible installation process.

The Microsoft Windows Installer Engine was updated with the release of Windows XP; those using a previous version of Windows can reference [this Microsoft Knowledge Base article](#) for information on upgrading to the latest version of the Windows Installer Engine.

In addition, Microsoft has introduced the WiX (Windows Installer XML) toolkit recently. This is the first highly acknowledged Open Source project from Microsoft. We have switched to WiX because it is an Open Source project and it enables us to handle the complete Windows installation process in a flexible manner using scripts.

Improving the MySQL Installation Wizard depends on the support and feedback of users like you. If you find that the MySQL Installation Wizard is lacking some feature important to you, or if you discover a bug, please report it in our bugs database using the instructions given in [Section 1.7, “How to Report Bugs or Problems”](#).

2.3.4.1.1. Downloading and Starting the MySQL Installation Wizard

The MySQL installation packages can be downloaded from <http://dev.mysql.com/downloads/>. If the package you download is contained within a Zip archive, you need to extract the archive first.



Note

If you are installing on Windows Vista or newer, it is best to open a network port before beginning the installation. To do this, first ensure that you are logged in as an Administrator, go to the [Control Panel](#), and double-click the [Windows Firewall](#) icon. Choose the [Allow a program through Windows Firewall](#) option and click the [Add port](#) button. Enter [MySQL](#) into the **Name** text box and [3306](#) (or the port of your choice) into the **Port number** text box. Also ensure that the **TCP** protocol radio button is selected. If you wish, you can also limit access to the MySQL server by choosing the **Change scope** button. Confirm your choices by clicking the [OK](#) button. If you do not open a port prior to installation, you cannot configure the MySQL server immediately after installation. Additionally, when running the MySQL Installation Wizard on Windows Vista or newer, ensure that you are logged in as a user with administrative rights.

The process for starting the wizard depends on the contents of the installation package you download. If there is a [setup.exe](#) file present, double-click it to start the installation process. If there is an [.msi](#) file present, double-click it to start the installation process.

2.3.4.1.2. Choosing an Install Type

There are three installation types available: **Typical**, **Complete**, and **Custom**.

The **Typical** installation type installs the MySQL server, the [mysql](#) command-line client, and the command-line utilities. The command-line clients and utilities include [mysqldump](#), [myisamchk](#), and several other tools to help you manage the MySQL server.

The **Complete** installation type installs all components included in the installation package. The full installation package includes components such as the embedded server library, the benchmark suite, support scripts, and documentation.

The **Custom** installation type gives you complete control over which packages you wish to install and the installation path that is used. See [Section 2.3.4.1.3, “The Custom Install Dialog”](#), for more information on performing a custom install.

If you choose the **Typical** or **Complete** installation types and click the [Next](#) button, you advance to the confirmation screen to verify your choices and begin the installation. If you choose the **Custom** installation type and click the [Next](#) button, you advance to the custom installation dialog, described in [Section 2.3.4.1.3, “The Custom Install Dialog”](#).

2.3.4.1.3. The Custom Install Dialog

If you wish to change the installation path or the specific components that are installed by the MySQL Installation Wizard, choose the **Custom** installation type.

A tree view on the left side of the custom install dialog lists all available components. Components that are not installed have a red X icon; components that are installed have a gray icon. To change whether a component is installed, click that component's icon and choose a new option from the drop-down list that appears.

You can change the default installation path by clicking the **Change...** button to the right of the displayed installation path.

After choosing your installation components and installation path, click the **Next** button to advance to the confirmation dialog.

2.3.4.1.4. The Confirmation Dialog

Once you choose an installation type and optionally choose your installation components, you advance to the confirmation dialog. Your installation type and installation path are displayed for you to review.

To install MySQL if you are satisfied with your settings, click the **Install** button. To change your settings, click the **Back** button. To exit the MySQL Installation Wizard without installing MySQL, click the **Cancel** button.

The final screen of the installer provides a summary of the installation and gives you the option to launch the MySQL Configuration Wizard, which you can use to create a configuration file, install the MySQL service, and configure security settings.

2.3.4.1.5. Changes Made by MySQL Installation Wizard

Once you click the **Install** button, the MySQL Installation Wizard begins the installation process and makes certain changes to your system which are described in the sections that follow.

Changes to the Registry

The MySQL Installation Wizard creates one Windows registry key in a typical install situation, located in `HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB`.

The MySQL Installation Wizard creates a key named after the major version of the server that is being installed, such as `MySQL Server 5.5`. It contains two string values, `Location` and `Version`. The `Location` string contains the path to the installation directory. In a default installation it contains `C:\Program Files\MySQL\MySQL Server 5.5\`. The `Version` string contains the release number. For example, for an installation of MySQL Server 5.5.31, the key contains a value of `5.5.31`.

These registry keys are used to help external tools identify the installed location of the MySQL server, preventing a complete scan of the hard-disk to determine the installation path of the MySQL server. The registry keys are not required to run the server, and if you install MySQL using the `noinstall` Zip archive, the registry keys are not created.

Changes to the Start Menu

The MySQL Installation Wizard creates a new entry in the Windows **Start** menu under a common MySQL menu heading named after the major version of MySQL that you have installed. For example, if you install MySQL 5.5, the MySQL Installation Wizard creates a MySQL Server 5.5 section in the **Start** menu.

The following entries are created within the new **Start** menu section:

- MySQL Command Line Client: This is a shortcut to the `mysql` command-line client and is configured to connect as the `root` user. The shortcut prompts for a `root` user password when you connect.
- MySQL Server Instance Config Wizard: This is a shortcut to the MySQL Configuration Wizard. Use this shortcut to configure a newly installed server, or to reconfigure an existing server.
- MySQL Documentation: This is a link to the MySQL server documentation that is stored locally in the MySQL server installation directory.

Changes to the File System

The MySQL Installation Wizard by default installs the MySQL 5.5 server to `C:\Program Files\MySQL\MySQL Server 5.5`, where *Program Files* is the default location for applications in your system, and *5.5* is the major version of your MySQL server. This is the recommended location for the MySQL server, replacing the former default location `C:\mysql`.

By default, all MySQL applications are stored in a common directory at `C:\Program Files\MySQL`, where *Program Files* is the default location for applications in your Windows installation. A typical MySQL installation on a developer machine might look like this:

```
C:\Program Files\MySQL\MySQL Server 5.5
C:\Program Files\MySQL\MySQL Workbench 5.1 OSS
```

This approach makes it easier to manage and maintain all MySQL applications installed on a particular system.

The default location of the data directory is the `AppData` directory configured for the user that installed the MySQL application.

2.3.4.1.6. Upgrading MySQL with the Installation Wizard

The MySQL Installation Wizard can perform server upgrades automatically using the upgrade capabilities of MSI. That means you do not need to remove a previous installation manually before installing a new release. The installer automatically shuts down and removes the previous MySQL service before installing the new version.

Automatic upgrades are available only when upgrading between installations that have the same major and minor version numbers. For example, you can upgrade automatically from MySQL 5.5.5 to MySQL 5.5.6, but not from MySQL 5.1 to MySQL 5.5.

See [Section 2.3.9, "Upgrading MySQL on Windows"](#).

2.3.4.2. Automating MySQL Installation on Microsoft Windows using the MSI Package

The Microsoft Installer (MSI) supports both a *quiet* and a *passive* mode that can be used to install MySQL automatically without requiring intervention. You can use this either in scripts to automatically install MySQL or through a terminal connection such as Telnet where you do not have access to the standard Windows user interface. The MSI packages can also be used in combination with Microsoft's Group Policy system (part of Windows Server 2003 and Windows Server 2008) to install MySQL across multiple machines.

To install MySQL from one of the MSI packages automatically from the command line (or within a script), you need to use the `msiexec.exe` tool. For example, to perform a quiet installation (which shows no dialog boxes or progress):

```
shell> msiexec /i /quiet mysql-5.5.31.msi
```

The `/i` indicates that you want to perform an installation. The `/quiet` option indicates that you want no interactive elements.

To provide a dialog box showing the progress during installation, and the dialog boxes providing information on the installation and registration of MySQL, use `/passive` mode instead of `/quiet`:

```
shell> msiexec /i /passive mysql-5.5.31.msi
```

Regardless of the mode of the installation, installing the package in this manner performs a 'Typical' installation, and installs the default components into the standard location.

You can also use this method to uninstall MySQL by using the `/uninstall` or `/x` options:

```
shell> msiexec /x /quiet mysql-5.5.31.msi
```

To install MySQL and configure a MySQL instance from the command line, see [Section 2.3.6.13, "MySQL Server Instance Config Wizard: Creating an Instance from the Command Line"](#).

For information on using MSI packages to install software automatically using Group Policy, see [How to use Group Policy to remotely install software in Windows Server 2003](#).

2.3.4.3. Removing MySQL When Installed from the MSI Package

To uninstall a MySQL where you have used the MSI packages, you must use the **Add/Remove Programs** tool within **Control Panel**. To do this:

1. Right-click the **start** menu and choose **Control Panel**.
2. If the Control Panel is set to category mode (you will see **Pick a category** at the top of the **Control Panel** window), double-click **Add or Remove Programs**. If the Control is set to classic mode, double-click the **Add or Remove Programs** icon.
3. Find MySQL in the list of installed software. MySQL Server is installed against major version numbers (MySQL 5.1, MySQL 5.5, etc.). Select the version that you want to remove and click **Remove**.
4. You will be prompted to confirm the removal. Click **Yes** to remove MySQL.

When MySQL is removed using this method, only the installed components are removed. Any database information (including the tables and data), import or export files, log files, and binary logs produced during execution are kept in their configured location.

If you try to install MySQL again the information will be retained and you will be prompted to enter the password configured with the original installation.

If you want to delete MySQL completely:

- Delete the associated data directory. On Windows XP and Windows Server 2003, the default data directory is the configured AppData directory, which is `C:\Documents and Settings\All Users\Application Data\MySQL` by default.
- On Windows 7 and Windows Server 2008, the default data directory location is `C:\ProgramData\MySQL`.



Note

The `C:\ProgramData` directory is hidden by default. You must change your folder options to view the hidden file. Choose **Organize**, Folder and search options, **Show hidden folders**.

2.3.5. MySQL Notifier for Microsoft Windows

The MySQL Notifier for Microsoft Windows is a tool that allows you to monitor and adjust the status of your local MySQL Server instances through an indicator that resides in the system tray. The MySQL Notifier for Microsoft Windows also gives quick access to several MySQL GUI tools (such as MySQL Workbench) through its context menu.

The MySQL Notifier for Microsoft Windows is installed by MySQL Installer, and (by default) will start-up when Microsoft Windows is started.



Note

To install, download and execute the [MySQL Installer](#), be sure the MySQL Notifier for Microsoft Windows product is selected, then proceed with the installation. See the [MySQL Installer manual](#) for additional details.

For release notes detailing the changes in each release of MySQL Notifier for Microsoft Windows, see [MySQL Notifier Release Notes](#).

Visit the [MySQL Notifier forum](#) for additional MySQL Notifier for Microsoft Windows help and support.

Features include:

- Start, Stop, and Restart instances of the MySQL Server.
- Automatically detects (and adds) new MySQL Server services. These are listed under [Manage Services](#), and may also be configured.
- The Tray icon changes, depending on the status. It's green if all monitored MySQL Server instances are running, or red if at least one service is stopped.
- Links to other applications like MySQL Workbench, Windows Installer, and the MySQL Utilities. For example, choosing [Configure Instance](#) will load the MySQL Workbench Server Administration window for that particular instance.
- If MySQL Workbench is also installed, then the [Configure Instance](#) and [SQL Editor](#) options are available.

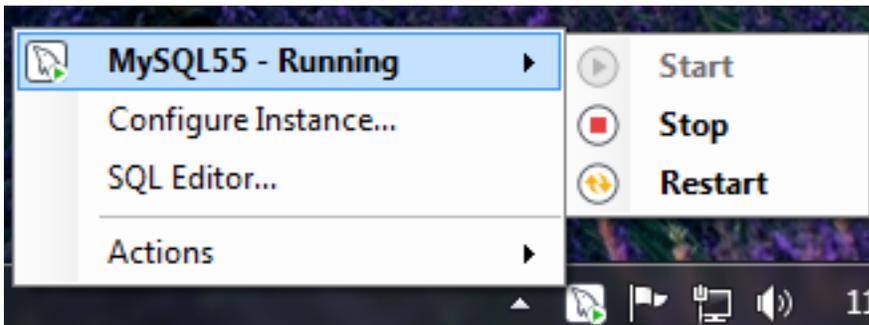
The MySQL Notifier for Microsoft Windows resides in the system tray and provides visual status information for your MySQL Server instances. A green icon is displayed at the top left corner of the tray icon if the current MySQL Server is running, or a red icon if the service is stopped.

The MySQL Notifier for Microsoft Windows automatically adds discovered MySQL Services on the local machine, and each service is saved and configurable. By default, the **Automatically add new services that match this pattern** option is enabled and set to `.*mysqld.*`. Related **Notifications Options** include being notified when new services are either discovered or experience status changes, and are also enabled by default. And uninstalling a service will also remove the service from the MySQL Notifier for Microsoft Windows.

Clicking the system tray icon will reveal several options, as seen in the screenshots below:

The Service Instance menu is the main MySQL Notifier for Microsoft Windows window, and allows you to Stop, Start, and Restart the MySQL Server.

Figure 2.16. MySQL Notifier for Microsoft Windows Service Instance menu



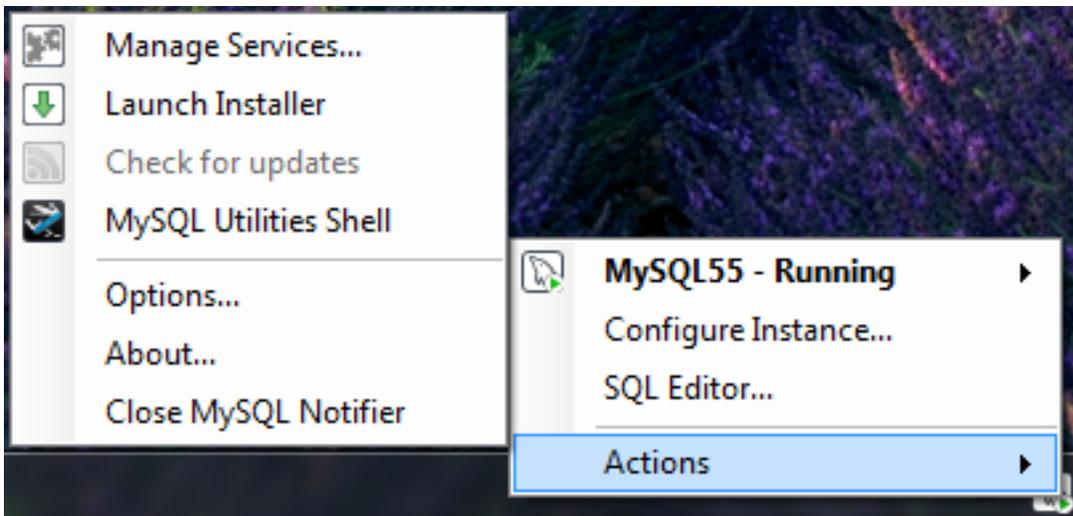
The Actions menu actions include several links to external applications (if they are installed).



Note

The main menu will not show the Actions menu when there are no services being monitored by MySQL Notifier for Microsoft Windows.

Figure 2.17. MySQL Notifier for Microsoft Windows Actions menu

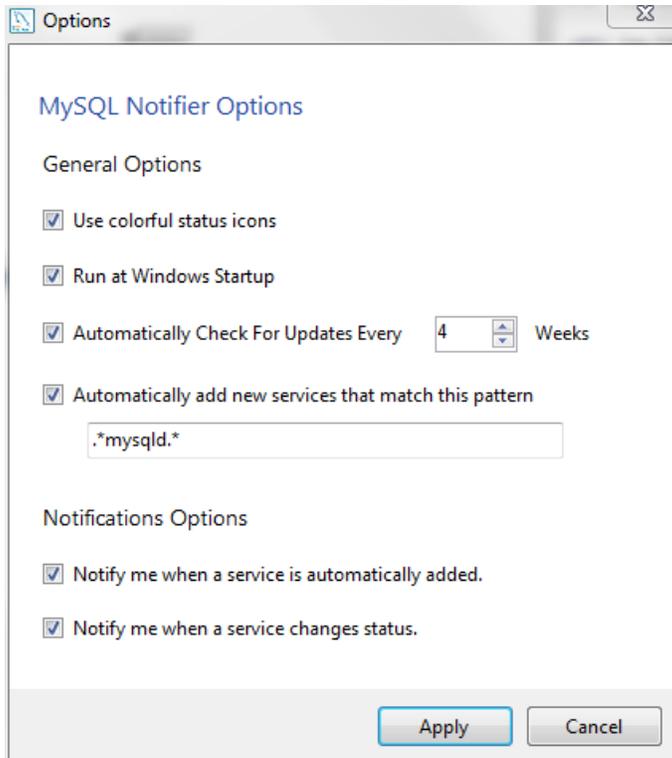


The Actions, Options menu configures MySQL Notifier for Microsoft Windows and includes options to:

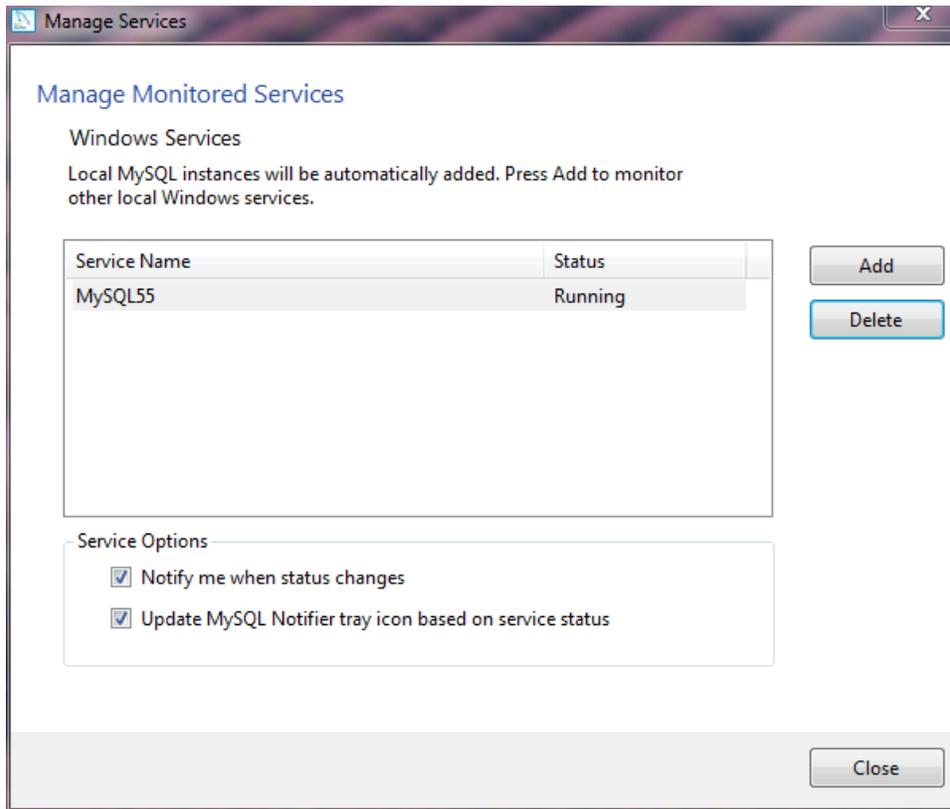
- **Use colorful status icons:** Enables a colorful style of icons for the tray of the MySQL Notifier for Microsoft Windows.
- **Run at Windows Startup:** Allows the application to be loaded when Microsoft Windows starts.
- **Automatically Check For Updates Every # Weeks:** Checks for a new version of MySQL Notifier for Microsoft Windows, and runs this check every # weeks.
- **Automatically add new services that match this pattern:** A custom regular expression that filters the monitored services, and also filters the list of the Microsoft Windows services for the **Add New Service** dialog.
- **Notify me when a service is automatically added:** Will display a notification from the taskbar when a newly discovered service is added to the monitored services list.

- **Notify me when a service changes status:** Will display a notification from the taskbar when a monitored service changes its status.

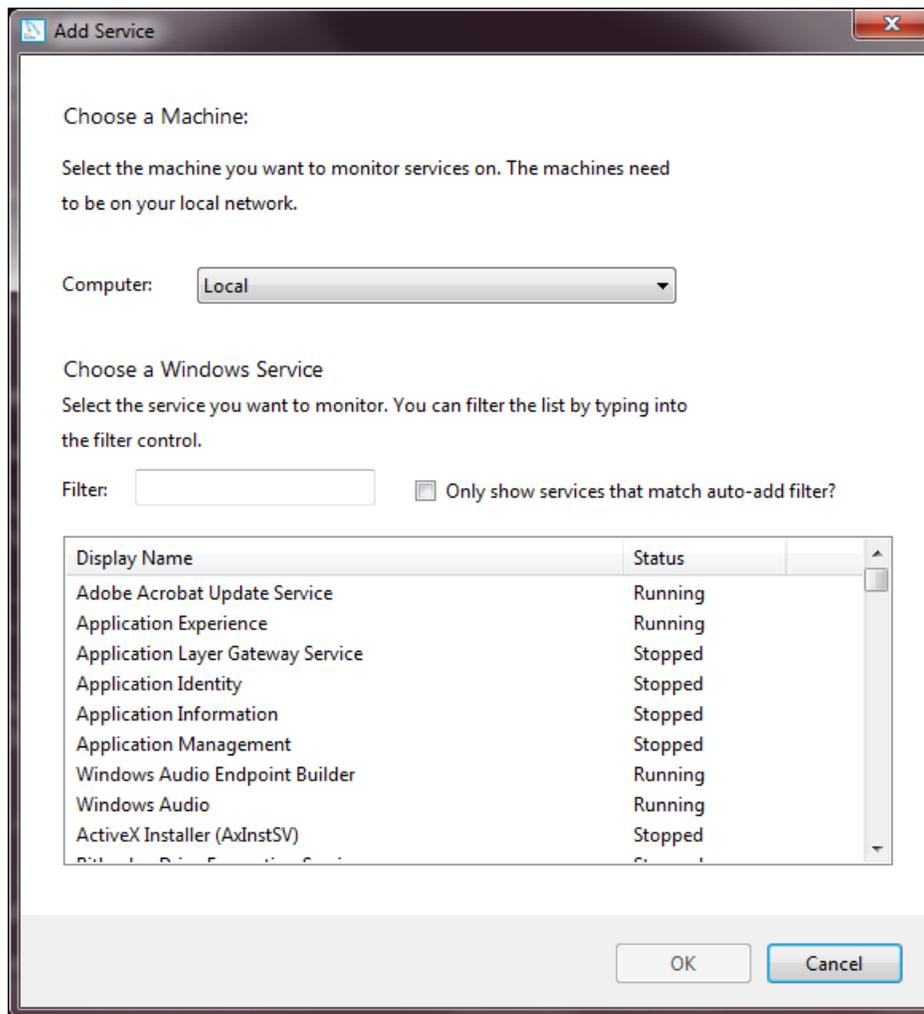
Figure 2.18. MySQL Notifier for Microsoft Windows Options menu



The Actions, Manage Services menu allows you to configure the monitored services.

Figure 2.19. MySQL Notifier for Microsoft Windows Service Instance menu

Adding a service (after clicking **Add** in the **Manage Services** window) allows you to select a running Microsoft Windows service, and configure MySQL Notifier for Microsoft Windows to monitor it. Add a new service by clicking service name from the list, then **OK** to accept. Multiple services may be selected.

Figure 2.20. MySQL Notifier for Microsoft Windows Adding a new services

2.3.6. MySQL Server Instance Configuration Wizard

The MySQL Server Instance Configuration Wizard helps automate the process of configuring your server. It creates a custom MySQL configuration file (`my.ini` or `my.cnf`) by asking you a series of questions and then applying your responses to a template to generate the configuration file that is tuned to your installation.

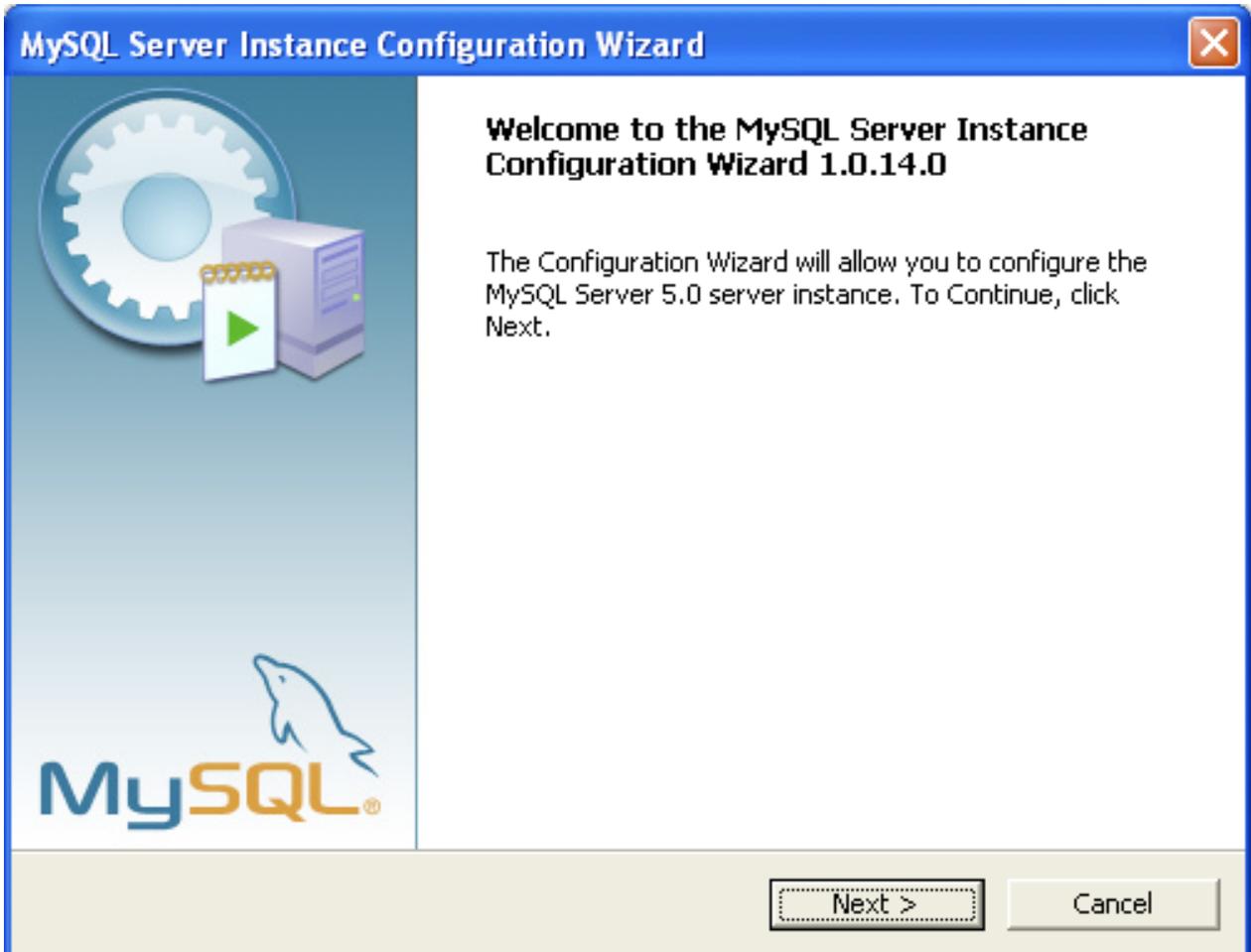
The MySQL Server Instance Configuration Wizard is included with the MySQL 5.5 server. The MySQL Server Instance Configuration Wizard is only available for Windows.

2.3.6.1. Starting the MySQL Server Instance Configuration Wizard

The MySQL Server Instance Configuration Wizard is normally started as part of the installation process. You should only need to run the MySQL Server Instance Configuration Wizard again when you need to change the configuration parameters of your server.

If you chose not to open a port prior to installing MySQL on Windows Vista or newer, you can choose to use the MySQL Server Configuration Wizard after installation. However, you must open a port in the Windows Firewall. To do this see the instructions given in [Section 2.3.4.1.1, "Downloading and Starting the](#)

MySQL Installation Wizard". Rather than opening a port, you also have the option of adding MySQL as a program that bypasses the Windows Firewall. One or the other option is sufficient—you need not do both. Additionally, when running the MySQL Server Configuration Wizard on Windows Vista or newer, ensure that you are logged in as a user with administrative rights.



You can launch the MySQL Configuration Wizard by clicking the MySQL Server Instance Config Wizard entry in the MySQL section of the Windows Start menu.

Alternatively, you can navigate to the `bin` directory of your MySQL installation and launch the `MySQLInstanceConfig.exe` file directly.

The MySQL Server Instance Configuration Wizard places the `my.ini` file in the installation directory for the MySQL server. This helps associate configuration files with particular server instances.

To ensure that the MySQL server knows where to look for the `my.ini` file, an argument similar to this is passed to the MySQL server as part of the service installation:

```
--defaults-file="C:\Program Files\MySQL\MySQL Server 5.5\my.ini"
```

Here, `C:\Program Files\MySQL\MySQL Server 5.5` is replaced with the installation path to the MySQL Server. The `--defaults-file [248]` option instructs the MySQL server to read the specified file for configuration options when it starts.

Apart from making changes to the `my.ini` file by running the MySQL Server Instance Configuration Wizard again, you can modify it by opening it with a text editor and making any necessary changes. You

can also modify the server configuration with the <http://www.mysql.com/products/administrator/> utility. For more information about server configuration, see [Section 5.1.3, “Server Command Options”](#).

MySQL clients and utilities such as the `mysql` and `mysqldump` command-line clients are not able to locate the `my.ini` file located in the server installation directory. To configure the client and utility applications, create a new `my.ini` file in the Windows installation directory (for example, `C:\WINDOWS`).

Under Windows Server 2003, Windows Server 2000, Windows XP, and Windows Vista, MySQL Server Instance Configuration Wizard will configure MySQL to work as a Windows service. To start and stop MySQL you use the [Services](#) application that is supplied as part of the Windows Administrator Tools.

2.3.6.2. Choosing a Maintenance Option

If the MySQL Server Instance Configuration Wizard detects an existing configuration file, you have the option of either reconfiguring your existing server, or removing the server instance by deleting the configuration file and stopping and removing the MySQL service.

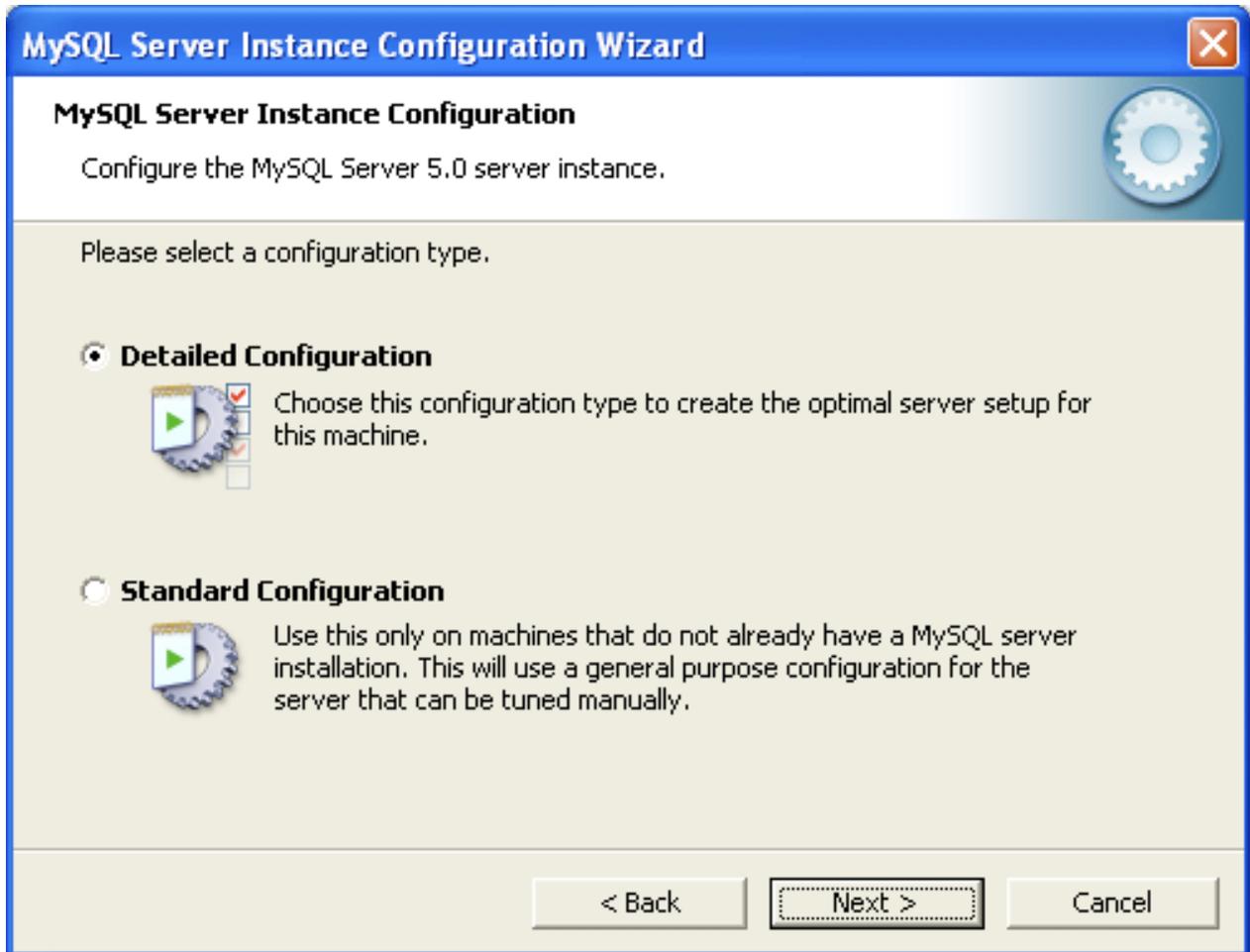
To reconfigure an existing server, choose the Re-configure Instance option and click the **Next** button. Any existing configuration file is not overwritten, but renamed (within the same directory) using a timestamp (Windows) or sequential number (Linux). To remove the existing server instance, choose the Remove Instance option and click the **Next** button.

If you choose the Remove Instance option, you advance to a confirmation window. Click the **Execute** button. The MySQL Server Configuration Wizard stops and removes the MySQL service, and then deletes the configuration file. The server installation and its `data` folder are not removed.

If you choose the Re-configure Instance option, you advance to the **Configuration Type** dialog where you can choose the type of installation that you wish to configure.

2.3.6.3. Choosing a Configuration Type

When you start the MySQL Server Instance Configuration Wizard for a new MySQL installation, or choose the Re-configure Instance option for an existing installation, you advance to the **Configuration Type** dialog.



There are two configuration types available: Detailed Configuration and Standard Configuration. The Standard Configuration option is intended for new users who want to get started with MySQL quickly without having to make many decisions about server configuration. The Detailed Configuration option is intended for advanced users who want more fine-grained control over server configuration.

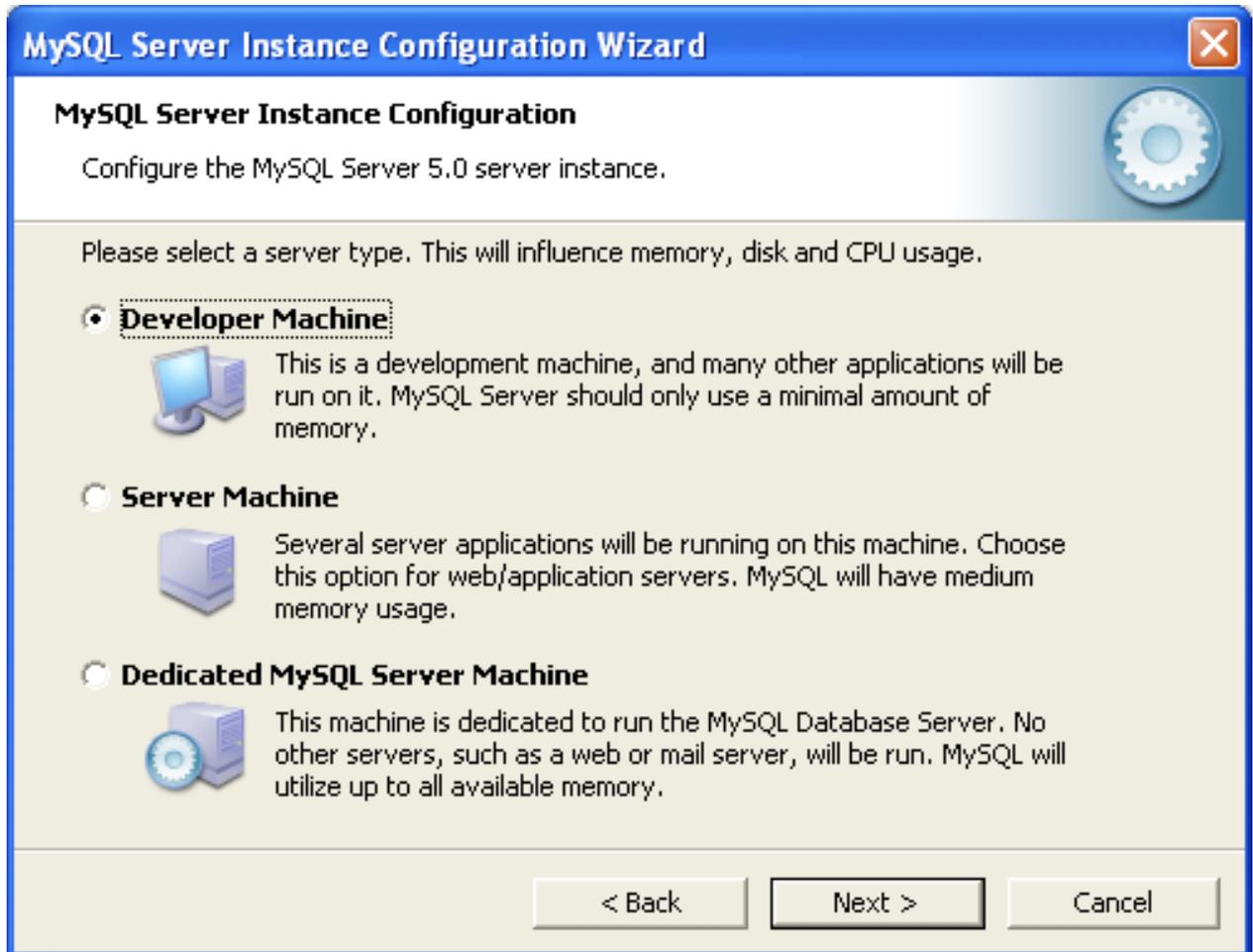
If you are new to MySQL and need a server configured as a single-user developer machine, the Standard Configuration should suit your needs. Choosing the Standard Configuration option causes the MySQL Configuration Wizard to set all configuration options automatically with the exception of Service Options and Security Options.

The Standard Configuration sets options that may be incompatible with systems where there are existing MySQL installations. If you have an existing MySQL installation on your system in addition to the installation you wish to configure, the Detailed Configuration option is recommended.

To complete the Standard Configuration, please refer to the sections on Service Options and Security Options in [Section 2.3.6.10, “The Service Options Dialog”](#), and [Section 2.3.6.11, “The Security Options Dialog”](#), respectively.

2.3.6.4. The Server Type Dialog

There are three different server types available to choose from. The server type that you choose affects the decisions that the MySQL Server Instance Configuration Wizard makes with regard to memory, disk, and processor usage.



- **Developer Machine:** Choose this option for a typical desktop workstation where MySQL is intended only for personal use. It is assumed that many other desktop applications are running. The MySQL server is configured to use minimal system resources.
- **Server Machine:** Choose this option for a server machine where the MySQL server is running alongside other server applications such as FTP, email, and Web servers. The MySQL server is configured to use a moderate portion of the system resources.
- **Dedicated MySQL Server Machine:** Choose this option for a server machine that is intended to run only the MySQL server. It is assumed that no other applications are running. The MySQL server is configured to use all available system resources.

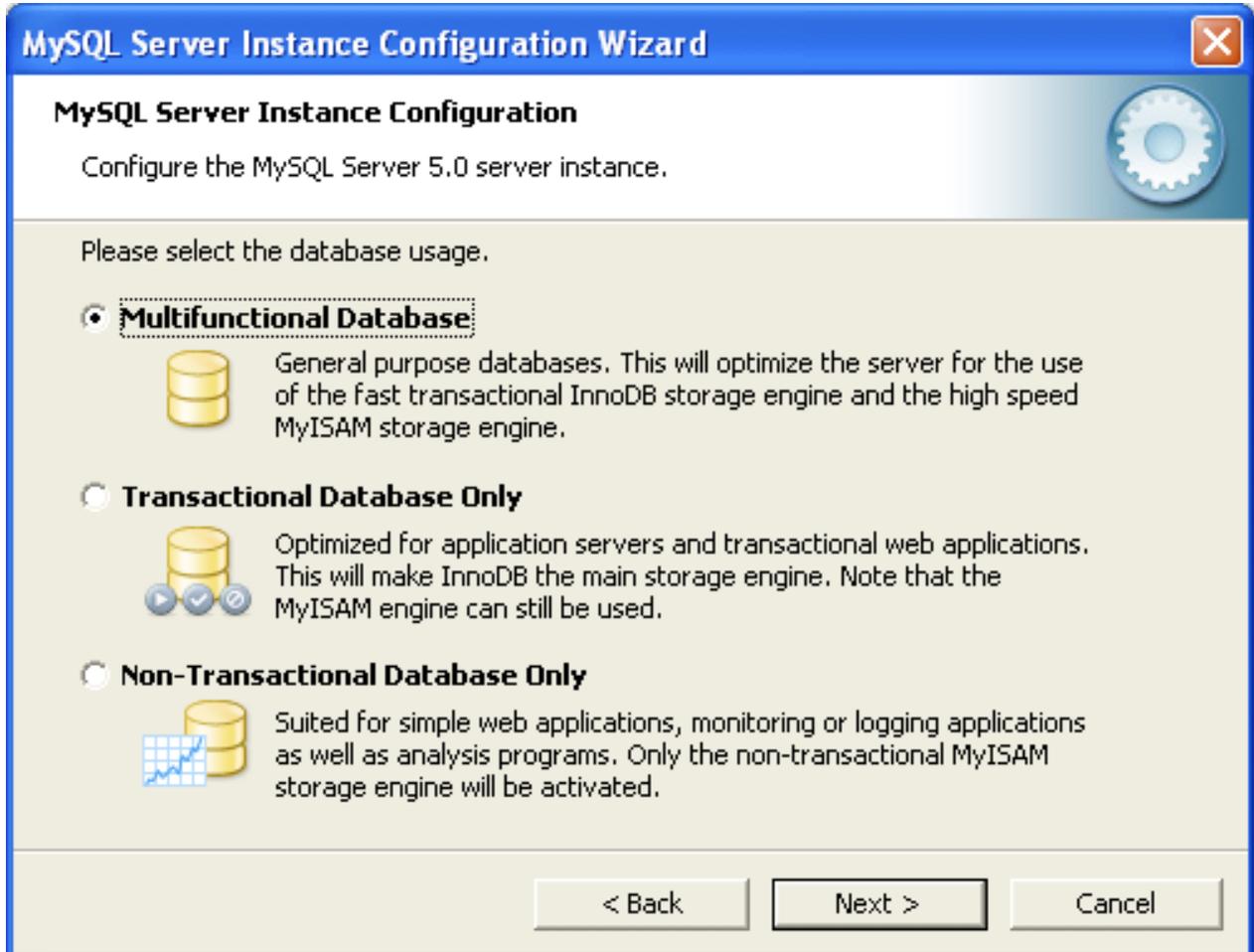


Note

By selecting one of the preconfigured configurations, the values and settings of various options in your `my.cnf` or `my.ini` will be altered accordingly. The default values and options as described in the reference manual may therefore be different to the options and values that were created during the execution of the configuration wizard.

2.3.6.5. The Database Usage Dialog

The [Database Usage](#) dialog enables you to indicate the storage engines that you expect to use when creating MySQL tables. The option you choose determines whether the [InnoDB](#) storage engine is available and what percentage of the server resources are available to [InnoDB](#).



- **Multifunctional Database:** This option enables both the [InnoDB](#) and [MyISAM](#) storage engines and divides resources evenly between the two. This option is recommended for users who use both storage engines on a regular basis.
- **Transactional Database Only:** This option enables both the [InnoDB](#) and [MyISAM](#) storage engines, but dedicates most server resources to the [InnoDB](#) storage engine. This option is recommended for users who use [InnoDB](#) almost exclusively and make only minimal use of [MyISAM](#).
- **Non-Transactional Database Only:** This option disables the [InnoDB](#) storage engine completely and dedicates all server resources to the [MyISAM](#) storage engine. This option is recommended for users who do not use [InnoDB](#).

The Configuration Wizard uses a template to generate the server configuration file. The [Database Usage](#) dialog sets one of the following option strings:

```
Multifunctional Database:      MIXED
Transactional Database Only:  INNODB
Non-Transactional Database Only: MYISAM
```

When these options are processed through the default template (my-template.ini) the result is:

```
Multifunctional Database:
default-storage-engine=InnoDB
_myisam_pct=50
```

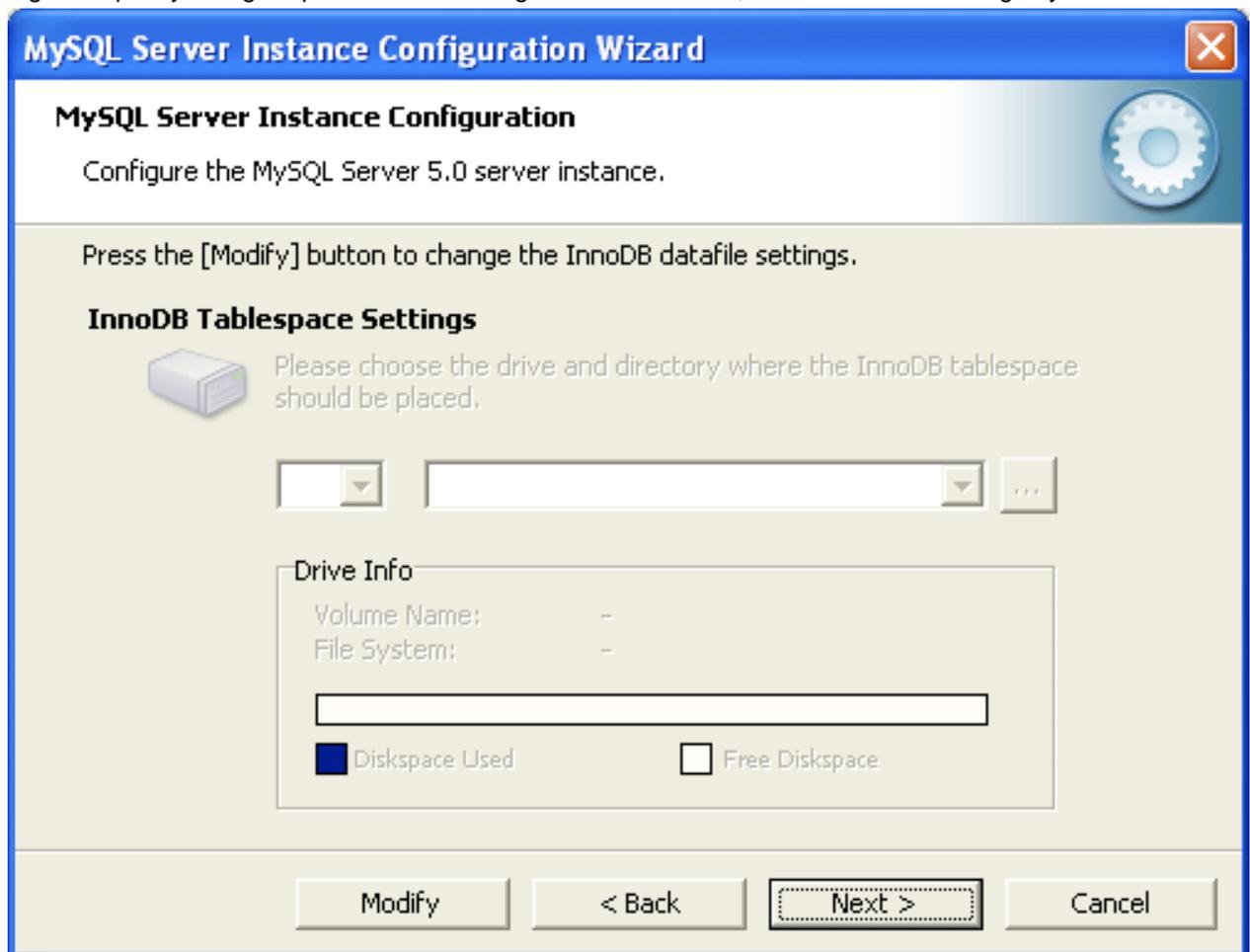
```
Transactional Database Only:
default-storage-engine=InnoDB
_myisam_pct=5
```

```
Non-Transactional Database Only:
default-storage-engine=MyISAM
_myisam_pct=100
skip-innodb
```

The `_myisam_pct` value is used to calculate the percentage of resources dedicated to `MyISAM`. The remaining resources are allocated to `InnoDB`.

2.3.6.6. The InnoDB Tablespace Dialog

Some users may want to locate the `InnoDB` tablespace files in a different location than the MySQL server data directory. Placing the tablespace files in a separate location can be desirable if your system has a higher capacity or higher performance storage device available, such as a RAID storage system.

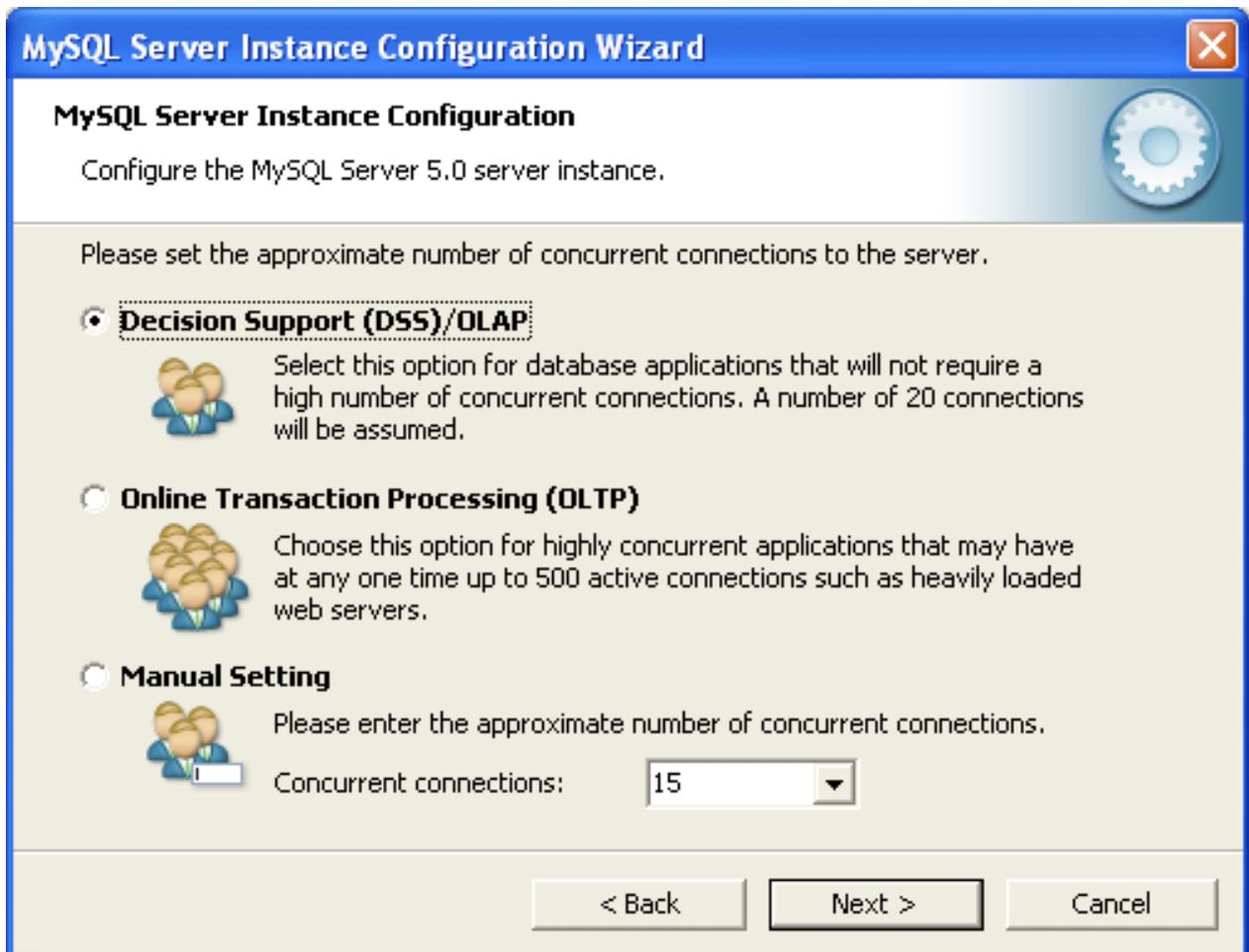


To change the default location for the `InnoDB` tablespace files, choose a new drive from the drop-down list of drive letters and choose a new path from the drop-down list of paths. To create a custom path, click the `...` button.

If you are modifying the configuration of an existing server, you must click the **Modify** button before you change the path. In this situation you must move the existing tablespace files to the new location manually before starting the server.

2.3.6.7. The Concurrent Connections Dialog

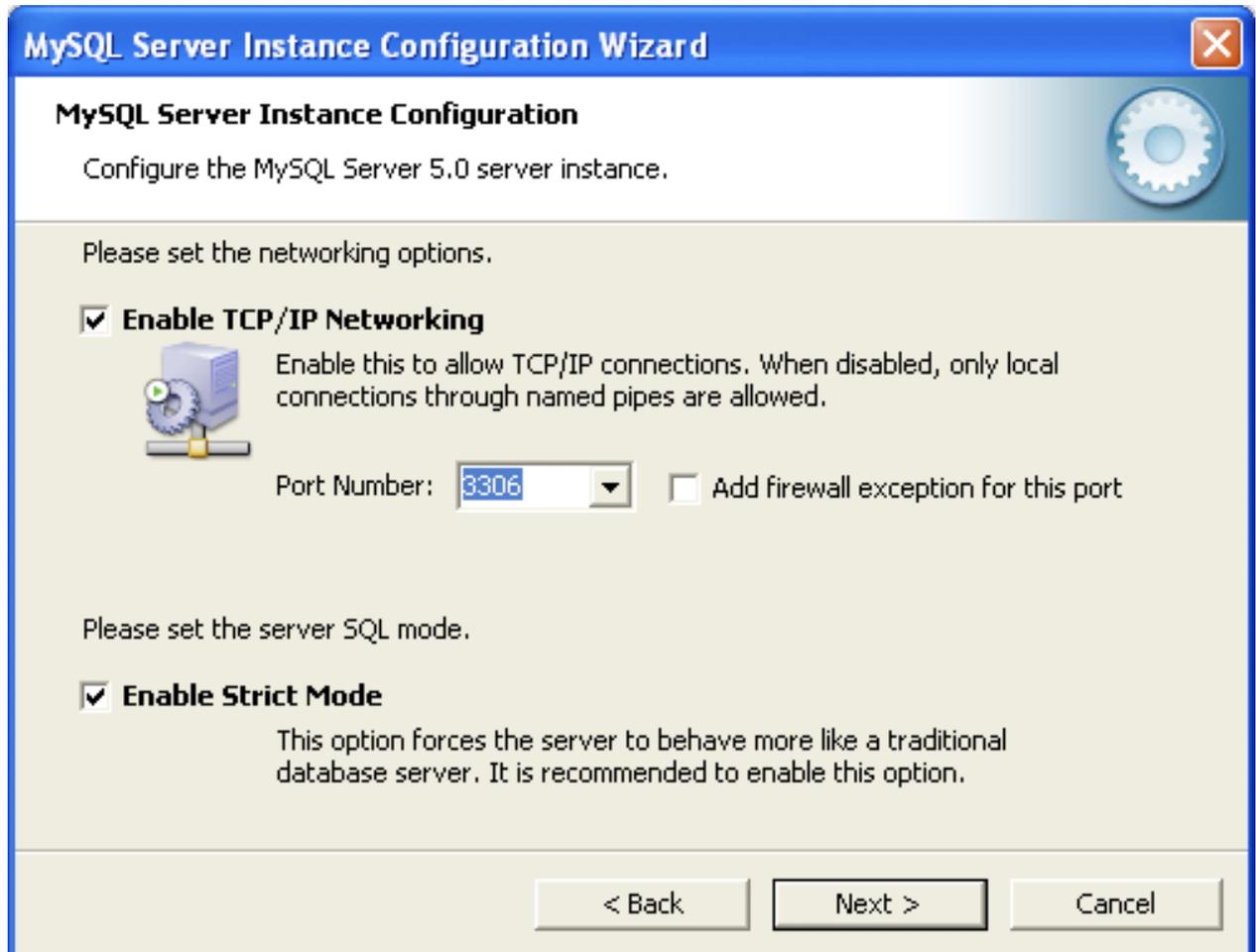
To prevent the server from running out of resources, it is important to limit the number of concurrent connections to the MySQL server that can be established. The **Concurrent Connections** dialog enables you to choose the expected usage of your server, and sets the limit for concurrent connections accordingly. It is also possible to set the concurrent connection limit manually.



- **Decision Support (DSS)/OLAP:** Choose this option if your server does not require a large number of concurrent connections. The maximum number of connections is set at 100, with an average of 20 concurrent connections assumed.
- **Online Transaction Processing (OLTP):** Choose this option if your server requires a large number of concurrent connections. The maximum number of connections is set at 500.
- **Manual Setting:** Choose this option to set the maximum number of concurrent connections to the server manually. Choose the number of concurrent connections from the drop-down box provided, or enter the maximum number of connections into the drop-down box if the number you desire is not listed.

2.3.6.8. The Networking and Strict Mode Options Dialog

Use the [Networking Options](#) dialog to enable or disable TCP/IP networking and to configure the port number that is used to connect to the MySQL server.



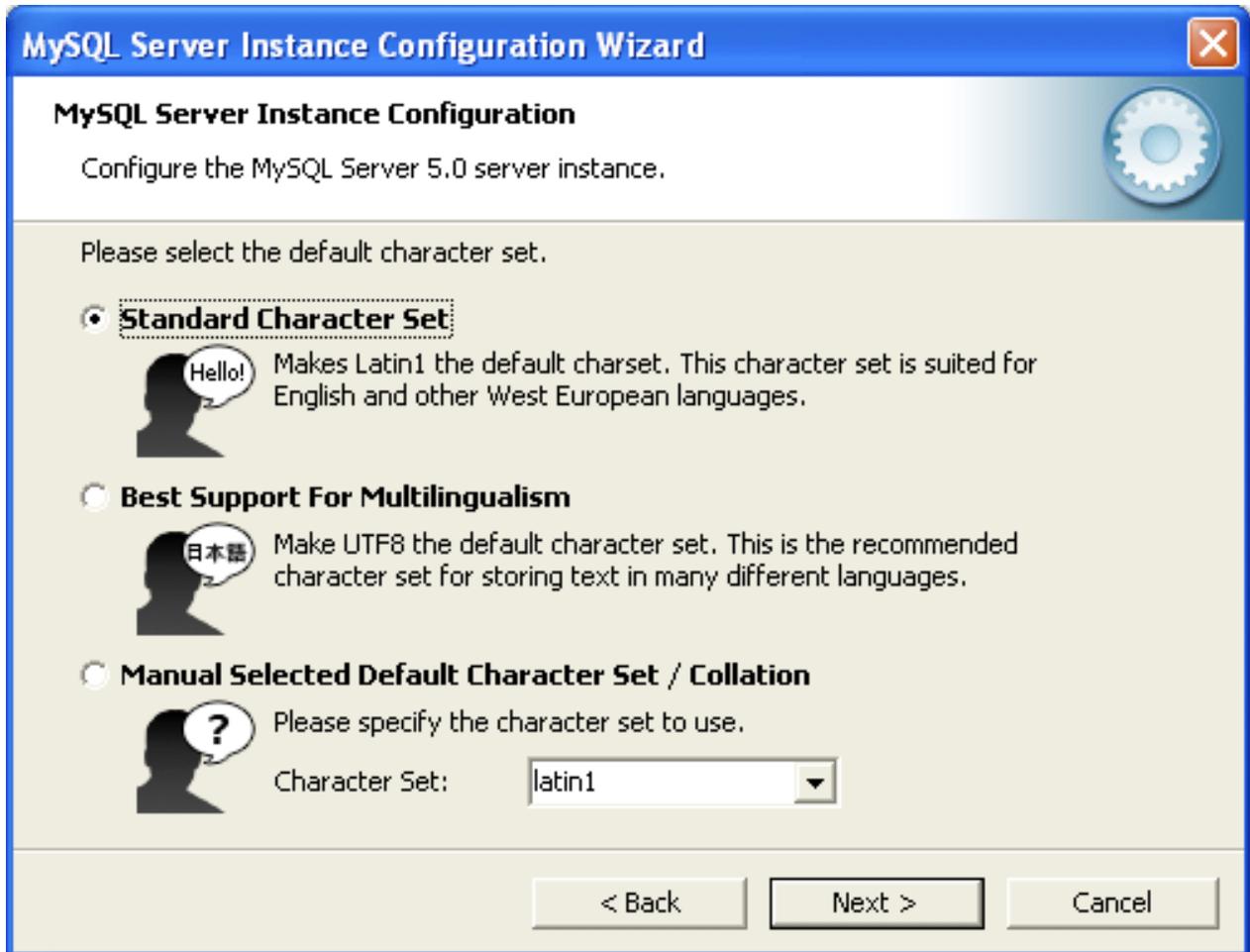
TCP/IP networking is enabled by default. To disable TCP/IP networking, uncheck the box next to the Enable TCP/IP Networking option.

Port 3306 is used by default. To change the port used to access MySQL, choose a new port number from the drop-down box or type a new port number directly into the drop-down box. If the port number you choose is in use, you are prompted to confirm your choice of port number.

Set the [Server SQL Mode](#) to either enable or disable strict mode. Enabling strict mode (default) makes MySQL behave more like other database management systems. *If you run applications that rely on MySQL's old "forgiving" behavior, make sure to either adapt those applications or to disable strict mode.* For more information about strict mode, see [Section 5.1.7, "Server SQL Modes"](#).

2.3.6.9. The Character Set Dialog

The MySQL server supports multiple character sets and it is possible to set a default server character set that is applied to all tables, columns, and databases unless overridden. Use the [Character Set](#) dialog to change the default character set of the MySQL server.



- **Standard Character Set:** Choose this option if you want to use `latin1` as the default server character set. `latin1` is used for English and many Western European languages.
- **Best Support For Multilingualism:** Choose this option if you want to use `utf8` as the default server character set. This is a Unicode character set that can store characters from many different languages.
- **Manual Selected Default Character Set / Collation:** Choose this option if you want to pick the server's default character set manually. Choose the desired character set from the provided drop-down list.

2.3.6.10. The Service Options Dialog

On Windows platforms, the MySQL server can be installed as a Windows service. When installed this way, the MySQL server can be started automatically during system startup, and even restarted automatically by Windows in the event of a service failure.

The MySQL Server Instance Configuration Wizard installs the MySQL server as a service by default, using the service name `MySQL`. If you do not wish to install the service, uncheck the box next to the `Install As Windows Service` option. You can change the service name by picking a new service name from the drop-down box provided or by entering a new service name into the drop-down box.



Note

Service names can include any legal character except forward (/) or backward (\) slashes, and must be less than 256 characters long.

**Warning**

If you are installing multiple versions of MySQL onto the same machine, you *must* choose a different service name for each version that you install. If you do not choose a different service for each installed version then the service manager information will be inconsistent and this will cause problems when you try to uninstall a previous version.

If you have already installed multiple versions using the same service name, you must manually edit the contents of the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services` parameters within the Windows registry to update the association of the service name with the correct server version.

Typically, when installing multiple versions you create a service name based on the version information. For example, you might install MySQL 5.x as `mysql5`, or specific versions such as MySQL 5.5.0 as `mysql50500`.

To install the MySQL server as a service but not have it started automatically at startup, uncheck the box next to the Launch the MySQL Server Automatically option.

2.3.6.11. The Security Options Dialog

The content of the security options portion of the MySQL Server Instance Configuration Wizard will depend on whether this is a new installation, or modifying an existing installation.

- **Setting the root password for a new installation**

It is strongly recommended that you set a `root` password for your MySQL server, and the MySQL Server Instance Config Wizard requires by default that you do so. If you do not wish to set a `root` password, uncheck the box next to the Modify Security Settings option.

MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please set the security options.

Modify Security Settings

 New root password: Enter the root password.

 Confirm: Retype the password.

Enable root access from remote machines

Create An Anonymous Account

 This option will create an anonymous account on this server. Please note that this can lead to an insecure system.

< Back Next > Cancel

- To set the `root` password, enter the desired password into both the New root password and Confirm boxes.

Setting the root password for an existing installation

If you are modifying the configuration of an existing configuration, or you are installing an upgrade and the MySQL Server Instance Configuration Wizard has detected an existing MySQL system, then you must enter the existing password for `root` before changing the configuration information.

MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration
Configure the MySQL Server 5.1 server instance.

Please set the security options.

Modify Security Settings

 Current root password: Enter the current password.

New root password: Enter the root password.

Confirm: Retype the password.

Enable root access from remote machines

Create An Anonymous Account

 This option will create an anonymous account on this server. Please note that this can lead to an insecure system.

< Back Next > Cancel

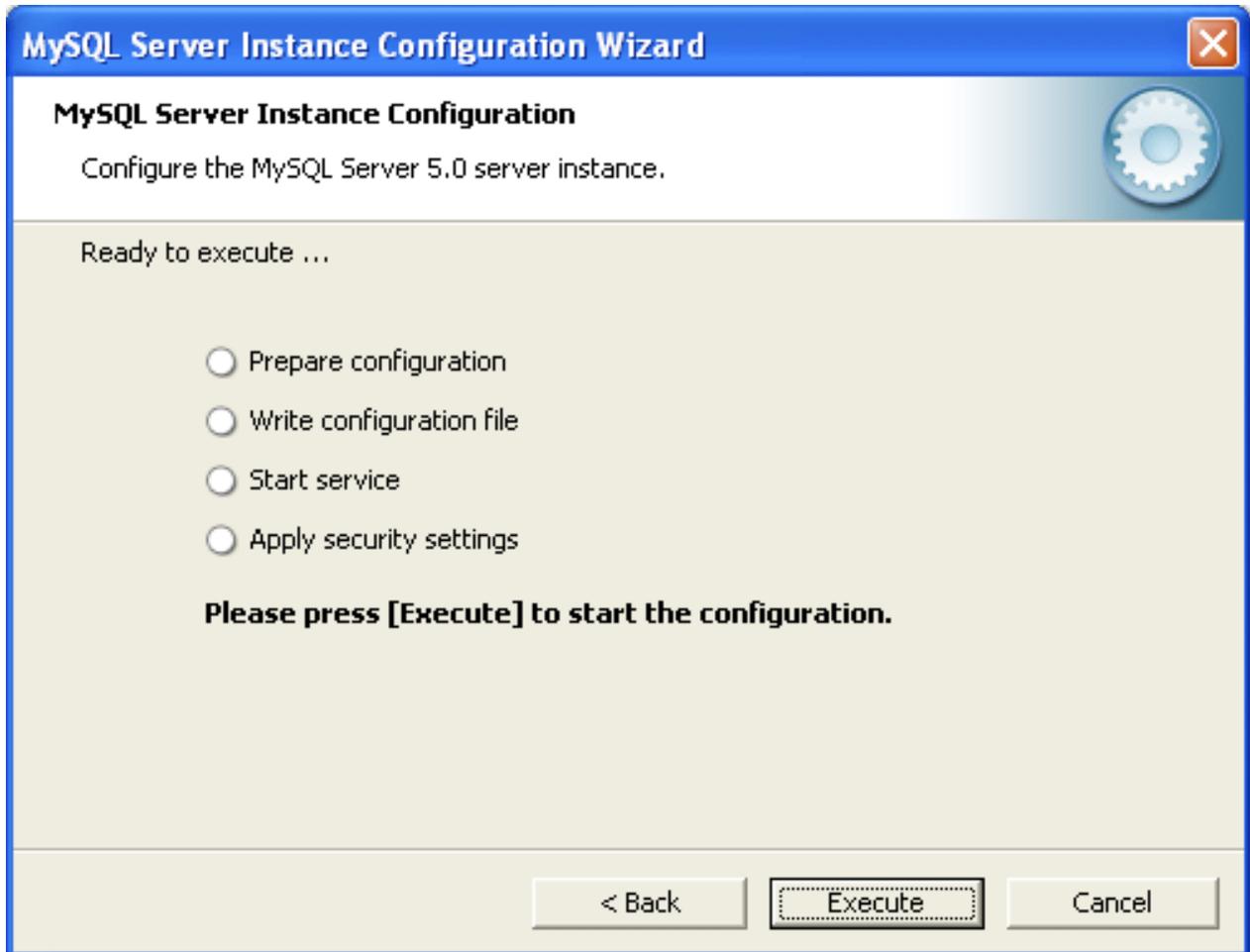
If you want to change the current `root` password, enter the desired new password into both the New root password and Confirm boxes.

To permit `root` logins from across the network, check the box next to the Enable root access from remote machines option. This decreases the security of your `root` account.

To create an anonymous user account, check the box next to the Create An Anonymous Account option. Creating an anonymous account can decrease server security and cause login and permission difficulties. For this reason, it is not recommended.

2.3.6.12. The Confirmation Dialog

The final dialog in the MySQL Server Instance Configuration Wizard is the Confirmation Dialog. To start the configuration process, click the Execute button. To return to a previous dialog, click the Back button. To exit the MySQL Server Instance Configuration Wizard without configuring the server, click the Cancel button.



After you click the **Execute** button, the MySQL Server Instance Configuration Wizard performs a series of tasks and displays the progress onscreen as the tasks are performed.

The MySQL Server Instance Configuration Wizard first determines configuration file options based on your choices using a template prepared by MySQL developers and engineers. This template is named `my-template.ini` and is located in your server installation directory.

The MySQL Configuration Wizard then writes these options to the corresponding configuration file.

If you chose to create a service for the MySQL server, the MySQL Server Instance Configuration Wizard creates and starts the service. If you are reconfiguring an existing service, the MySQL Server Instance Configuration Wizard restarts the service to apply your configuration changes.

If you chose to set a `root` password, the MySQL Configuration Wizard connects to the server, sets your new `root` password, and applies any other security settings you may have selected.

After the MySQL Server Instance Configuration Wizard has completed its tasks, it displays a summary. Click the **Finish** button to exit the MySQL Server Configuration Wizard.

2.3.6.13. MySQL Server Instance Config Wizard: Creating an Instance from the Command Line

In addition to using the GUI interface to the MySQL Server Instance Config Wizard, you can also create instances automatically from the command line.

To use the MySQL Server Instance Config Wizard on the command line, you need to use the `MySQLInstanceConfig.exe` command that is installed with MySQL in the `bin` directory within the installation directory. `MySQLInstanceConfig.exe` takes a number of command-line arguments that set the properties that would normally be selected through the GUI interface, and then creates a new configuration file (`my.ini`) by combining these selections with a template configuration file to produce the working configuration file.

The main command line options are provided in the table below. Some of the options are required, while some options are optional.

Table 2.4. MySQL Server Instance Config Wizard Command Line Options

Option	Description
Required Parameters	
<code>-nPRODUCTNAME</code>	The name of the instance when installed
<code>-pPATH</code>	Path of the base directory for installation. This is equivalent to the directory when using the <code>basedir</code> configuration parameter
<code>-vVERSION</code>	The version tag to use for this installation
Action to Perform	
<code>-i</code>	Install an instance
<code>-r</code>	Remove an instance
<code>-s</code>	Stop an existing instance
<code>-q</code>	Perform the operation quietly
<code>-lFILENAME</code>	Save the installation progress in a logfile
Config File to Use	
<code>-tFILENAME</code>	Path to the template config file that will be used to generate the installed configuration file
<code>-cFILENAME</code>	Path to a config file to be generated

The `-t` and `-c` options work together to set the configuration parameters for a new instance. The `-t` option specifies the template configuration file to use as the basic configuration, which are then merged with the configuration parameters generated by the MySQL Server Instance Config Wizard into the configuration file specified by the `-c` option.

A sample template file, `my-template.ini` is provided in the toplevel MySQL installation directory. The file contains elements that are replaced automatically by the MySQL Server Instance Config Wizard during configuration.

If you specify a configuration file that already exists, the existing configuration file will be saved in the file with the original, with the date and time added. For example, the `mysql.ini` will be copied to `mysql 2009-10-27 1646.ini.bak`.

The parameters that you can specify on the command line are listed in the table below.

Table 2.5. MySQL Server Instance Config Wizard Parameters

Parameter	Description
<code>ServiceName=\$</code>	Specify the name of the service to be created
<code>AddBinToPath={yes no}</code>	Specifies whether to add the binary directory of MySQL to the standard <code>PATH</code> environment variable

Parameter	Description
<code>ServerType={DEVELOPMENT SERVER DEDICATED}</code>	Specify the server type. For more information, see Section 2.3.6.4, “The Server Type Dialog”
<code>DatabaseType={MIXED INNODB MYISAM}</code>	Specify the default database type. For more information, see Section 2.3.6.5, “The Database Usage Dialog”
<code>ConnectionUsage={DSS OLTP}</code>	Specify the type of connection support, this automates the setting for the number of concurrent connections (see the <code>ConnectionCount</code> parameter). For more information, see Section 2.3.6.7, “The Concurrent Connections Dialog”
<code>ConnectionCount=#</code>	Specify the number of concurrent connections to support. For more information, see Section 2.3.6.4, “The Server Type Dialog”
<code>SkipNetworking={yes no}</code>	Specify whether network support should be supported. Specifying <code>yes</code> disables network access altogether
<code>Port=#</code>	Specify the network port number to use for network connections. For more information, see Section 2.3.6.8, “The Networking and Strict Mode Options Dialog”
<code>StrictMode={yes no}</code>	Specify whether to use the <code>strict</code> SQL mode. For more information, see Section 2.3.6.8, “The Networking and Strict Mode Options Dialog”
<code>Charset=\$</code>	Specify the default character set. For more information, see Section 2.3.6.9, “The Character Set Dialog”
<code>RootPassword=\$</code>	Specify the root password
<code>RootCurrentPassword=\$</code>	Specify the current root password then stopping or reconfiguring an existing service



Note

When specifying options on the command line, you can enclose the entire command-line option and the value you are specifying using double quotation marks. This enables you to use spaces in the options. For example, `"-cC:\mysql.ini"`.

The following command installs a MySQL Server 5.5 instance from the directory `C:\Program Files\MySQL\MySQL Server 5.5` using the service name `MySQL55` and setting the root password to `1234`.

```
shell> MySQLInstanceConfig.exe -i -q "-lC:\mysql_install_log.txt" »
"-nMySQL Server 5.5" "-pC:\Program Files\MySQL\MySQL Server 5.5" -v5.5.31 »
"-tmy-template.ini" "-cC:\mytest.ini" ServerType=DEVELOPMENT DatabaseType=MIXED »
ConnectionUsage=DSS Port=3311 ServiceName=MySQL55 RootPassword=1234
```

In the above example, a log file will be generated in `mysql_install_log.txt` containing the information about the instance creation process. The log file generated by the above example is shown below:

```
Welcome to the MySQL Server Instance Configuration Wizard 1.0.16.0
Date: 2009-10-27 17:07:21

Installing service ...

Product Name:      MySQL Server 5.5
Version:           5.5.31
Installation Path: C:\Program Files\MySQL\MySQL Server 5.5\

Creating configuration file C:\mytest.ini using template my-template.ini.
Options:
```

```

DEVELOPMENT
MIXED
DSS
STRICTMODE

Variables:
port: 3311
default-character-set: latin1
basedir: "C:/Program Files/MySQL/MySQL Server 5.5/"
datadir: "C:/Program Files/MySQL/MySQL Server 5.5/Data/"

Creating Windows service entry.
Service name: "MySQL55"
Parameters: "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --defaults-file="C:\mytest.ini" MySQL55
Windows service MySQL55 installed.

```

When using the command line, the return values in the following table indicate an error performing the specified option.

Table 2.6. Return Value from MySQL Server Instance Config Wizard

Value	Description
2	Configuration template file cannot be found
3	The Windows service entry cannot be created
4	Could not connect to the Service Control Manager
5	The MySQL service cannot be started
6	The MySQL service cannot be stopped
7	The security settings cannot be applied
8	The configuration file cannot be written
9	The Windows service entry cannot be removed

You can perform an installation of MySQL automatically using the MSI package. For more information, see [Section 2.3.4.2, “Automating MySQL Installation on Microsoft Windows using the MSI Package”](#).

2.3.7. Installing MySQL on Microsoft Windows Using a `noinstall` Zip Archive

Users who are installing from the Noinstall package can use the instructions in this section to manually install MySQL. The process for installing MySQL from a Zip archive is as follows:

1. Extract the archive to the desired install directory
2. Create an option file
3. Choose a MySQL server type
4. Start the MySQL server
5. Secure the default user accounts

This process is described in the sections that follow.

2.3.7.1. Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to [Section 2.3.9, “Upgrading MySQL on Windows”](#), before beginning the upgrade process.

2. Make sure that you are logged in as a user with administrator privileges.
3. Choose an installation location. Traditionally, the MySQL server is installed in `C:\mysql`. The MySQL Installation Wizard installs MySQL under `C:\Program Files\MySQL`. If you do not install MySQL at `C:\mysql`, you must specify the path to the install directory during startup or in an option file. See [Section 2.3.7.2, “Creating an Option File”](#).
4. Extract the install archive to the chosen installation location using your preferred Zip archive tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs, you can move the contents of the subfolder into the chosen installation location.

2.3.7.2. Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations are different from the default locations (`C:\Program Files\MySQL\MySQL Server 5.5` and `C:\Program Files\MySQL\MySQL Server 5.5\data`).
- You need to tune the server settings, such as memory, cache, or InnoDB configuration information.

When the MySQL server starts on Windows, it looks for option files in several locations, such as the Windows directory, `C:\`, and the MySQL installation directory (for the full list of locations, see [Section 4.2.3.3, “Using Option Files”](#)). The Windows directory typically is named something like `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

MySQL looks for options in each location first in the `my.ini` file, and then in the `my.cnf` file. However, to avoid confusion, it is best if you use only one file. If your PC uses a boot loader where `C:` is not the boot drive, your only option is to use the `my.ini` file. Whichever option file you use, it must be a plain text file.



Note

When using the MySQL Installer to install MySQL Server, it will create the `my.ini` at the default location. And as of MySQL Server 5.5.27, the user running MySQL Installer is granted full permissions to this new `my.ini`.

In other words, be sure that the MySQL Server user has permission to read the `my.ini` file.

You can also make use of the example option files included with your MySQL distribution; see [Section 5.1.2, “Server Configuration Defaults”](#).

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is in `E:\mydata\data`, you can create an option file containing a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Note that Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

The rules for use of backslash in option file values are given in [Section 4.2.3.3, “Using Option Files”](#).

The data directory is located within the `AppData` directory for the user running MySQL.

If you would like to use a data directory in a different location, you should copy the entire contents of the `data` directory to the new location. For example, if you want to use `E:\\mydata` as the data directory instead, you must do two things:

1. Move the entire `data` directory and all of its contents from the default location (for example `C:\\Program Files\\MySQL\\MySQL Server 5.5\\data`) to `E:\\mydata`.
2. Use a `--datadir` [439] option to specify the new data directory location each time you start the server.

2.3.7.3. Selecting a MySQL Server Type

The following table shows the available servers for Windows in MySQL 5.5.

Binary	Description
<code>mysqld</code>	Optimized binary with named-pipe support
<code>mysqld-debug</code>	Like <code>mysqld</code> , but compiled with full debugging and automatic memory allocation checking

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

Each of the servers in a distribution support the same set of storage engines. The `SHOW ENGINES` statement displays which engines a given server supports.

All Windows MySQL 5.5 servers have support for symbolic linking of database directories.

MySQL supports TCP/IP on all Windows platforms. MySQL servers on Windows support named pipes as indicated in the following list. However, the default is to use TCP/IP regardless of platform. (Named pipes are slower than TCP/IP in many Windows configurations.)

Named pipes are enabled only if you start the server with the `--enable-named-pipe` [442] option. It is necessary to use this option explicitly because some users have experienced problems with shutting down the MySQL server when named pipes were used.

2.3.7.4. Starting the Server for the First Time

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The information here applies primarily if you installed MySQL using the `noinstall` version, or if you wish to configure and test MySQL manually rather than with the GUI tools.

The examples in these sections assume that MySQL is installed under the default location of `C:\\Program Files\\MySQL\\MySQL Server 5.5`. Adjust the path names shown in the examples if you have MySQL installed in a different location.

Clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named-pipe connections.

MySQL for Windows also supports shared-memory connections if the server is started with the `--shared-memory` [460] option. Clients can connect through shared memory by using the `--protocol=MEMORY` [239] option.

For information about which server binary to run, see [Section 2.3.7.3, “Selecting a MySQL Server Type”](#).

Testing is best done from a command prompt in a console window (or “DOS window”). In this way you can have the server display status messages in the window where they are easy to see. If something is wrong with your configuration, these messages make it easier for you to identify and fix any problems.

To start the server, enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --console
```

For a server that includes `InnoDB` support, you should see the messages similar to those following as it starts (the path names and sizes may differ):

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25 InnoDB: Started
```

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
Version: '5.5.31' socket: '' port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the `--console` [438] option, the server writes diagnostic output to the error log in the data directory (`C:\Program Files\MySQL\MySQL Server 5.5\data` by default). The error log is the file with the `.err` extension, or may be specified by passing in the `--log-error` [448] option.



Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.11, “Postinstallation Setup and Testing”](#).

2.3.7.5. Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

To start the `mysqld` server from the command line, you should start a console window (or “DOS window”) and enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld"
```

The path to `mysqld` may vary depending on the install location of MySQL on your system.

You can stop the MySQL server by executing this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqladmin" -u root shutdown
```



Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

If `mysqld` doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. By default, the error log is located in the `C:\Program Files\MySQL\MySQL Server 5.5\data` directory. It is the file with a suffix of `.err`, or may be specified by passing in the `--log-error [448]` option. Alternatively, you can try to start the server as `mysqld --console`; in this case, you may get some useful information on the screen that may help solve the problem.

The last option is to start `mysqld` with the `--standalone [462]` and `--debug [439]` options. In this case, `mysqld` writes a log file `C:\mysqld.trace` that should contain the reason why `mysqld` doesn't start. See [MySQL Internals: Porting to Other Systems](#).

Use `mysqld --verbose --help` to display all the options that `mysqld` supports.

2.3.7.6. Customizing the PATH for MySQL Tools

To make it easier to invoke MySQL programs, you can add the path name of the MySQL `bin` directory to your Windows system `PATH` environment variable:

- On the Windows desktop, right-click the My Computer icon, and select Properties.
- Next select the Advanced tab from the [System Properties](#) menu that appears, and click the [Environment Variables](#) button.
- Under **System Variables**, select Path, and then click the [Edit](#) button. The [Edit System Variable](#) dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL `bin` directory (for example, `C:\Program Files\MySQL\MySQL Server 5.5\bin`)



Note

There must be a semicolon separating this path from any values present in this field.

Dismiss this dialogue, and each dialogue in turn, by clicking **OK** until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by

typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the `mysql` client, and all MySQL command-line utilities such as `mysqladmin` and `mysqldump`.

You should not add the MySQL `bin` directory to your Windows `PATH` if you are running multiple MySQL servers on the same machine.



Warning

You must exercise great care when editing your system `PATH` by hand; accidental deletion or modification of any portion of the existing `PATH` value can leave you with a malfunctioning or even unusable system.

2.3.7.7. Starting MySQL as a Windows Service

On Windows, the recommended way to run MySQL is to install it as a Windows service, whereby MySQL starts and stops automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using `NET` commands, or with the graphical `Services` utility. Generally, to install MySQL as a Windows service you should be logged in using an account that has administrator rights.

The `Services` utility (the Windows `Service Control Manager`) can be found in the Windows Control Panel (under Administrative Tools on Windows 2000, XP, Vista, and Server 2003). To avoid conflicts, it is advisable to close the `Services` utility while performing server installation or removal operations from the command line.

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqladmin"  
-u root shutdown
```



Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

Install the server as a service using this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

To make it easier to invoke MySQL programs, you can add the path name of the MySQL `bin` directory to your Windows system `PATH` environment variable:

- On the Windows desktop, right-click the My Computer icon, and select Properties.
- Next select the Advanced tab from the `System Properties` menu that appears, and click the `Environment Variables` button.

- Under **System Variables**, select Path, and then click the **Edit** button. The **Edit System Variable** dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL `bin` directory (for example, `C:\Program Files\MySQL\MySQL Server 5.5\bin`), Note that there should be a semicolon separating this path from any values present in this field. Dismiss this dialogue, and each dialogue in turn, by clicking **OK** until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the `mysql` client, and all MySQL command-line utilities such as `mysqladmin` and `mysqldump`.

You should not add the MySQL `bin` directory to your Windows `PATH` if you are running multiple MySQL servers on the same machine.



Warning

You must exercise great care when editing your system `PATH` by hand; accidental deletion or modification of any portion of the existing `PATH` value can leave you with a malfunctioning or even unusable system.

The following additional arguments can be used when installing the service:

- You can specify a service name immediately following the `--install` option. The default service name is `MySQL`.
- If a service name is given, it can be followed by a single option. By convention, this should be `--defaults-file=file_name [248]` to specify the name of an option file from which the server should read options when it starts.

The use of a single option other than `--defaults-file [248]` is possible but discouraged. `--defaults-file [248]` is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file.

- You can also specify a `--local-service` option following the service name. This causes the server to run using the `LocalService` Windows account that has limited system privileges. This account is available only for Windows XP or newer. If both `--defaults-file [248]` and `--local-service` are given following the service name, they can be in any order.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (`MySQL`) following the `--install` option, the server uses the a service name of `MySQL` and reads options from the `[mysqld]` group in the standard option files.
- If the service-installation command specifies a service name other than `MySQL` following the `--install` option, the server uses that service name. It reads options from the `[mysqld]` group and the group that has the same name as the service in the standard option files. This enables you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group with the service name for use by the server installed with that service name.
- If the service-installation command specifies a `--defaults-file [248]` option after the service name, the server reads options the same way as described in the previous item, except that it reads options only from the the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld"
      --install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (`MySQL`) is given after the `--install` option. If no `--defaults-file` [248] option had been given, this command would have the effect of causing the server to read the `[mysqld]` group from the standard option files. However, because the `--defaults-file` [248] option is present, the server reads options from the `[mysqld]` option group, and only from the named file.

You can also specify options as Start parameters in the Windows `Services` utility before you start the MySQL service.

Once a MySQL server has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from the `Services` utility, or by using a `NET START MySQL` command. The `NET` command is not case sensitive.

When run as a service, `mysqld` has no access to a console window, so no messages can be seen there. If `mysqld` does not start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the MySQL data directory (for example, `C:\Program Files\MySQL\MySQL Server 5.5\data`). It is the file with a suffix of `.err`.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually by using the `Services` utility, the `NET STOP MySQL` command, or the `mysqladmin shutdown` command.

You also have the choice of installing the server as a manual service if you do not wish for the service to be started automatically during the boot process. To do this, use the `--install-manual` option rather than the `--install` option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --install-manual
```

To remove a server that is installed as a service, first stop it if it is running by executing `NET STOP MySQL`. Then use the `--remove` [458] option to remove it:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --remove
```

If `mysqld` is not running as a service, you can start it from the command line. For instructions, see [Section 2.3.7.5, "Starting MySQL from the Windows Command Line"](#).

Please see [Section 2.3.8, "Troubleshooting a Microsoft Windows MySQL Server Installation"](#), if you encounter difficulties during installation.

2.3.7.8. Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqlshow"
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqlshow" -u root mysql
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqladmin" version status proc
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysql" test
```

If `mysqld` is slow to respond to TCP/IP connections from client programs, there is probably a problem with your DNS. In this case, start `mysqld` with the `--skip-name-resolve` [461] option and use only `localhost` and IP addresses in the `Host` column of the MySQL grant tables.

You can force a MySQL client to use a named-pipe connection rather than TCP/IP by specifying the `--pipe` [239] or `--protocol=PIPE` [239] option, or by specifying `.` (period) as the host name. Use the `--socket` [240] option to specify the name of the pipe if you do not want to use the default pipe name.

Note that if you have set a password for the `root` account, deleted the anonymous account, or created a new user account, then you must use the appropriate `-u` and `-p` options with the commands shown above to connect with the MySQL Server. See [Section 4.2.2, “Connecting to the MySQL Server”](#).

For more information about `mysqlshow`, see [Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”](#).

2.3.8. Troubleshooting a Microsoft Windows MySQL Server Installation

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. The purpose of this section is to help you diagnose and correct some of these errors.

Your first resource when troubleshooting server issues is the [error log](#). The MySQL server uses the error log to record information relevant to the error that prevents the server from starting. The error log is located in the [data directory](#) specified in your `my.ini` file. The default data directory location is `C:\Program Files\MySQL\MySQL Server 5.5\data`, or `C:\ProgramData\Mysql` on Windows 7 and Windows Server 2008. The `C:\ProgramData` directory is hidden by default. You need to change your folder options to see the directory and contents. For more information on the error log and understanding the content, see [Section 5.2.2, “The Error Log”](#).

Another source of information regarding possible errors is the console messages displayed when the MySQL service is starting. Use the `NET START MySQL` command from the command line after installing `mysqld` as a service to see any error messages regarding the starting of the MySQL server as a service. See [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

The following examples show other common error messages you may encounter when installing MySQL and starting the server for the first time:

- If the MySQL server cannot find the `mysql` privileges database or other critical files, you may see these messages:

```
System error 1067 has occurred.
Fatal error: Can't open privilege tables: Table 'mysql.host' doesn't exist
```

These messages often occur when the MySQL base or data directories are installed in different locations than the default locations (`C:\Program Files\MySQL\MySQL Server 5.5` and `C:\Program Files\MySQL\MySQL Server 5.5\data`, respectively).

This situation may occur when MySQL is upgraded and installed to a new location, but the configuration file is not updated to reflect the new location. In addition, there may be old and new configuration files that conflict. Be sure to delete or rename any old configuration files when upgrading MySQL.

If you have installed MySQL to a directory other than `C:\Program Files\MySQL\MySQL Server 5.5`, you need to ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. The `my.ini` file needs to be located in your Windows directory, typically `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable by issuing the following command from the command prompt:

```
C:\> echo %WINDIR%
```

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is `D:\MySQLdata`, you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
```

```
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

Note that Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\MySQL\\MySQL Server 5.5
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

The rules for use of backslash in option file values are given in [Section 4.2.3.3, “Using Option Files”](#).

If you change the `datadir` value in your MySQL configuration file, you must move the contents of the existing MySQL data directory before restarting the MySQL server.

See [Section 2.3.7.2, “Creating an Option File”](#).

- If you reinstall or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Configuration Wizard, you may see this error:

```
Error: Cannot create Windows service for MySql. Error: 0
```

This occurs when the Configuration Wizard tries to install the service and finds an existing service with the same name.

One solution to this problem is to choose a service name other than `mysql` when using the configuration wizard. This enables the new service to be installed correctly, but leaves the outdated service in place. Although this is harmless, it is best to remove old services that are no longer in use.

To permanently remove the old `mysql` service, execute the following command as a user with administrative privileges, on the command-line:

```
C:\> sc delete mysql
[SC] DeleteService SUCCESS
```

If the `sc` utility is not available for your version of Windows, download the `delsrv` utility from <http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp> and use the `delsrv mysql` syntax.

2.3.9. Upgrading MySQL on Windows

This section lists some of the steps you should take when upgrading MySQL on Windows.

1. Review [Section 2.12.1, “Upgrading MySQL”](#), for additional information on upgrading MySQL that is not specific to Windows.
2. You should always back up your current MySQL installation before performing an upgrade. See [Section 7.2, “Database Backup Methods”](#).
3. Download the latest Windows distribution of MySQL from <http://dev.mysql.com/downloads/>.
4. Before upgrading MySQL, you must stop the server. If the server is installed as a service, stop the service with the following command from the command prompt:

```
C:\> NET STOP MySQL
```

If you are not running the MySQL server as a service, use `mysqladmin` to stop it. For example, before upgrading from MySQL 5.1 to 5.5, use `mysqladmin` from MySQL 5.1 as follows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" -u root shutdown
```

**Note**

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

5. Before upgrading a MySQL service from MySQL 5.1 to 5.5, you should stop the 5.1 server and remove the instance. Run the MySQL Instance Configuration Wizard, choose the Remove Instance option and in the next screen, confirm removal. After that it is safe to uninstall MySQL Server 5.1.
6. Before upgrading to MySQL 5.5 from a version previous to 4.1.5, or from a version of MySQL installed from a Zip archive to a version of MySQL installed with the MySQL Installation Wizard, you must first manually remove the previous installation and MySQL service (if the server is installed as a service).

To remove the MySQL service, use the following command:

```
C:\> C:\mysql\bin\mysqld --remove
```

If you do not remove the existing service, the MySQL Installation Wizard may fail to properly install the new MySQL service.

7. If you are using the MySQL Installation Wizard, start the wizard as described in [Section 2.3.4.1, “Using the MySQL Installation Wizard”](#).
8. If you are installing MySQL from a Zip archive, extract the archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql5`. Overwriting the existing installation is recommended.
9. If you were running MySQL as a Windows service and you had to remove the service earlier in this procedure, reinstall the service. (See [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).)
10. Restart the server. For example, use `NET START MySQL` if you run MySQL as a service, or invoke `mysqld` directly otherwise.
11. As Administrator, run `mysql_upgrade` to check your tables, attempt to repair them if necessary, and update your grant tables if they have changed so that you can take advantage of any new capabilities. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).
12. If you encounter errors, see [Section 2.3.8, “Troubleshooting a Microsoft Windows MySQL Server Installation”](#).

2.3.10. Windows Postinstallation Procedures

On Windows, you need not create the data directory and the grant tables. MySQL Windows distributions include the grant tables with a set of preinitialized accounts in the `mysql` database under the data directory. Regarding passwords, if you installed MySQL using the Windows Installation Wizard, you may have already assigned passwords to the accounts. (See [Section 2.3.4.1, “Using the MySQL Installation Wizard”](#).) Otherwise, use the password-assignment procedure given in [Section 2.11.2, “Securing the Initial MySQL Accounts”](#).

Before setting up passwords, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure that the server is running (see [Section 2.3.7.4, “Starting the Server for the First Time”](#)), and then issue the following commands to verify that you can retrieve information from the server. You may need to specify directory different from `C:\mysql\bin` on the command line. If you used the Windows Installation Wizard, the default directory is

C:\Program Files\MySQL\MySQL Server 5.5, and the `mysql` and `mysqlshow` client programs are in C:\Program Files\MySQL\MySQL Server 5.5\bin. See [Section 2.3.4.1, “Using the MySQL Installation Wizard”](#), for more information.

Use `mysqlshow` to see what databases exist:

```
C:\> C:\mysql\bin\mysqlshow
+-----+
|   Databases   |
+-----+
| information_schema |
| mysql          |
| test          |
+-----+
```

The list of installed databases may vary, but will always include the minimum of `mysql` and `information_schema`. In most cases, the `test` database will also be installed automatically.

The preceding command (and commands for other MySQL programs such as `mysql`) may not work if the correct MySQL account does not exist. For example, the program may fail with an error, or you may not be able to view all databases. If you installed using the MSI packages and used the MySQL Server Instance Config Wizard, then the `root` user will have been created automatically with the password you supplied. In this case, you should use the `-u root` and `-p` options. (You will also need to use the `-u root` and `-p` options if you have already secured the initial MySQL accounts.) With `-p`, you will be prompted for the `root` password. For example:

```
C:\> C:\mysql\bin\mysqlshow -u root -p
Enter password: (enter root password here)
+-----+
|   Databases   |
+-----+
| information_schema |
| mysql          |
| test          |
+-----+
```

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
C:\> C:\mysql\bin\mysqlshow mysql
Database: mysql
+-----+
|   Tables      |
+-----+
| columns_priv  |
| db            |
| event         |
| func          |
| help_category |
| help_keyword  |
| help_relation |
| help_topic    |
| host         |
| plugin        |
| proc          |
| procs_priv    |
| servers       |
| tables_priv   |
| time_zone     |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user         |
+-----+
```

Use the `mysql` program to select information from a table in the `mysql` database:

```
C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM mysql.db"
+-----+-----+-----+
| host | db   | user |
+-----+-----+-----+
| %    | test|      |
| %    | test_%|      |
+-----+-----+-----+
```

For more information about `mysqlshow` and `mysql`, see [Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”](#), and [Section 4.5.1, “mysql — The MySQL Command-Line Tool”](#).

If you are running a version of Windows that supports services, you can set up the MySQL server to run automatically when Windows starts. See [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

2.4. Installing MySQL on Mac OS X

MySQL for Mac OS X is available in a number of different forms:

- Native Package Installer format, which uses the native Mac OS X installer to walk you through the installation of MySQL. For more information, see [Section 2.4.2, “Installing MySQL on Mac OS X Using Native Packages”](#). You can use the package installer with Mac OS X 10.3 and later, and the package is available for both PowerPC and Intel architectures, and 32-bit and 64-bit architectures. There is no Universal Binary available using the package installation method. The user you use to perform the installation must have administrator privileges.
- Tar package format, which uses a file packaged using the Unix `tar` and `gzip` commands. To use this method, you will need to open a `Terminal` window. You do not need administrator privileges using this method, as you can install the MySQL server anywhere using this method. For more information on using this method, you can use the generic instructions for using a tarball, [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#). You can use the package installer with Mac OS X 10.3 and later, and available for both PowerPC and Intel architectures, and both 32-bit and 64-bit architectures. A Universal Binary, incorporating both Power PC and Intel architectures and 32-bit and 64-bit binaries is available.

In addition to the core installation, the Package Installer also includes [Section 2.4.3, “Installing the MySQL Startup Item”](#) and [Section 2.4.4, “Installing and Using the MySQL Preference Pane”](#), both of which simplify the management of your installation.

- Mac OS X server includes a version of MySQL as standard. If you want to use a more recent version than that supplied with the Mac OS X server release, you can make use of the package or tar formats. For more information on using the MySQL bundled with Mac OS X, see [Section 2.4.5, “Using the Bundled MySQL on Mac OS X Server”](#).

For additional information on using MySQL on Mac OS X, see [Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”](#).

2.4.1. General Notes on Installing MySQL on Mac OS X

You should keep the following issues and notes in mind:

- The default location for the MySQL Unix socket is different on Mac OS X and Mac OS X Server depending on the installation type you chose. The following table shows the default locations by installation type.

Table 2.7. MySQL Unix Socket Locations on Mac OS X by Installation Type

Installation Type	Socket Location
Package Installer from MySQL	<code>/tmp/mysql.sock</code>
Tarball from MySQL	<code>/tmp/mysql.sock</code>
MySQL Bundled with Mac OS X Server	<code>/var/mysql/mysql.sock</code>

To prevent issues, you should either change the configuration of the socket used within your application (for example, changing `php.ini`), or you should configure the socket location using a MySQL configuration file and the `socket` [464] option. For more information, see [Section 5.1.3, “Server Command Options”](#).

- You may need (or want) to create a specific `mysql` user to own the MySQL directory and data. On Mac OS X 10.4 and lower you can do this by using the [Netinfo Manager](#) application, located within the [Utilities](#) folder within the [Applications](#) folder. On Mac OS X 10.5 and later you can do this through the [Directory Utility](#). From Mac OS X 10.5 and later (including Mac OS X Server 10.5) the `mysql` should already exist. For use in single user mode, an entry for `_mysql` (note the underscore prefix) should already exist within the system `/etc/passwd` file.
- Due to a bug in the Mac OS X package installer, you may see this error message in the destination disk selection dialog:

```
You cannot install this software on this disk. (null)
```

If this error occurs, click the [Go Back](#) button once to return to the previous screen. Then click [Continue](#) to advance to the destination disk selection again, and you should be able to choose the destination disk correctly. We have reported this bug to Apple and it is investigating this problem.

- Because the MySQL package installer installs the MySQL contents into a version and platform specific directory, you can use this to upgrade and migrate your database between versions. You will need to either copy the `data` directory from the old version to the new version, or alternatively specify an alternative `datadir` value to set location of the data directory.
- You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as `mysql` and `mysqladmin` from the command line. The syntax for `bash` is:

```
alias mysql=/usr/local/mysql/bin/mysql
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

For `tcsh`, use:

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. You can do this by modifying the appropriate startup file for your shell. For more information, see [Section 4.2.1, “Invoking MySQL Programs”](#).

- After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-VERSION.pkg`.

2.4.2. Installing MySQL on Mac OS X Using Native Packages

You can install MySQL on Mac OS X 10.3.x (“Panther”) or newer using a Mac OS X binary package in PKG format instead of the binary tarball distribution. Please note that older versions of Mac OS X (for example, 10.1.x or 10.2.x) are *not* supported by this package.

The package is located inside a disk image (`.dmg`) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.



Note

Before proceeding with the installation, be sure to stop all running MySQL server instances by using either the MySQL Manager Application (on Mac OS X Server) or `mysqladmin shutdown` on the command line.

When installing from the package version, you should also install the MySQL Preference Pane, which will enable you to control the startup and execution of your MySQL server from System Preferences. For more information, see [Section 2.4.4, “Installing and Using the MySQL Preference Pane”](#).

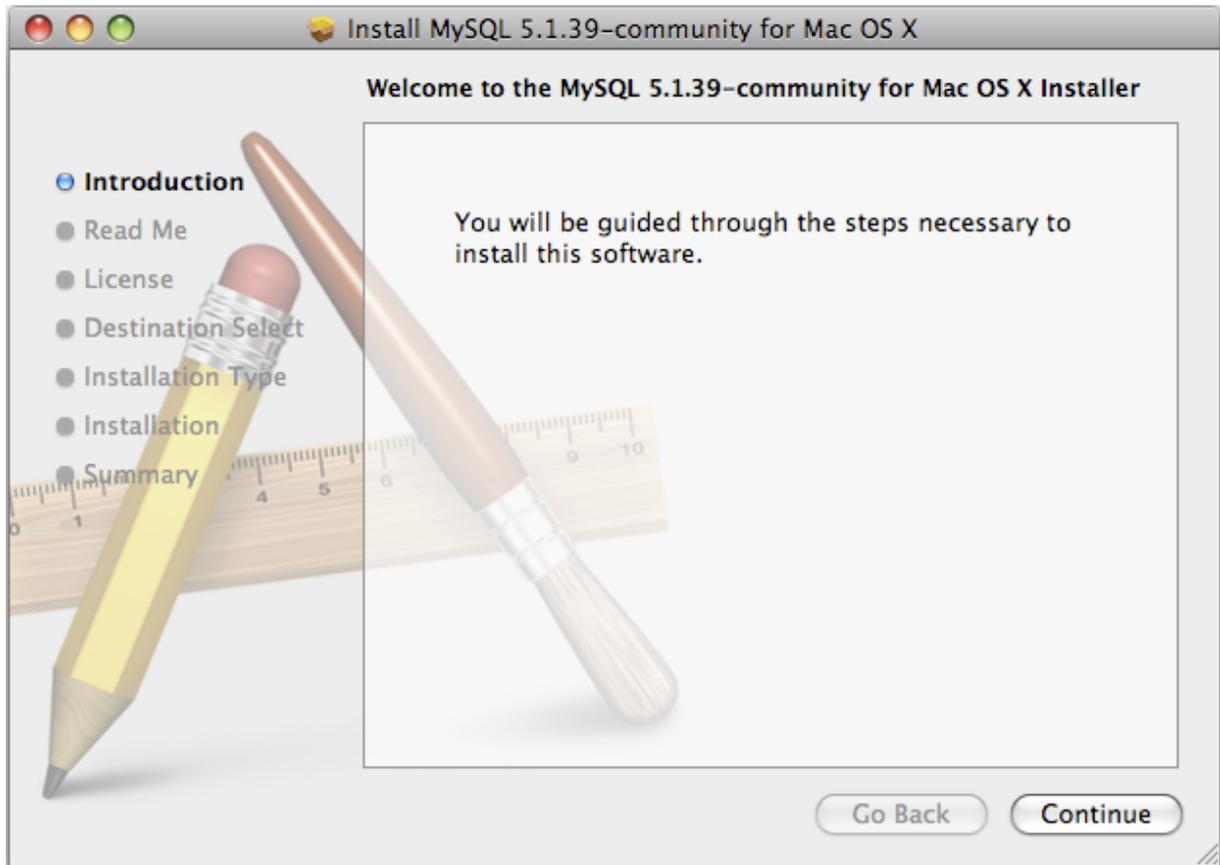
When installing using the package installer, the files are installed into a directory within `/usr/local` matching the name of the installation version and platform. For example, the installer file `mysql-5.1.39-osx10.5-x86_64.pkg` installs MySQL into `/usr/local/mysql-5.1.39-osx10.5-x86_64`. The following table shows the layout of the installation directory.

Table 2.8. MySQL Installation Layout on Mac OS X

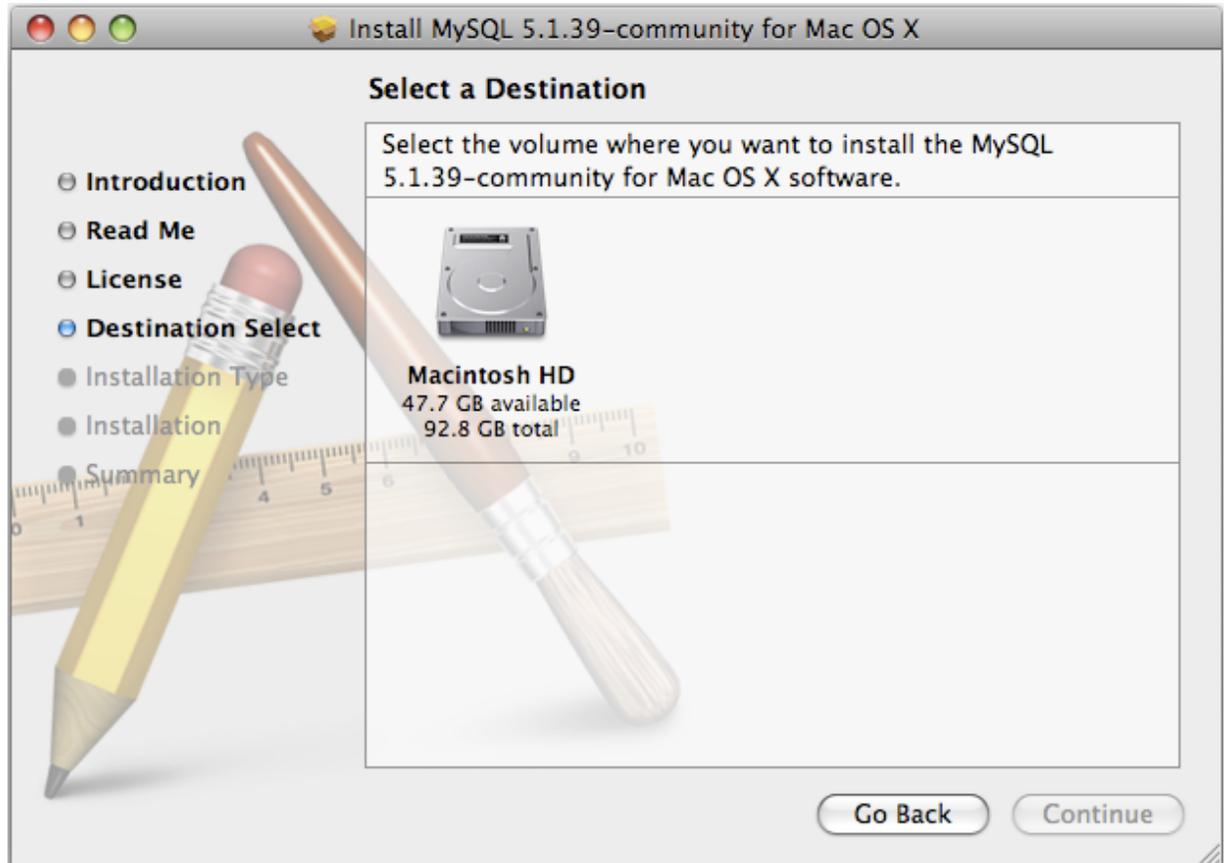
Directory	Contents of Directory
<code>bin</code>	Client programs and the <code>mysqld</code> server
<code>data</code>	Log files, databases
<code>docs</code>	Manual in Info format
<code>include</code>	Include (header) files
<code>lib</code>	Libraries
<code>man</code>	Unix manual pages
<code>mysql-test</code>	MySQL test suite
<code>scripts</code>	<code>mysql_install_db</code>
<code>share</code>	Miscellaneous support files, including error messages, sample configuration files, SQL for database installation
<code>sql-bench</code>	Benchmarks
<code>support-files</code>	Scripts and sample configuration files
<code>/tmp/mysql.sock</code>	Location of the MySQL Unix socket

During the package installer process, a symbolic link from `/usr/local/mysql` to the version/platform specific directory created during installation will be created automatically.

1. Download and open the MySQL package installer, which is provided on a disk image (`.dmg`) that includes the main MySQL installation package, the `MySQLStartupItem.pkg` installation package, and the `MySQL.prefPane`. Double-click the disk image to open it.
2. Double-click the MySQL installer package. It will be named according to the version of MySQL you have downloaded. For example, if you have downloaded MySQL 5.1.39, double-click `mysql-5.1.39-osx10.5-x86.pkg`.
3. You will be presented with the opening installer dialog. Click **Continue** to begin installation.



4. A copy of the installation instructions and other important information relevant to this installation are displayed. Click **Continue** .
5. If you have downloaded the community version of MySQL, you will be shown a copy of the relevant GNU General Public License. Click **Continue** .
6. Select the drive you want to use to install the MySQL Startup Item. The drive must have a valid, bootable, Mac OS X operating system installed. Click **Continue**.



7. You will be asked to confirm the details of the installation, including the space required for the installation. To change the drive on which the startup item is installed, click either **Go Back** or **Change Install Location...**. To install the startup item, click **Install**.
8. Once the installation has been completed successfully, you will be shown an **Install Succeeded** message.

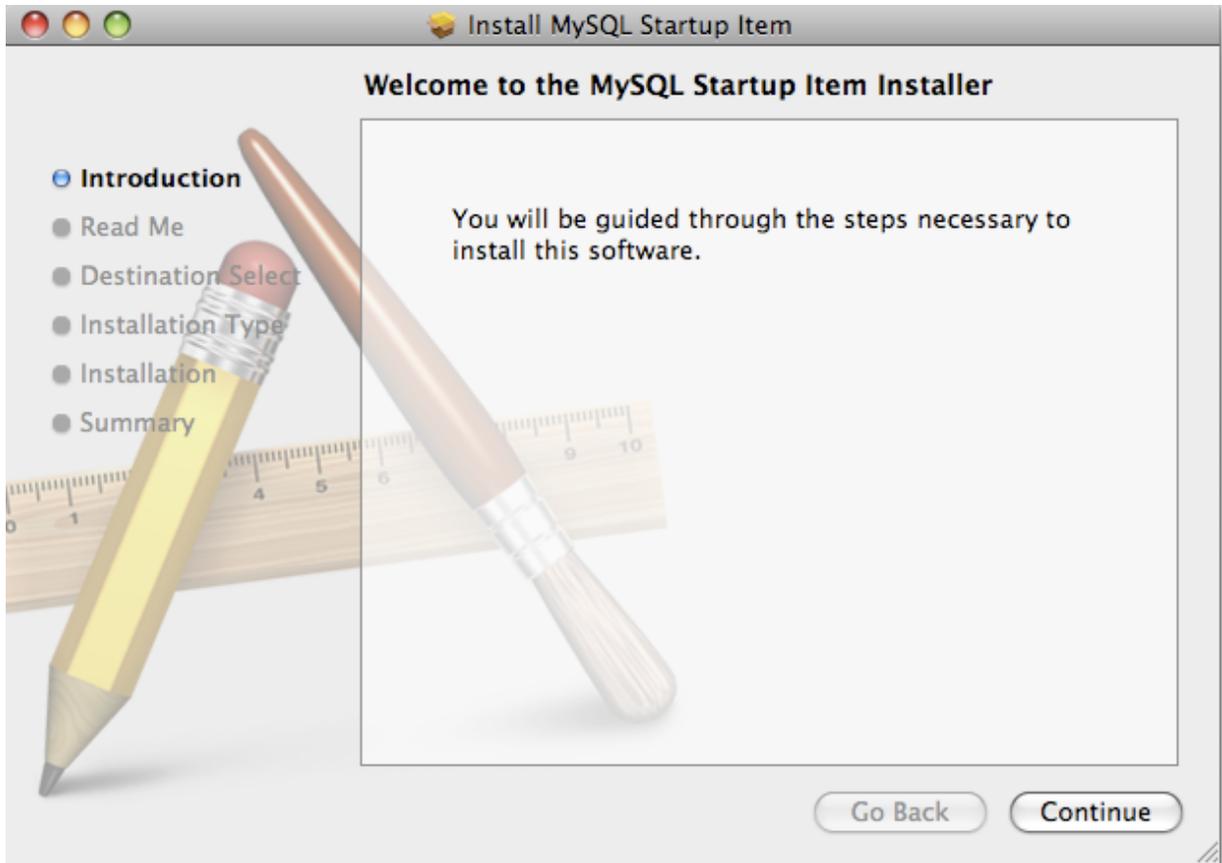
For convenience, you may also want to install the startup item and preference pane. See [Section 2.4.3, “Installing the MySQL Startup Item”](#), and [Section 2.4.4, “Installing and Using the MySQL Preference Pane”](#).

2.4.3. Installing the MySQL Startup Item

The MySQL Installation Package includes a startup item that can be used to automatically start and stop MySQL.

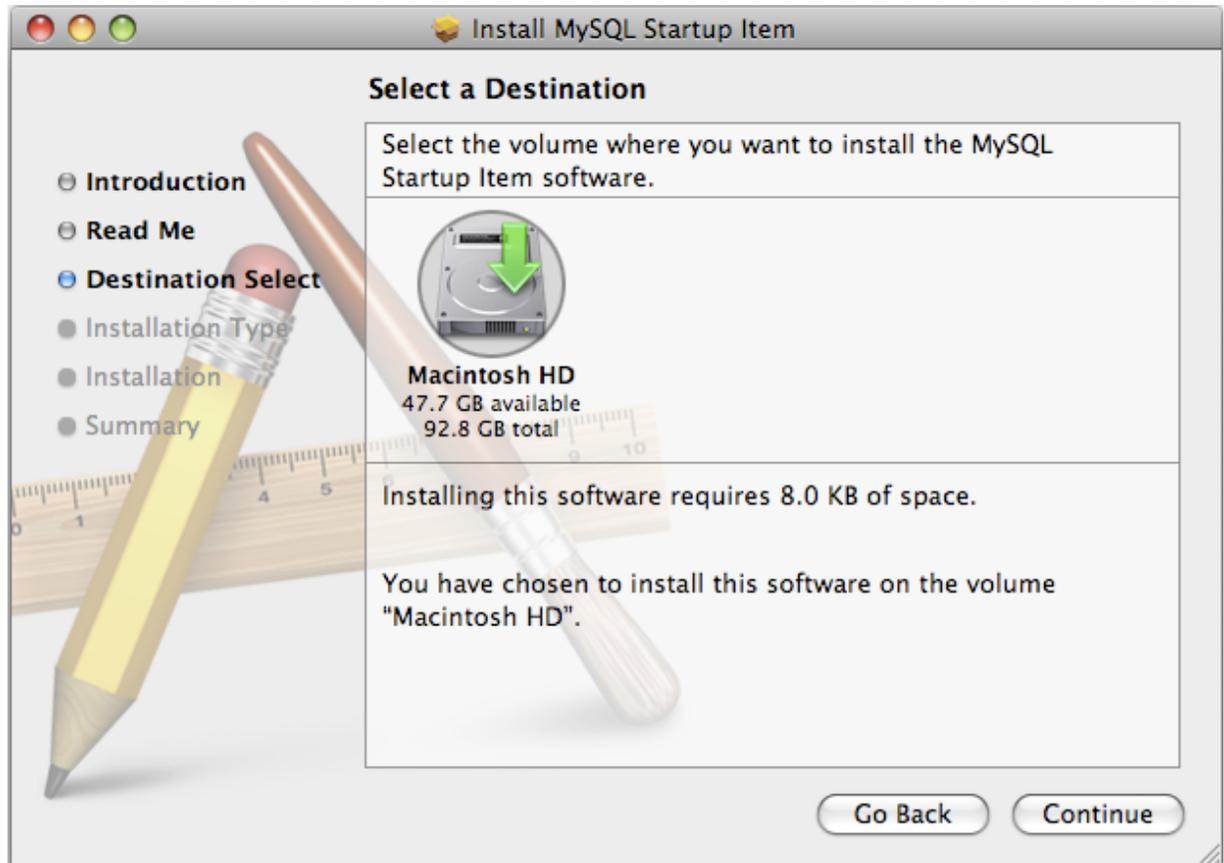
To install the MySQL Startup Item:

1. Download and open the MySQL package installer, which is provided on a disk image (`.dmg`) that includes the main MySQL installation package, the `MySQLStartupItem.pkg` installation package, and the `MySQL.prefPane`. Double-click the disk image to open it.
2. Double-click the `MySQLStartItem.pkg` file to start the installation process.
3. You will be presented with the **Install MySQL Startup Item** dialog.

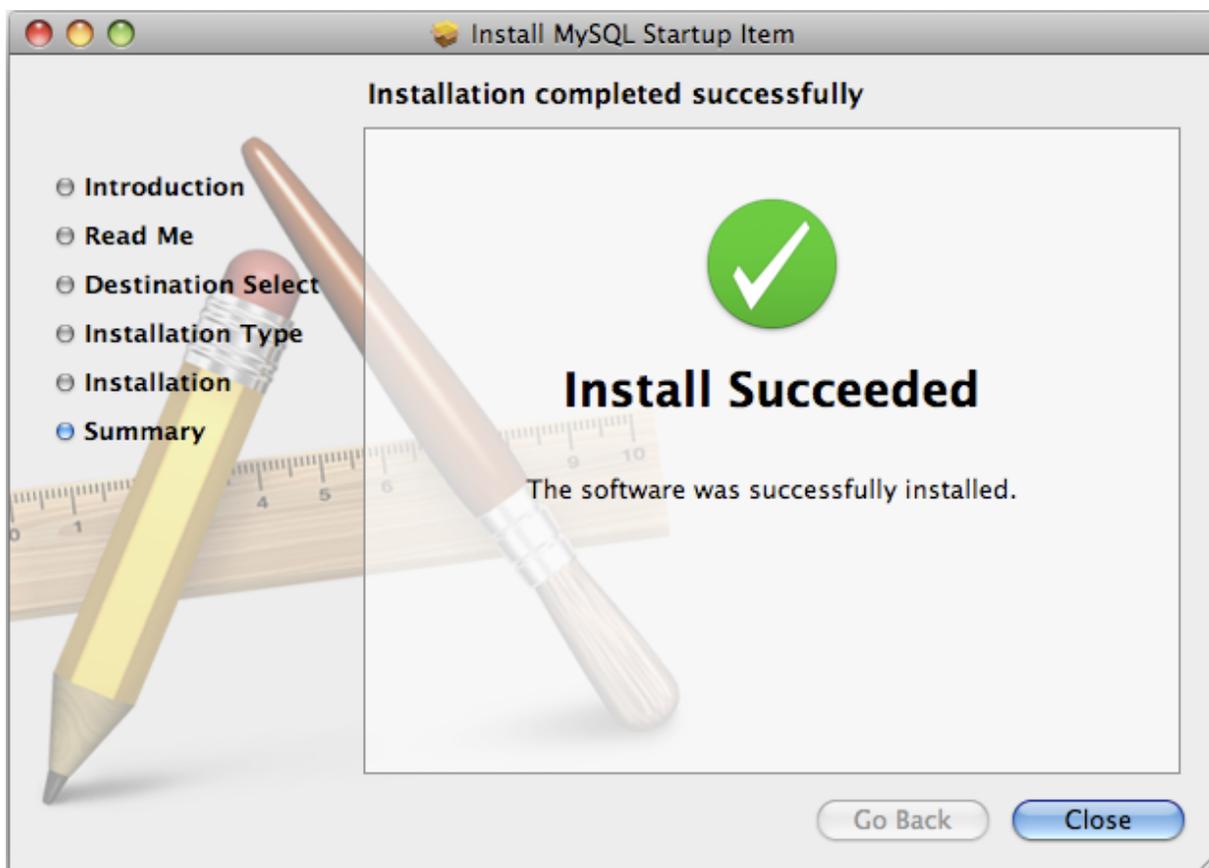


Click **Continue** to continue the installation process.

4. A copy of the installation instructions and other important information relevant to this installation are displayed. Click **Continue**.
5. Select the drive you want to use to install the MySQL Startup Item. The drive must have a valid, bootable, Mac OS X operating system installed. Click **Continue**.



6. You will be asked to confirm the details of the installation. To change the drive on which the startup item is installed, click either **Go Back** or **Change Install Location...**. To install the startup item, click **Install**.
7. Once the installation has been completed successfully, you will be shown an **Install Succeeded** message.



The Startup Item for MySQL is installed into `/Library/StartupItems/MySQLCOM`. The Startup Item installation adds a variable `MYSQLCOM=-YES-` to the system configuration file `/etc/hostconfig`. If you want to disable the automatic startup of MySQL, change this variable to `MYSQLCOM=-NO-`.

After the installation, you can start and stop MySQL by running the following commands in a terminal window. You must have administrator privileges to perform these tasks, and you may be prompted for your password.

If you have installed the Startup Item, use this command to start the server:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
```

If you have installed the Startup Item, use this command to stop the server:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM stop
```

2.4.4. Installing and Using the MySQL Preference Pane

The MySQL Package installer disk image also includes a custom MySQL Preference Pane that enables you to start, stop, and control automated startup during boot of your MySQL installation.

To install the MySQL Preference Pane:

1. Download and open the MySQL package installer package, which is provided on a disk image (`.dmg`) that includes the main MySQL installation package, the `MySQLStartupItem.pkg` installation package, and the `MySQL.prefPane`. Double-click the disk image to open it.
2. Double-click the `MySQL.prefPane`. The MySQL System Preferences will open.

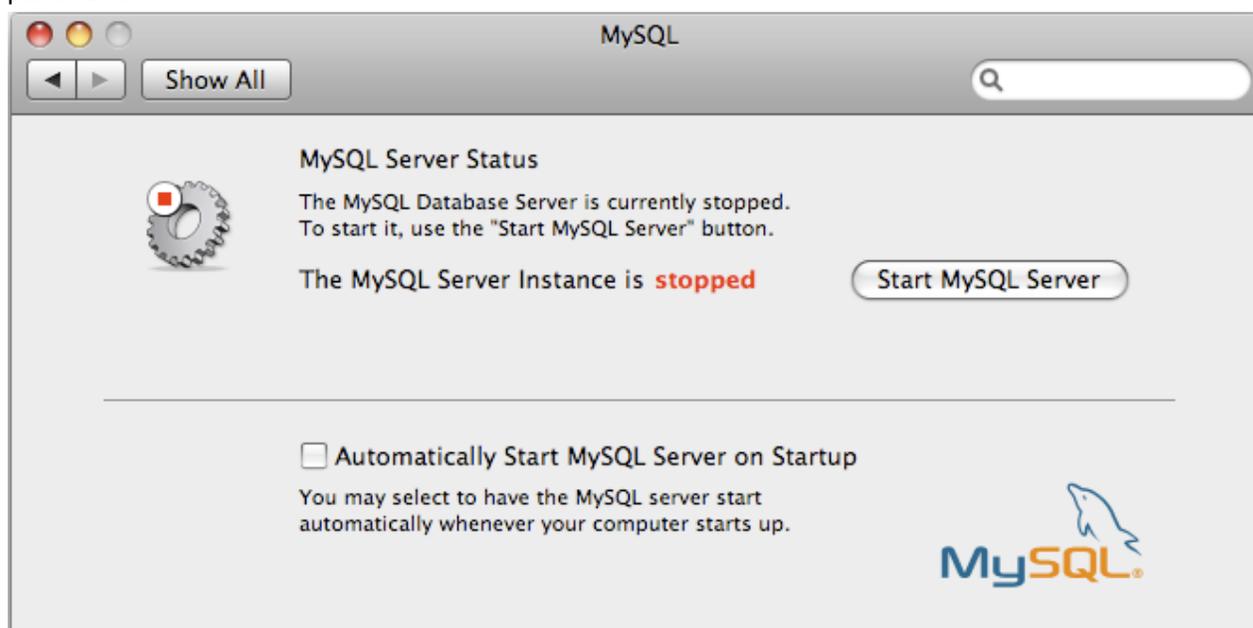
3. If this is the first time you have installed the preference pane, you will be asked to confirm installation and whether you want to install the preference pane for all users, or only the current user. To install the preference pane for all users you will need administrator privileges. If necessary, you will be prompted for the username and password for a user with administrator privileges.
4. If you already have the MySQL Preference Pane installed, you will be asked to confirm whether you want to overwrite the existing MySQL Preference Pane.



Note

The MySQL Preference Pane only starts and stops MySQL installation installed from the MySQL package installation that have been installed in the default location.

Once the MySQL Preference Pane has been installed, you can control your MySQL server instance using the preference pane. To use the preference pane, open the **System Preferences...** from the Apple menu. Select the MySQL preference pane by clicking the MySQL logo within the **Other** section of the preference panes list.



The MySQL Preference Pane shows the current status of the MySQL server, showing **stopped** (in red) if the server is not running and **running** (in green) if the server has already been started. The preference pane also shows the current setting for whether the MySQL server has been set to start automatically.

- **To start MySQL using the preference pane:**

Click **Start MySQL Server**. You may be prompted for the username and password of a user with administrator privileges to start the MySQL server.

- **To stop MySQL using the preference pane:**

Click **Stop MySQL Server**. You may be prompted for the username and password of a user with administrator privileges to stop the MySQL server.

- **To automatically start the MySQL server when the system boots:**

Check the check box next to **Automatically Start MySQL Server on Startup**.

- **To disable automatic MySQL server startup when the system boots:**

Uncheck the check box next to **Automatically Start MySQL Server on Startup**.

You can close the `System Preferences...` window once you have completed your settings.

2.4.5. Using the Bundled MySQL on Mac OS X Server

If you are running Mac OS X Server, a version of MySQL should already be installed. The following table shows the versions of MySQL that ship with Mac OS X Server versions.

Table 2.9. MySQL Versions Preinstalled with Mac OS X Server

Mac OS X Server Version	MySQL Version
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16
10.4.0	4.1.10a
10.5.0	5.0.45
10.6.0	5.0.82

The following table shows the installation layout of MySQL on Mac OS X Server.

Table 2.10. MySQL Directory Layout for Preinstalled MySQL Installations on Mac OS X Server

Directory	Contents of Directory
<code>/usr/bin</code>	Client programs
<code>/var/mysql</code>	Log files, databases
<code>/usr/libexec</code>	The <code>mysqld</code> server
<code>/usr/share/man</code>	Unix manual pages
<code>/usr/share/mysql/mysql-test</code>	MySQL test suite
<code>/usr/share/mysql</code>	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
<code>/var/mysql/mysql.sock</code>	Location of the MySQL Unix socket

Additional Resources

- For more information on managing the bundled MySQL instance in Mac OS X Server 10.5, see [Mac OS X Server: Web Technologies Administration For Version 10.5 Leopard](#).
- For more information on managing the bundled MySQL instance in Mac OS X Server 10.6, see [Mac OS X Server: Web Technologies Administration Version 10.6 Snow Leopard](#).
- The MySQL server bundled with Mac OS X Server does not include the MySQL client libraries and header files required to access and use MySQL from a third-party driver, such as Perl DBI or PHP. For more information on obtaining and installing MySQL libraries, see [Mac OS X Server version 10.5: MySQL libraries available for download](#). Alternatively, you can ignore the bundled MySQL server and install MySQL from the package or tarball installation.

2.5. Installing MySQL on Linux

Linux supports a number of different solutions for installing MySQL. The recommended method is to use one of the distributions from Oracle. If you choose this method, there are three options available:

- Installing from a generic binary package in `.tar.gz` format. See [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#) for more information.
- Extracting and compiling MySQL from a source distribution. For detailed instructions, see [Section 2.10, “Installing MySQL from Source”](#).
- Installing using a pre-compiled RPM package. For more information on using the RPM solution, see [Section 2.5.1, “Installing MySQL from RPM Packages on Linux”](#).
- Installing using Oracle's Unbreakable Linux Network (ULN). For more information, see [Section 2.6, “Installing MySQL Using Unbreakable Linux Network \(ULN\)”](#).

As an alternative, you can use the native package manager within your Linux distribution to automatically download and install MySQL for you. Native package installations can take of the download and dependencies required to run MySQL, but the MySQL version will often be some way behind the currently available release. You will also normally be unable to install developmental releases, as these are not usually made available in the native repository. For more information on using the native package installers, see [Section 2.5.2, “Installing MySQL on Linux using Native Package Manager”](#).



Note

For many Linux installations, you will want to set up MySQL to be started automatically when your machine starts. Many of the native package installations perform this operation for you, but for source, binary and RPM solutions you may need to set this up separately. The required script, `mysql.server`, can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See [Section 2.11.1.2, “Starting and Stopping MySQL Automatically”](#).

2.5.1. Installing MySQL from RPM Packages on Linux

The recommended way to install MySQL on RPM-based Linux distributions is by using the RPM packages. The RPMs that we provide to the community should work on all versions of Linux that support RPM packages and use `glibc` 2.3. To obtain RPM packages, see [Section 2.1.3, “How to Get MySQL”](#).

For non-RPM Linux distributions, you can install MySQL using a `.tar.gz` package. See [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#).

Installations created from our Linux RPM distributions result in files under the following system directories.

Table 2.11. MySQL Installation Layout for Linux RPM

Directory	Contents of Directory
<code>/usr/bin</code>	Client programs and scripts
<code>/usr/sbin</code>	The <code>mysqld</code> server
<code>/var/lib/mysql</code>	Log files, databases
<code>/usr/share/info</code>	Manual in Info format
<code>/usr/share/man</code>	Unix manual pages
<code>/usr/include/mysql</code>	Include (header) files

Directory	Contents of Directory
<code>/usr/lib/mysql</code>	Libraries
<code>/usr/share/mysql</code>	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
<code>/usr/share/sql-bench</code>	Benchmarks



Note

RPM distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by us in features, capabilities, and conventions (including communication setup), and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead. Because of these differences, RPM packages built by us check whether such RPMs built by other vendors are installed. If so, the RPM does not install and produces a message explaining this.

In most cases, you need to install only the `MySQL-server` and `MySQL-client` packages to get a functional MySQL installation. The other packages are not required for a standard installation.

RPMs for MySQL Cluster. Standard MySQL server RPMs built by MySQL do not provide support for the `NDBCLUSTER` storage engine.



Important

When upgrading a MySQL Cluster RPM installation, you must upgrade *all* installed RPMs, including the `Server` and `Client` RPMs.

For more information about installing MySQL Cluster from RPMs, see [Section 17.2, “MySQL Cluster Installation”](#).

For upgrades, if your installation was originally produced by installing multiple RPM packages, it is best to upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

If you get a dependency failure when trying to install MySQL packages (for example, `error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...`), you should also install the `MySQL-shared-compat` package, which includes the shared libraries for older releases for backward compatibility.

The RPM packages shown in the following list are available. The names shown here use a suffix of `.glibc23.i386.rpm`, but particular packages can have different suffixes, described later.

- `MySQL-server-VERSION.glibc23.i386.rpm`

The MySQL server. You need this unless you only want to connect to a MySQL server running on another machine.

- `MySQL-client-VERSION.glibc23.i386.rpm`

The standard MySQL client programs. You probably always want to install this package.

- `MySQL-devel-VERSION.glibc23.i386.rpm`

The libraries and include files that are needed if you want to compile other MySQL clients, such as the Perl modules.

- `MySQL-shared-VERSION.glibc23.i386.rpm`

This package contains the shared libraries (`libmysqlclient.so*`) that certain languages and applications need to dynamically load and use MySQL. It contains single-threaded and thread-safe libraries. Prior to MySQL 5.5.6, if you install this package, do not install the `MySQL-shared-compat` package.

- `MySQL-shared-compat-VERSION.glibc23.i386.rpm`

This package includes the shared libraries for older releases. It contains single-threaded and thread-safe libraries. Install this package if you have applications installed that are dynamically linked against older versions of MySQL but you want to upgrade to the current version without breaking the library dependencies. Before MySQL 5.5.6, `MySQL-shared-compat` also includes the libraries for the current release, so if you install it, you should not also install `MySQL-shared`. As of 5.5.6, `MySQL-shared-compat` does not include the current library version, so there is no conflict.

As of MySQL 5.5.23, the `MySQL-shared-compat` RPM package enables users of Red Hat-provided `mysql-*-5.1` RPM packages to migrate to Oracle-provided `MySQL-*-5.5` packages. `MySQL-shared-compat` replaces the Red Hat `mysql-libs` package by replacing `libmysqlclient.so` files of the latter package, thus satisfying dependencies of other packages on `mysql-libs`. This change affects only users of Red Hat (or Red Hat-compatible) RPM packages. Nothing is different for users of Oracle RPM packages.

- `MySQL-embedded-VERSION.glibc23.i386.rpm`

The embedded MySQL server library.

- `MySQL-test-VERSION.glibc23.i386.rpm`

This package includes the MySQL test suite.

- `MySQL-VERSION.src.rpm`

This contains the source code for all of the previous packages. It can also be used to rebuild the RPMs on other architectures (for example, Alpha or SPARC).

The suffix of RPM package names (following the `VERSION` value) has the following syntax:

```
.PLATFORM.CPU.rpm
```

The `PLATFORM` and `CPU` values indicate the type of system for which the package is built. `PLATFORM` indicates the platform and `CPU` indicates the processor type or family.

All packages are dynamically linked against `glibc 2.3`. The `PLATFORM` value indicates whether the package is platform independent or intended for a specific platform, as shown in the following table.

Table 2.12. MySQL Linux Installation Packages

<i>PLATFORM</i> Value	Intended Use
<code>glibc23</code>	Platform independent, should run on any Linux distribution that supports <code>glibc 2.3</code>
<code>rhel4, rhel5</code>	Red Hat Enterprise Linux 4 or 5
<code>el6</code>	Enterprise Linux 6
<code>sles10, sles11</code>	SuSE Linux Enterprise Server 10 or 11

In MySQL 5.5, only `glibc23` packages are available currently.

The *CPU* value indicates the processor type or family for which the package is built.

Table 2.13. MySQL Installation Packages for Linux CPU Identifier

<i>CPU</i> Value	Intended Processor Type or Family
i386, i586, i686	Pentium processor or better, 32 bit
x86_64	64-bit x86 processor
ia64	Itanium (IA-64) processor

To see all files in an RPM package (for example, a `MySQL-server-VERSION.glibc23.i386.rpm`), run a command like this:

```
shell> rpm -qpl MySQL-server-VERSION.glibc23.i386.rpm
```

To perform a standard minimal installation, install the server and client RPMs:

```
shell> rpm -i MySQL-server-VERSION.glibc23.i386.rpm
shell> rpm -i MySQL-client-VERSION.glibc23.i386.rpm
```

To install only the client programs, install just the client RPM:

```
shell> rpm -i MySQL-client-VERSION.glibc23.i386.rpm
```

RPM provides a feature to verify the integrity and authenticity of packages before installing them. If you would like to learn more about this feature, see [Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”](#).

The server RPM places data under the `/var/lib/mysql` directory. The RPM also creates a login account for a user named `mysql` (if one does not exist) to use for running the MySQL server, and creates the appropriate entries in `/etc/init.d/` to start the server automatically at boot time. (This means that if you have performed a previous installation and have made changes to its startup script, you may want to make a copy of the script so that you do not lose it when you install a newer RPM.) See [Section 2.11.1.2, “Starting and Stopping MySQL Automatically”](#), for more information on how MySQL can be started automatically on system startup.

In MySQL 5.5.5 and later, during a new installation, the server boot scripts are installed, but the MySQL server is not started at the end of the installation, since the status of the server during an unattended installation is not known.

In MySQL 5.5.5 and later, during an upgrade installation using the RPM packages, if the MySQL server is running when the upgrade occurs, the MySQL server is stopped, the upgrade occurs, and the MySQL server is restarted. If the MySQL server is not already running when the RPM upgrade occurs, the MySQL server is not started at the end of the installation.

If something goes wrong, you can find more information in the binary installation section. See [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#).



Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.11, “Postinstallation Setup and Testing”](#).

During RPM installation, a user named `mysql` and a group named `mysql` are created on the system. This is done using the `useradd`, `groupadd`, and `usermod` commands. Those commands require appropriate administrative privileges, which is required for locally managed users and groups (as listed in the `/etc/passwd` and `/etc/group` files) by the RPM installation process being run by `root`.

If you log in as the `mysql` user, you may find that MySQL displays “Invalid (old?) table or database name” errors that mention `.mysqlgui`, `lost+found`, `.mysqlgui`, `.bash_history`, `.fonts.cache-1`,

`.lesshst`, `.mysql_history`, `.profile`, `.viminfo`, and similar files created by MySQL or operating system utilities. You can safely ignore these error messages or remove the files or directories that cause them if you do not need them.

For nonlocal user management (LDAP, NIS, and so forth), the administrative tools may require additional authentication (such as a password), and will fail if the installing user does not provide this authentication. Even if they fail, the RPM installation will not abort but succeed, and this is intentional. If they failed, some of the intended transfer of ownership may be missing, and it is recommended that the system administrator then manually ensures some appropriate user and group exists and manually transfers ownership following the actions in the RPM spec file.

2.5.2. Installing MySQL on Linux using Native Package Manager

Many Linux distributions include a version of the MySQL server, client tools, and development components into the standard package management system built into distributions such as Fedora, Debian, Ubuntu, and Gentoo. This section provides basic instructions for installing MySQL using these systems.



Important

Native package installations can take care of the download and dependencies required to run MySQL, but the MySQL version will often be some way behind the currently available release. You will also normally be unable to install developmental releases, as these are not usually made available in the native repository.

Distribution specific instructions are shown below:

- **Red Hat Linux, Fedora, CentOS**

For Red Hat and similar distributions, the MySQL distribution is divided into a number of separate packages, `mysql` for the client tools, `mysql-server` for the server and associated tools, and `mysql-libs` for the libraries. The libraries are required if you want to provide connectivity from different languages and environments such as Perl, Python and others.

To install, use the `yum` command to specify the packages that you want to install. For example:

```
root-shell> yum install mysql mysql-server mysql-libs mysql-server
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package mysql.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-libs.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-server.x86_64 0:5.1.48-2.fc13 set to be updated
--> Processing Dependency: perl-DBD-MySQL for package: mysql-server-5.1.48-2.fc13.x86_64
--> Running transaction check
---> Package perl-DBD-MySQL.x86_64 0:4.017-1.fc13 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch          Version           Repository        Size
=====
Installing:
mysql                  x86_64        5.1.48-2.fc13    updates           889 k
mysql-libs             x86_64        5.1.48-2.fc13    updates           1.2 M
mysql-server          x86_64        5.1.48-2.fc13    updates           8.1 M
Installing for dependencies:
perl-DBD-MySQL        x86_64        4.017-1.fc13     updates           136 k
=====
```

Installing MySQL on Linux using Native Package Manager

```
Transaction Summary
=====
Install      4 Package(s)
Upgrade      0 Package(s)

Total download size: 10 M
Installed size: 30 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
Package(s) data still to download: 10 M
(1/4): mysql-5.1.48-2.fc13.x86_64.rpm | 889 kB 00:04
(2/4): mysql-libs-5.1.48-2.fc13.x86_64.rpm | 1.2 MB 00:06
(3/4): mysql-server-5.1.48-2.fc13.x86_64.rpm | 8.1 MB 00:40
(4/4): perl-DBD-MySQL-4.017-1.fc13.x86_64.rpm | 136 kB 00:00
-----
Total                                          201 kB/s | 10 MB 00:52
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing      : mysql-libs-5.1.48-2.fc13.x86_64          1/4
  Installing      : mysql-5.1.48-2.fc13.x86_64             2/4
  Installing      : perl-DBD-MySQL-4.017-1.fc13.x86_64     3/4
  Installing      : mysql-server-5.1.48-2.fc13.x86_64     4/4

Installed:
mysql.x86_64 0:5.1.48-2.fc13          mysql-libs.x86_64 0:5.1.48-2.fc13
mysql-server.x86_64 0:5.1.48-2.fc13

Dependency Installed:
perl-DBD-MySQL.x86_64 0:4.017-1.fc13

Complete!
```

MySQL and the MySQL server should now be installed. A sample configuration file is installed into `/etc/my.cnf`. An init script, to start and stop the server, will have been installed into `/etc/init.d/mysql`. To start the MySQL server use `service`:

```
root-shell> service mysqld start
```

To enable the server to be started and stopped automatically during boot, use `chkconfig`:

```
root-shell> chkconfig --levels 235 mysqld on
```

Which enables the MySQL server to be started (and stopped) automatically at the specified the run levels.

The database tables will have been automatically created for you, if they do not already exist. You should, however, run `mysql_secure_installation` to set the root passwords on your server.

- **Debian, Ubuntu, Kubuntu**

On Debian and related distributions, there are two packages, `mysql-client` and `mysql-server`, for the client and server components respectively. You should specify an explicit version, for example `mysql-client-5.1`, to ensure that you install the version of MySQL that you want.

To download and install, including any dependencies, use the `apt-get` command, specifying the packages that you want to install.

**Note**

Before installing, make sure that you update your `apt-get` index files to ensure you are downloading the latest available version.

A sample installation of the MySQL packages might look like this (some sections trimmed for clarity):

```
root-shell> apt-get install mysql-client-5.1 mysql-server-5.1
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-2.6.28-11 linux-headers-2.6.28-11-generic
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  bsd-mailx libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient15off libmysqlclient16 libnet-daemon-perl libplrpc-perl mailx
  mysql-common postfix
Suggested packages:
  dbshell libipc-sharedcache-perl tinyca procmail postfix-mysql postfix-pgsql
  postfix-ldap postfix-pcre sasl2-bin resolvconf postfix-cdb
The following NEW packages will be installed:
  bsd-mailx libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient15off libmysqlclient16 libnet-daemon-perl libplrpc-perl mailx
  mysql-client-5.1 mysql-common mysql-server-5.1 postfix
0 upgraded, 13 newly installed, 0 to remove and 182 not upgraded.
Need to get 1907kB/25.3MB of archives.
After this operation, 59.5MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get: 1 http://gb.archive.ubuntu.com jaunty-updates/main mysql-common 5.1.30really5.0.75-0ubuntu10.5 [63.
Get: 2 http://gb.archive.ubuntu.com jaunty-updates/main libmysqlclient15off 5.1.30really5.0.75-0ubuntu10
Fetched 1907kB in 9s (205kB/s)
Preconfiguring packages ...
Selecting previously deselected package mysql-common.
(Reading database ... 121260 files and directories currently installed.)
...
Processing 1 added doc-base file(s)...
Registering documents with scrollkeeper...
Setting up libnet-daemon-perl (0.43-1) ...
Setting up libplrpc-perl (0.2020-1) ...
Setting up libdbi-perl (1.607-1) ...
Setting up libmysqlclient15off (5.1.30really5.0.75-0ubuntu10.5) ...

Setting up libdbd-mysql-perl (4.008-1) ...
Setting up libmysqlclient16 (5.1.31-lubuntu2) ...

Setting up mysql-client-5.1 (5.1.31-lubuntu2) ...

Setting up mysql-server-5.1 (5.1.31-lubuntu2) ...
* Stopping MySQL database server mysqld
  ...done.
100825 11:46:15 InnoDB: Started; log sequence number 0 46409
100825 11:46:15 InnoDB: Starting shutdown...
100825 11:46:17 InnoDB: Shutdown completed; log sequence number 0 46409
100825 11:46:17 [Warning] Forcing shutdown of 1 plugins
* Starting MySQL database server mysqld
  ...done.
* Checking for corrupt, not cleanly closed and upgrade needing tables.
...
Processing triggers for libc6 ...
ldconfig deferred processing now taking place
```



Note

The `apt-get` command will install a number of packages, including the MySQL server, in order to provide the typical tools and application environment. This can mean that you install a large number of packages in addition to the main MySQL package.

During installation, the initial database will be created, and you will be prompted for the MySQL root password (and confirmation). A configuration file will have been created in `/etc/mysql/my.cnf`. An init script will have been created in `/etc/init.d/mysql`.

The server will already be started. You can manually start and stop the server using:

```
root-shell> service mysql [start|stop]
```

The service will automatically be added to the 2, 3 and 4 run levels, with stop scripts in the single, shutdown and restart levels.

• Gentoo Linux

As a source-based distribution, installing MySQL on Gentoo involves downloading the source, patching the Gentoo specifics, and then compiling the MySQL server and installing it. This process is handled automatically by the `emerge` command. Depending on the version of MySQL that you want to install, you may need to unmask the specific version that you want for your chosen platform.

The MySQL server and client tools are provided within a single package, `dev-db/mysql`. You can obtain a list of the versions available to install by looking at the portage directory for the package:

```
root-shell> ls /usr/portage/dev-db/mysql/mysql-5.1*
mysql-5.1.39-r1.ebuild
mysql-5.1.44-r1.ebuild
mysql-5.1.44-r2.ebuild
mysql-5.1.44-r3.ebuild
mysql-5.1.44.ebuild
mysql-5.1.45-r1.ebuild
mysql-5.1.45.ebuild
mysql-5.1.46.ebuild
```

To install a specific MySQL version, you must specify the entire atom. For example:

```
root-shell> emerge =dev-db/mysql-5.1.46
```

A simpler alternative is to use the `virtual/mysql-5.1` package, which will install the latest version:

```
root-shell> emerge =virtual/mysql-5.1
```

If the package is masked (because it is not tested or certified for the current platform), use the `ACCEPT_KEYWORDS` environment variable. For example:

```
root-shell> ACCEPT_KEYWORDS="~x86" emerge =virtual/mysql-5.1
```

After installation, you should create a new database using `mysql_install_db`, and set the password for the root user on MySQL. You can use the configuration interface to set the password and create the initial database:

```
root-shell> emerge --config =dev-db/mysql-5.1.46
```

A sample configuration file will have been created for you in `/etc/mysql/my.cnf`, and an init script will have been created in `/etc/init.d/mysql`.

To enable MySQL to start automatically at the normal (default) run levels, you can use:

```
root-shell> rc-update add mysql default
```

2.6. Installing MySQL Using Unbreakable Linux Network (ULN)

Linux supports a number of different solutions for installing MySQL, covered in [Section 2.5, “Installing MySQL on Linux”](#). One of the methods, covered in this section, is installing from Oracle's Unbreakable Linux Network (ULN). You can find information about Oracle Linux and ULN under <http://linux.oracle.com/>.

To use ULN, you need to obtain a ULN login and register the machine used for installation with ULN. This is described in detail in the [ULN FAQ](#). This page also describes how to install and update packages.



Note

At the time of writing this (September 2012), ULN provides MySQL 5.5 for Oracle Linux 6.

Once MySQL has been installed using ULN, you can find information on starting and stopping the server, and more, in [this section](#), particularly under [Section 2.5.1, “Installing MySQL from RPM Packages on Linux”](#).

If you're updating an existing MySQL installation to an installation using ULN, the recommended procedure is to export your data using `mysqldump`, remove the existing installation, install MySQL from ULN, and load the exported data into your freshly installed MySQL.

If the existing MySQL installation you're upgrading from is from a previous release series (prior to MySQL 5.5), make sure to read the section on upgrading MySQL, [Section 2.12.1, “Upgrading MySQL”](#).

2.7. Installing MySQL on Solaris and OpenSolaris

MySQL on Solaris and OpenSolaris is available in a number of different formats.

- For information on installing using the native Solaris `PKG` format, see [Section 2.7.1, “Installing MySQL on Solaris using a Solaris `PKG`”](#).
- On OpenSolaris, the standard package repositories include MySQL packages specially built for OpenSolaris that include entries for the Service Management Framework (SMF) to enable control of the installation using the SMF administration commands. For more information, see [Section 2.7.2, “Installing MySQL on OpenSolaris using IPS”](#).
- To use a standard `tar` binary installation, use the notes provided in [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#). Check the notes and hints at the end of this section for Solaris specific notes that you may need before or after installation.

To obtain a binary MySQL distribution for Solaris in tarball or `PKG` format, <http://dev.mysql.com/downloads/mysql/5.5.html>.

Additional notes to be aware of when installing and using MySQL on Solaris:

- If you want to use MySQL with the `mysql` user and group, use the `groupadd` and `useradd` commands:

```
groupadd mysql
useradd -g mysql mysql
```

- If you install MySQL using a binary tarball distribution on Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution. In Solaris 10 and OpenSolaris `gtar` is normally located in `/usr/sfw/bin/gtar`, but may not be included in the default path definition.

- When using Solaris 10 for `x86_64`, you should mount any file systems on which you intend to store `InnoDB` files with the `forcedirectio` option. (By default mounting is done without this option.) Failing to do so will cause a significant drop in performance when using the `InnoDB` storage engine on this platform.
- If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.
- If too many processes try to connect very rapidly to `mysqld`, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

You might try starting the server with the `--back_log=50` [485] option as a workaround for this.

- To configure the generation of core files on Solaris you should use the `coreadm` command. Because of the security implications of generating a core on a `setuid()` application, by default, Solaris does not support core files on `setuid()` programs. However, you can modify this behavior using `coreadm`. If you enable `setuid()` core files for the current user, they will be generated using the mode 600 and owned by the superuser.

2.7.1. Installing MySQL on Solaris using a Solaris **PKG**

You can install MySQL on Solaris and OpenSolaris using a binary package using the native Solaris **PKG** format instead of the binary tarball distribution.

To use this package, download the corresponding `mysql-VERSION-solaris10-PLATFORM.pkg.gz` file, then decompress it. For example:

```
shell> gunzip mysql-5.5.31-solaris10-x86_64.pkg.gz
```

To install a new package, use `pkgadd` and follow the onscreen prompts. You must have root privileges to perform this operation:

```
shell> pkgadd -d mysql-5.5.31-solaris10-x86_64.pkg

The following packages are available:
  1  mysql      MySQL Community Server (GPL)
      (i86pc) 5.5.31

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

The **PKG** installer installs all of the files and tools needed, and then initializes your database if one does not exist. To complete the installation, you should set the root password for MySQL as provided in the instructions at the end of the installation. Alternatively, you can run the `mysql_secure_installation` script that comes with the installation.

The default installation directory is `/opt/mysql`. You can only change the root path of the installation when using `pkgadd`, which can be used to install MySQL in a different Solaris zone. If you need to install in a specific directory, use the binary `tar` file.

The `pkg` installer copies a suitable startup script for MySQL into `/etc/init.d/mysql`. To enable MySQL to startup and shutdown automatically, you should create a link between this file and the init script

directories. For example, to ensure safe startup and shutdown of MySQL you could use the following commands to add the right links:

```
shell> ln /etc/init.d/mysql /etc/rc3.d/S91mysql
shell> ln /etc/init.d/mysql /etc/rc0.d/K02mysql
```

To remove MySQL, the installed package name is `mysql`. You can use this in combination with the `pkgrm` command to remove the installation.

To upgrade when using the Solaris package file format, you must remove the existing installation before installing the updated package. Removal of the package does not delete the existing database information, only the server, binaries and support files. The typical upgrade sequence is therefore:

```
shell> mysqladmin shutdown
shell> pkgrm mysql
shell> pkgadd -d mysql-5.5.31-solaris10-x86_64.pkg
shell> mysqld_safe &
shell> mysql_upgrade
```

You should check the notes in [Section 2.12, “Upgrading or Downgrading MySQL”](#) before performing any upgrade.

2.7.2. Installing MySQL on OpenSolaris using IPS

OpenSolaris includes standard packages for MySQL in the core repository. The MySQL packages are based on a specific release of MySQL and updated periodically. For the latest release you must use either the native Solaris `PKG`, `tar`, or source installations. The native OpenSolaris packages include SMF files so that you can easily control your MySQL installation, including automatic startup and recovery, using the native service management tools.

To install MySQL on OpenSolaris, use the `pkg` command. You will need to be logged in as root, or use the `pfexec` tool, as shown in the example below:

```
shell> pfexec pkg install SUNWmysql155
```

The package set installs three individual packages, `SUNWmysql155lib`, which contains the MySQL client libraries; `SUNWmysql155r` which contains the root components, including SMF and configuration files; and `SUNWmysql155u` which contains the scripts, binary tools and other files. You can install these packages individually if you only need the corresponding components.

The MySQL files are installed into `/usr/mysql` which symbolic links for the sub directories (`bin`, `lib`, etc.) to a version specific directory. For MySQL 5.5, the full installation is located in `/usr/mysql/5.5`. The default data directory is `/var/mysql/5.5/data`. The configuration file is installed in `/etc/mysql/5.5/my.cnf`. This layout permits multiple versions of MySQL to be installed, without overwriting the data and binaries from other versions.

Once installed, you must run `mysql_install_db` to initialize the database, and use the `mysql_secure_installation` to secure your installation.

Using SMF to manage your MySQL installation

Once installed, you can start and stop your MySQL server using the installed SMF configuration. The service name is `mysql`, or if you have multiple versions installed, you should use the full version name, for example `mysql:version_55`. To start and enable MySQL to be started at boot time:

```
shell> svcadm enable mysql
```

To disable MySQL from starting during boot time, and shut the MySQL server down if it is running, use:

```
shell> svcadm disable mysql
```

To restart MySQL, for example after a configuration file changes, use the `restart` option:

```
shell> svcadm restart mysql
```

You can also use SMF to configure the data directory and enable full 64-bit mode. For example, to set the data directory used by MySQL:

```
shell> svccfg
svc:> select mysql:version_55
svc:/application/database/mysql:version_55> setprop mysql/data=/data0/mysql
```

By default, the 32-bit binaries are used. To enable the 64-bit server on 64-bit platforms, set the `enable_64bit` parameter. For example:

```
svc:/application/database/mysql:version_55> setprop mysql/enable_64bit=1
```

You need to refresh the SMF after settings these options:

```
shell> svcadm refresh mysql
```

2.8. Installing MySQL on HP-UX

MySQL for HP-UX is available in a number of different forms:

- Using a DEPOT distribution provided at <http://dev.mysql.com/downloads/>. Please read the [general notes on HP-UX installation](#) before continuing. For more information on DEPOT installations, see [Section 2.8.2, “Installing MySQL on HP-UX using DEPOT”](#).
- Using a binary tarball distribution provided at <http://dev.mysql.com/downloads/>. Please read the [general notes on HP-UX installation](#) before continuing. For more information on binary installations, see [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#).

2.8.1. General Notes on Installing MySQL on HP-UX

Some additional notes on installing and using MySQL on HP-UX:

- If you install MySQL using a binary tarball distribution on HP-UX, you may run into trouble even before you get the MySQL distribution unpacked, as the HP-UX `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution.

- Because of some critical bugs in the standard HP-UX libraries, you should install the following patches before trying to run MySQL on HP-UX 11.0:

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

This solves the problem of getting `EWOULDBLOCK` from `recv()` and `EBADF` from `accept()` in threaded applications.

2.8.2. Installing MySQL on HP-UX using DEPOT

The HP-UX DEPOT format packages can be installed using the `swinstall` command. You should install the `ncurses` and `zlib` libraries before installing the MySQL DEPOT package. You can use the free software `depothelper` tool to install these packages and any dependencies for you automatically.

To install using the MySQL DEPOT packages, follow this guide:

1. Download the MySQL DEPOT package from <http://dev.mysql.com/downloads/>. You must decompress the package before installation:

```
root-shell> gunzip mysql-5.1.48-hpux11.31-ia64-64bit.depot.gz
```

2. Install the DEPOT package using `swinstall`:

```
root-shell> swinstall -s mysql-5.1.49-hpux11.31-ia64-64bit.depot
```

MySQL will be installed into a directory matching the depot package name, within `/usr/local`. For convenience, you may want to create a symbolic link to the installed directory, for example:

```
root-shell> ln -s mysql-5.1.49-hpux11.31-ia64-64bit mysql
```

3. Your package is now installed. You should complete the configuration of MySQL by creating a user and group:

```
root-shell> /usr/sbin/groupadd mysql
root-shell> /usr/sbin/useradd -g mysql -d /var/lib/mysql/ -s /bin/false mysql
```

4. Create the standard database using the new user/group you have created, and set the permissions:

```
root-shell> cd /usr/local/
root-shell> scripts/mysql_install_db --user=mysql
root-shell> chown -R root .
root-shell> chown -R mysql data
```

5. Finally, secure your new installation by setting the root passwords, and then start your MySQL server using the `mysql` user:

```
root-shell> mysql_secure_installation
root-shell> mysqld_safe --user=mysql &
```

2.9. Installing MySQL on FreeBSD

This section provides information about installing MySQL on variants of FreeBSD Unix.

You can install MySQL on FreeBSD by using the binary distribution provided by Oracle. For more information, see [Section 2.2, "Installing MySQL from Generic Binaries on Unix/Linux"](#).

The easiest (and preferred) way to install MySQL is to use the `mysql-server` and `mysql-client` ports available at <http://www.freebsd.org/>. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.
- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

The MySQL build process requires GNU make (`gmake`) to work. If GNU `make` is not available, you must install it first before compiling MySQL.

To install using the ports system:

```
# cd /usr/ports/databases/mysql51-server
# make
...
# cd /usr/ports/databases/mysql51-client
# make
...
```

The standard port installation places the server into `/usr/local/libexec/mysqld`, with the startup script for the MySQL server placed in `/usr/local/etc/rc.d/mysql-server`.

Some additional notes on the BSD implementation:

- To remove MySQL after installation using the ports system:

```
# cd /usr/ports/databases/mysql51-server
# make deinstall
...
# cd /usr/ports/databases/mysql51-client
# make deinstall
...
```

- If you get problems with the current date in MySQL, setting the `TZ` variable should help. See [Section 2.13, “Environment Variables”](#).

2.10. Installing MySQL from Source

Building MySQL from the source code enables you to customize build parameters, compiler optimizations, and installation location. For a list of systems on which MySQL is known to run, see [Section 2.1.1, “Operating Systems Supported by MySQL Community Server”](#).

Before you proceed with an installation from source, check whether Oracle produces a precompiled binary distribution for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options for optimal performance. Instructions for installing binary distributions are available in [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#).



Note

This section describes how to build MySQL from source using `CMake`. Before MySQL 5.5, source builds used the GNU autotools on Unix-like systems. Source builds on Windows used `CMake`, but the process was different from that described here. For source-building instructions for older versions of MySQL, see [Installing MySQL from Source](#), in the MySQL 5.1 Reference Manual. If you are familiar with autotools but not `CMake`, you might find these transition instructions helpful: [Autotools to CMake Transition Guide](#)

Source Installation Methods

There are two methods for installing MySQL from source:

- Use a standard MySQL source distribution. To obtain a standard distribution, see [Section 2.1.3, “How to Get MySQL”](#). For instructions on building from a standard distribution, see [Section 2.10.2, “Installing MySQL from a Standard Source Distribution”](#).

Standard distributions are available as compressed `tar` files, Zip archives, or RPM packages. Distribution files have names of the form `mysql-VERSION.tar.gz`, `mysql-VERSION.zip`, or `mysql-VERSION.rpm`, where `VERSION` is a number like `5.5.31`. File names for source distributions can be distinguished from those for precompiled binary distributions in that source distribution names

are generic and include no platform name, whereas binary distribution names include a platform name indicating the type of system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

- Use a MySQL development tree. Development trees have not necessarily received the same level of testing as standard release distributions, so this installation method is usually required only if you need the most recent code changes. For information on building from one of the development trees, see [Section 2.10.3, “Installing MySQL from a Development Source Tree”](#).

Source Installation System Requirements

Installation of MySQL from source requires several development tools. Some of these tools are needed no matter whether you use a standard source distribution or a development source tree. Other tool requirements depend on which installation method you use.

To install MySQL from source, your system must have the following tools, regardless of installation method:

- [CMake](#), which is used as the build framework on all platforms. [CMake](#) can be downloaded from <http://www.cmake.org>.
- A good [make](#) program. Although some platforms come with their own [make](#) implementations, it is highly recommended that you use GNU [make](#) 3.75 or newer. It may already be available on your system as [gmake](#). GNU [make](#) is available from <http://www.gnu.org/software/make/>.
- A working ANSI C++ compiler. GCC 4.2.1 or later, Sun Studio 10 or later, Visual Studio 2008 or later, and many current vendor-supplied compilers are known to work.
- Perl is needed if you intend to run test scripts. Most Unix-like systems include Perl. On Windows, you can use a version such as ActiveState Perl.

To install MySQL from a standard source distribution, one of the following tools is required to unpack the distribution file:

- For a `.tar.gz` compressed [tar](#) file: GNU [gunzip](#) to uncompress the distribution and a reasonable [tar](#) to unpack it. If your [tar](#) program supports the `z` option, it can both uncompress and unpack the file.

GNU [tar](#) is known to work. The standard [tar](#) provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU [tar](#), or if available, use a preinstalled version of GNU tar. Usually this is available as [gnutar](#), [gtar](#), or as [tar](#) within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU [tar](#) is available from <http://www.gnu.org/software/tar/>.

- For a `.zip` Zip archive: [WinZip](#) or another tool that can read `.zip` files.
- For an `.rpm` RPM package: The [rpmbuild](#) program used to build the distribution unpacks it.

To install MySQL from a development source tree, the following additional tools are required:

- To obtain the source tree, you must have Bazaar installed. The [Bazaar VCS Web site](#) has instructions for downloading and installing Bazaar on different platforms. Bazaar is supported on any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows, or Mac OS X host.
- [bison](#) is needed to generate `sql_yacc.cc` from `sql_yacc.yy`. You should use the latest version of [bison](#) where possible. Versions 1.75 and 2.1 are known to work. There have been reported problems with [bison](#) 1.875. If you experience problems, upgrade to a later, rather than earlier, version.

`bison` is available from <http://www.gnu.org/software/bison/>. `bison` for Windows can be downloaded from <http://gnuwin32.sourceforge.net/packages/bison.htm>. Download the package labeled “Complete package, excluding sources”. On Windows, the default location for `bison` is the `C:\Program Files\GnuWin32` directory. Some utilities may fail to find `bison` because of the space in the directory name. Also, Visual Studio may simply hang if there are spaces in the path. You can resolve these problems by installing into a directory that does not contain a space; for example `C:\GnuWin32`.

- On OpenSolaris and Solaris Express, `m4` must be installed in addition to `bison`. `m4` is available from <http://www.gnu.org/software/m4/>.

**Note**

If you have to install any programs, modify your `PATH` environment variable to include any directories in which the programs are located. See [Section 4.2.4, “Setting Environment Variables”](#).

If you run into problems and need to file a bug report, please use the instructions in [Section 1.7, “How to Report Bugs or Problems”](#).

2.10.1. MySQL Layout for Source Installation

By default, when you install MySQL after compiling it from source, the installation step installs files under `/usr/local/mysql`. The component locations under the installation directory are the same as for binary distributions. See [Table 2.2, “MySQL Installation Layout for Generic Unix/Linux Binary Package”](#), and [Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”](#). To configure installation locations different from the defaults, use the options described at [Section 2.10.4, “MySQL Source-Configuration Options”](#).

2.10.2. Installing MySQL from a Standard Source Distribution

To install MySQL from a standard source distribution:

1. Verify that your system satisfies the tool requirements listed at [Section 2.10, “Installing MySQL from Source”](#).
2. Obtain a distribution file using the instructions in [Section 2.1.3, “How to Get MySQL”](#).
3. Configure, build, and install the distribution using the instructions in this section.
4. Perform postinstallation procedures using the instructions in [Section 2.11, “Postinstallation Setup and Testing”](#).

In MySQL 5.5, `CMake` is used as the build framework on all platforms. The instructions given here should enable you to produce a working installation. For additional information on using `CMake` to build MySQL, see [How to Build MySQL Server with CMake](#).

If you start from a source RPM, use the following command to make a binary RPM that you can install. If you do not have `rpmbuild`, use `rpm` instead.

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

The result is one or more binary RPM packages that you install as indicated in [Section 2.5.1, “Installing MySQL from RPM Packages on Linux”](#).

The sequence for installation from a compressed `tar` file or Zip archive source distribution is similar to the process for installing from a generic binary distribution (see [Section 2.2, “Installing MySQL from Generic](#)

Binaries on Unix/Linux”), except that it is used on all platforms and includes steps to configure and compile the distribution. For example, with a compressed `tar` file source distribution on Unix, the basic installation command sequence looks like this:

```
# Preconfiguration setup
shell> groupadd mysql
shell> useradd -r -g mysql mysql
# Beginning of source-build specific instructions
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> cmake .
shell> make
shell> make install
# End of source-build specific instructions
# Postinstallation setup
shell> cd /usr/local/mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
# Next command is optional
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

A more detailed version of the source-build specific instructions is shown following.



Note

The procedure shown here does not set up any passwords for MySQL accounts. After following the procedure, proceed to [Section 2.11, “Postinstallation Setup and Testing”](#), for postinstallation setup and testing.

Perform Preconfiguration Setup

On Unix, set up the `mysql` user and group that will be used to run and execute the MySQL server and own the database directory. For details, see [Creating a `mysql` System User and Group](#), in [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#). Then perform the following steps as the `mysql` user, except as noted.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it.

Obtain a distribution file using the instructions in [Section 2.1.3, “How to Get MySQL”](#).

Unpack the distribution into the current directory:

- To unpack a compressed `tar` file, `tar` can uncompress and unpack the distribution if it has `z` option support:

```
shell> tar zxvf mysql-VERSION.tar.gz
```

If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it:

```
shell> gunzip < mysql-VERSION.tar.gz | tar xvf -
```

Alternatively, `CMake` can uncompress and unpack the distribution:

```
shell> cmake -E tar zxvf mysql-VERSION.tar.gz
```

- To unpack a Zip archive, use [WinZip](#) or another tool that can read `.zip` files.

Unpacking the distribution file creates a directory named `mysql-VERSION`.

Configure the Distribution

Change location into the top-level directory of the unpacked distribution:

```
shell> cd mysql-VERSION
```

Configure the source directory. The minimum configuration command includes no options to override configuration defaults:

```
shell> cmake .
```

On Windows, specify the development environment. For example, the following commands configure MySQL for 32-bit or 64-bit builds, respectively:

```
shell> cmake . -G "Visual Studio 9 2008"
shell> cmake . -G "Visual Studio 9 2008 Win64"
```

On Mac OS X, to use the Xcode IDE:

```
shell> cmake . -G Xcode
```

When you run `cmake`, you might want to add options to the command line. Here are some examples:

- `-DBUILD_CONFIG=mysql_release` [150]: Configure the source with the same build options used by Oracle to produce binary distributions for official MySQL releases.
- `-DCMAKE_INSTALL_PREFIX=dir_name` [151]: Configure the distribution for installation under a particular location.
- `-DCPACK_MONOLITHIC_INSTALL=1` [151]: Cause `make package` to generate a single installation file rather than multiple files.
- `-DWITH_DEBUG=1` [155]: Build the distribution with debugging support.

For a more extensive list of options, see [Section 2.10.4, "MySQL Source-Configuration Options"](#).

To list the configuration options, use one of the following commands:

```
shell> cmake . -L # overview
shell> cmake . -LH # overview with help text
shell> cmake . -LAH # all params with help text
shell> ccmake . # interactive display
```

If `CMake` fails, you might need to reconfigure by running it again with different options. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run these commands on Unix before re-running `CMake`:

```
shell> make clean
shell> rm CMakeCache.txt
```

Or, on Windows:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

If you build out of the source tree (as described later), the `CMakeCache.txt` file and all built files are in the build directory, so you can remove that directory to object files and cached configuration information.

If you are going to send mail to a MySQL mailing list to ask for configuration assistance, first check the files in the `CMakeFiles` directory for useful information about the failure. To file a bug report, please use the instructions in [Section 1.7, "How to Report Bugs or Problems"](#).

Build the Distribution

On Unix:

```
shell> make
shell> make VERBOSE=1
```

The second command sets `VERBOSE` to show the commands for each compiled source.

Use `gmake` instead on systems where you are using GNU `make` and it has been installed as `gmake`.

On Windows:

```
shell> devenv MySQL.sln /build RelWithDebInfo
```

It is possible to build out of the source tree to keep the tree clean. If the top-level source directory is named `mysql-src` under your current working directory, you can build in a directory named `build` at the same level like this:

```
shell> mkdir build
shell> cd build
shell> cmake ../mysql-src
```

If you have gotten to the compilation stage, but the distribution does not build, see [Section 2.10.5, "Dealing with Problems Compiling MySQL"](#), for help. If that does not solve the problem, please enter it into our bugs database using the instructions given in [Section 1.7, "How to Report Bugs or Problems"](#). If you have installed the latest versions of the required tools, and they crash trying to process our configuration files, please report that also. However, if you get a `command not found` error or a similar problem for required tools, do not report it. Instead, make sure that all the required tools are installed and that your `PATH` variable is set correctly so that your shell can find them.

Install the Distribution

On Unix:

```
shell> make install
```

This installs the files under the configured installation directory (by default, `/usr/local/mysql`). You might need to run the command as `root`.

To install in a specific directory, add a `DESTDIR` parameter to the command line:

```
shell> make install DESTDIR="/opt/mysql"
```

Alternatively, generate installation package files that you can install where you like:

```
shell> make package
```

This operation produces one or more `.tar.gz` files that can be installed like generic binary distribution packages. See [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#). If you run `CMake` with `-DCPACK_MONOLITHIC_INSTALL=1` [151], the operation produces a single file. Otherwise, it produces multiple files.

On Windows, generate the data directory, then create a `.zip` archive installation package:

```
shell> devenv MySQL.sln /build RelWithDebInfo /project initial_database
shell> devenv MySQL.sln /build RelWithDebInfo /project package
```

You can install the resulting `.zip` archive where you like. See [Section 2.3.7, “Installing MySQL on Microsoft Windows Using a noinstall Zip Archive”](#).

Perform Postinstallation Setup

The remainder of the installation process involves setting up the configuration file, creating the core databases, and starting the MySQL server. For instructions, see [Section 2.11, “Postinstallation Setup and Testing”](#).



Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.11, “Postinstallation Setup and Testing”](#).

2.10.3. Installing MySQL from a Development Source Tree

This section discusses how to install MySQL from the latest development source code. Development trees have not necessarily received the same level of testing as standard release distributions, so this installation method is usually required only if you need the most recent code changes. Do not use a development tree for production systems. If your goal is simply to get MySQL up and running on your system, you should use a standard release distribution (either a binary or source distribution). See [Section 2.1.3, “How to Get MySQL”](#).

MySQL development projects are hosted on [Launchpad](#). MySQL projects, including MySQL Server, MySQL Workbench, and others are available from the [Oracle/MySQL Engineering](#) page. For the repositories related only to MySQL Server, see the [MySQL Server](#) page.

To install MySQL from a development source tree, your system must satisfy the tool requirements listed at [Section 2.10, “Installing MySQL from Source”](#), including the requirements for Bazaar and `bison`.

To create a local branch of the MySQL development tree on your machine, use this procedure:

1. To obtain a copy of the MySQL source code, you must create a new Bazaar branch. If you do not already have a Bazaar repository directory set up, you must initialize a new directory:

```
shell> mkdir mysql-server
shell> bazaar init-repo --trees mysql-server
```

This is a one-time operation.

2. Assuming that you have an initialized repository directory, you can branch from the public MySQL server repositories to create a local source tree. To create a branch of a specific version:

```
shell> cd mysql-server
shell> bzz branch lp:mysql-server/5.5 mysql-5.5
```

This is a one-time operation per source tree. You can branch the source trees for several versions of MySQL under the `mysql-server` directory.

3. The initial download will take some time to complete, depending on the speed of your connection. Please be patient. Once you have downloaded the first tree, additional trees should take significantly less time to download.
4. When building from the Bazaar branch, you may want to create a copy of your active branch so that you can make configuration and other changes without affecting the original branch contents. You can achieve this by branching from the original branch:

```
shell> bzz branch mysql-5.5 mysql-5.5-build
```

5. To obtain changes made after you have set up the branch initially, update it using the `pull` option periodically. Use this command in the top-level directory of the local copy:

```
shell> bzz pull
```

To examine the changeset comments for the tree, use the `log` option to `bzz`:

```
shell> bzz log
```

You can also browse changesets, comments, and source code online at the Launchpad [MySQL Server](#) page.

If you see diffs (changes) or code that you have a question about, do not hesitate to send email to the MySQL [internals](#) mailing list. See [Section 1.6.1, “MySQL Mailing Lists”](#). If you think you have a better idea on how to do something, send an email message to the list with a patch.

After you have the local branch, you can build MySQL server from the source code. For information, see [Section 2.10.2, “Installing MySQL from a Standard Source Distribution”](#), except that you skip the part about obtaining and unpacking the distribution.

Be careful about installing a build from a distribution source tree on a production machine. The installation command may overwrite your live release installation. If you already have MySQL installed and do not want to overwrite it, run `CMake` with values for the `CMAKE_INSTALL_PREFIX` [151], `MYSQL_TCP_PORT` [155], and `MYSQL_UNIX_ADDR` [155] options different from those used by your production server. For additional information about preventing multiple servers from interfering with each other, see [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#).

Play hard with your new installation. For example, try to make new features crash. Start by running `make test`. See [Section 23.1.2, “The MySQL Test Suite”](#).

2.10.4. MySQL Source-Configuration Options

The `CMake` program provides a great deal of control over how you configure a MySQL source distribution. Typically, you do this using options on the `CMake` command line. For information about options supported by `CMake`, run either of these commands in the top-level source directory:

```
shell> cmake . -LH
shell> cmake .
```

You can also affect `CMake` using certain environment variables. See [Section 2.13, “Environment Variables”](#).

The following table shows the available CMake options. In the `Default` column, `PREFIX` stands for the value of the `CMAKE_INSTALL_PREFIX` [151] option, which specifies the installation base directory. This value is used as the parent location for several of the installation subdirectories.

Table 2.14. MySQL Source-Configuration Option Reference (CMake)

Formats	Description	Default	Introduced	Removed
<code>BUILD_CONFIG</code> [150]	Use same build options as official releases		5.5.7	
<code>CMAKE_BUILD_TYPE</code> [151]	Type of build to produce	<code>RelWithDebInfo</code>	5.5.7	
<code>CMAKE_INSTALL_PREFIX</code> [151]	Installation base directory	<code>/usr/local/mysql</code>	5.5.8	
<code>COMPILATION_COMMENT</code> [154]	Comment about compilation environment		5.5.7	
<code>CPACK_MONOLITHIC_INSTALL</code> [151]	Whether package build produces single file	<code>OFF</code>	5.5.7	
<code>DEFAULT_CHARSET</code> [154]	The default server character set	<code>latin1</code>	5.5.7	
<code>DEFAULT_COLLATION</code> [154]	The default server collation	<code>latin1_swedish_ci</code>	5.5.7	
<code>ENABLE_DEBUG_SYNC</code> [154]	Whether to enable Debug Sync support	<code>ON</code>	5.5.7	
<code>ENABLE_DOWNLOADS</code> [154]	Whether to download optional files	<code>OFF</code>	5.5.7	
<code>ENABLE_DTRACE</code> [154]	Whether to include DTrace support		5.5.7	
<code>ENABLE_GCOV</code> [154]	Whether to include gcov support		5.5.14	
<code>ENABLED_LOCAL_INFILE</code> [154]	Whether to enable LOCAL for LOAD DATA INFILE	<code>OFF</code>	5.5.7	
<code>ENABLED_PROFILING</code> [155]	Whether to enable query profiling code	<code>ON</code>	5.5.7	
<code>INSTALL_BINDIR</code> [151]	User executables directory	<code>PREFIX/bin</code>	5.5.7	
<code>INSTALL_DOCDIR</code> [151]	Documentation directory	<code>PREFIX/docs</code>	5.5.7	
<code>INSTALL_DOCREADMEDIR</code> [151]	README file directory	<code>PREFIX</code>	5.5.7	
<code>INSTALL_INCLUDEDIR</code> [151]	Header file directory	<code>PREFIX/include</code>	5.5.7	
<code>INSTALL_INFODIR</code> [151]	Info file directory	<code>PREFIX/docs</code>	5.5.7	
<code>INSTALL_LAYOUT</code> [151]	Select predefined installation layout	<code>STANDALONE</code>	5.5.7	
<code>INSTALL_LIBDIR</code> [152]	Library file directory	<code>PREFIX/lib</code>	5.5.7	
<code>INSTALL_MANDIR</code> [152]	Manual page directory	<code>PREFIX/man</code>	5.5.7	
<code>INSTALL_MYSQLSHAREDIR</code> [152]	Shared data directory	<code>PREFIX/share</code>	5.5.7	
<code>INSTALL_MYSQLTESTDIR</code> [152]	mysql-test directory	<code>PREFIX/mysql-test</code>	5.5.7	
<code>INSTALL_PLUGINDIR</code> [152]	Plugin directory	<code>PREFIX/lib/plugin</code>	5.5.7	

Formats	Description	Default	Introduced	Removed
INSTALL_SBINDIR [152]	Server executable directory	<code>PREFIX/bin</code>	5.5.7	
INSTALL_SCRIPTDIR [152]	Scripts directory	<code>PREFIX/scripts</code>	5.5.7	
INSTALL_SHAREDIR [152]	aclocal/mysql.m4 installation directory	<code>PREFIX/share</code>	5.5.7	
INSTALL_SQLBENCHDIR [152]	sql-bench directory	<code>PREFIX</code>	5.5.7	
INSTALL_SUPPORTFILES [152]	Extra support files directory	<code>PREFIX/support-files</code>	5.5.7	
MEMCACHED_HOME [156]	Path to memcached	<code>[none]</code>	5.5.16-ndb-7.2.2	
MYSQL_DATADIR [152]	Data directory		5.5.7	
MYSQL_MAINTAINER_MODE [155]	Whether to enable MySQL maintainer-specific development environment	<code>OFF</code>	5.5.7	
MYSQL_PROJECT_NAME [155]	Windows/Mac OS X project name	<code>3306</code>	5.5.21	
MYSQL_TCP_PORT [155]	TCP/IP port number	<code>3306</code>	5.5.7	
MYSQL_UNIX_ADDR [155]	Unix socket file	<code>/tmp/mysql.sock</code>	5.5.7	
ODBC_INCLUDES [153]	ODBC includes directory			
ODBC_LIB_DIR [153]	ODBC library directory			
SYSCONFDIR [153]	Option file directory		5.5.7	
WITH_BUNDLED_LIBEVENT [157]	Use bundled libevent when building ndbmemcache	<code>ON</code>	5.5.16-ndb-7.2.2	
WITH_BUNDLED_MEMCACHED [157]	Use bundled memcached when building ndbmemcache	<code>ON</code>	5.5.16-ndb-7.2.2	
WITH_CLASSPATH [157]	Classpath to use when building MySQL Cluster Connector for Java. Default is an empty string.			
WITH_DEBUG [155]	Whether to include debugging support	<code>OFF</code>	5.5.7	
WITH_EMBEDDED_SERVER [155]	Whether to build embedded server	<code>OFF</code>	5.5.7	
WITH_xxx_STORAGE_ENGINE [155]	Compile storage engine xxx statically into server		5.5.7	
WITH_ERROR_INSERT [157]	Enable error injection in the NDBCLUSTER storage engine. Should not be used for building binaries intended for production.	<code>OFF</code>		
WITH_EXTRA_CHARSETS [155]	Which extra character sets to include	<code>all</code>	5.5.7	
WITH_LIBWRAP [155]	Whether to include libwrap (TCP wrappers) support	<code>OFF</code>	5.5.7	

Formats	Description	Default	Introduced	Removed
WITH_NDB_BINLOG [157]	Enable binary logging by default by mysqld.	ON		
WITH_NDB_DEBUG [157]	Produce a debug build for testing or troubleshooting.	OFF		
WITH_NDB_JAVA [157]	Enable building of Java and ClusterJ support. Enabled by default. Supported in MySQL Cluster only.	ON	5.5.27-ndb-7.2.9	
WITH_NDB_PORT [158]	Default port used by a management server built with this option. If this option was not used to build it, the management server's default port is 1186.	[none]		
WITH_NDB_TEST [158]	Include NDB API test programs.	OFF		
WITH_NDBCLUSTER_STORAGE_ENGINE [157]	Build the NDB storage engine	ON		
WITH_NDBMTD [157]	Build multi-threaded data node.	ON		
WITH_READLINE [155]	Use bundled readline	OFF	5.5.7	
WITH_SSL [156]	Type of SSL support	no	5.5.7	
WITH_UNIXODBC [153]	Enable unixODBC support	OFF		
WITH_ZLIB [156]	Type of zlib support	system	5.5.7	
WITHOUT_xxx_STORAGE_ENGINE [157]	Exclude storage engine xxx from build		5.5.7	

The following sections provide more information about CMake options.

- [General Options](#)
- [Installation Layout Options](#)
- [Feature Options](#)
- [Options for Compiling MySQL Cluster](#)
- [Compiler Flags](#)

For boolean options, the value may be specified as 1 or `ON` to enable the option, or as 0 or `OFF` to disable the option.

Many options configure compile-time defaults that can be overridden at server startup. For example, the `CMAKE_INSTALL_PREFIX [151]`, `MYSQL_TCP_PORT [155]`, and `MYSQL_UNIX_ADDR [155]` options that configure the default installation base directory location, TCP/IP port number, and Unix socket file can be changed at server startup with the `--basedir [435]`, `--port [458]`, and `--socket [464]` options for `mysqld`. Where applicable, configuration option descriptions indicate the corresponding `mysqld` startup option.

General Options

- `-DBUILD_CONFIG=mysql_release`

This option configures a source distribution with the same build options used by Oracle to produce binary distributions for official MySQL releases.

- `-DCMAKE_BUILD_TYPE=type`

The type of build to produce:

- `RelWithDebInfo`: Enable optimizations and generate debugging information. This is the default MySQL build type.
- `Debug`: Disable optimizations and generate debugging information. This build type is also used if the `WITH_DEBUG [155]` option is enabled. That is, `-DWITH_DEBUG=1 [155]` has the same effect as `-DCMAKE_BUILD_TYPE=Debug [151]`.
- `-DCPACK_MONOLITHIC_INSTALL=bool`

This option affects whether the `make package` operation produces multiple installation package files or a single file. If disabled, the operation produces multiple installation package files, which may be useful if you want to install only a subset of a full MySQL installation. If enabled, it produces a single file for installing everything.

Installation Layout Options

The `CMAKE_INSTALL_PREFIX [151]` option indicates the base installation directory. Other options with names of the form `INSTALL_xxx` that indicate component locations are interpreted relative to the prefix and their values are relative pathnames. Their values should not include the prefix.

- `-DCMAKE_INSTALL_PREFIX=dir_name`

The installation base directory.

This value can be set at server startup with the `--basedir [435]` option.

- `-DINSTALL_BINDIR=dir_name`

Where to install user programs.

- `-DINSTALL_DOCDIR=dir_name`

Where to install documentation.

- `-DINSTALL_DOCREADMEDIR=dir_name`

Where to install `README` files.

- `-DINSTALL_INCLUDEDIR=dir_name`

Where to install header files.

- `-DINSTALL_INFODIR=dir_name`

Where to install Info files.

- `-DINSTALL_LAYOUT=name`

Select a predefined installation layout:

- `STANDALONE`: Same layout as used for `.tar.gz` and `.zip` packages. This is the default.

- **RPM**: Layout similar to RPM packages.
- **SVR4**: Solaris package layout.
- **DEB**: DEB package layout (experimental).

You can select a predefined layout but modify individual component installation locations by specifying other options. For example:

```
shell> cmake . -DINSTALL_LAYOUT=SVR4 -DMYSQL_DATADIR=/var/mysql/data
```

- `-DINSTALL_LIBDIR=dir_name`

Where to install library files.

- `-DINSTALL_MANDIR=dir_name`

Where to install manual pages.

- `-DINSTALL_MYSQLSHAREDIR=dir_name`

Where to install shared data files.

- `-DINSTALL_MYSQLTESTDIR=dir_name`

Where to install the `mysql-test` directory.

- `-DINSTALL_PLUGINDIR=dir_name`

The location of the plugin directory.

This value can be set at server startup with the `--plugin_dir` [\[537\]](#) option.

- `-DINSTALL_SBINDIR=dir_name`

Where to install the `mysqld` server.

- `-DINSTALL_SCRIPTDIR=dir_name`

Where to install `mysql_install_db`.

- `-DINSTALL_SHAREDIR=dir_name`

Where to install `aclocal/mysql.m4`.

- `-DINSTALL_SQLBENCHDIR=dir_name`

Where to install the `sql-bench` directory. To not install this directory, use an empty value (`-DINSTALL_SQLBENCHDIR=` [\[152\]](#)).

- `-DINSTALL_SUPPORTFILESDIR=dir_name`

Where to install extra support files.

- `-DMYSQL_DATADIR=dir_name`

The location of the MySQL data directory.

This value can be set at server startup with the `--datadir` [\[439\]](#) option.

- `-DODBC_INCLUDES=dir_name`

The location of the ODBC includes directory, and may be used while configuring Connector/ODBC.

- `-DODBC_LIB_DIR=dir_name`

The location of the ODBC library directory, and may be used while configuring Connector/ODBC.

- `-DSYSCONFDIR=dir_name`

The default `my.cnf` option file directory.

This location cannot be set at server startup, but you can start the server with a given option file using the `--defaults-file=file_name` [248] option, where `file_name` is the full path name to the file.

- `-DWITH_UNIXODBC=1`

Enables unixODBC support, for Connector/ODBC.

Storage Engine Options

Storage engines are built as plugins. You can build a plugin as a static module (compiled into the server) or a dynamic module (built as a dynamic library that must be installed into the server using the `INSTALL PLUGIN` statement or the `--plugin-load` [457] option before it can be used). Some plugins might not support static or dynamic building.

The `MyISAM`, `MERGE`, `MEMORY`, and `CSV` engines are mandatory (always compiled into the server) and need not be installed explicitly.

To compile a storage engine statically into the server, use `-DWITH_engine_STORAGE_ENGINE=1`. Some permissible `engine` values are `ARCHIVE`, `BLACKHOLE`, `EXAMPLE`, `FEDERATED`, `INNOBASE` (InnoDB), `NDBCLUSTER` (NDB), `PARTITION` (partitioning support), and `PERFSHEMA` (Performance Schema).

Examples:

```
-DWITH_INNOBASE_STORAGE_ENGINE=1
-DWITH_ARCHIVE_STORAGE_ENGINE=1
-DWITH_BLACKHOLE_STORAGE_ENGINE=1
-DWITH_PERFSHEMA_STORAGE_ENGINE=1
```



Note

`WITH_NDBCLUSTER_STORAGE_ENGINE` [157] is supported only when building MySQL Cluster using the MySQL Cluster sources. It cannot be used to enable clustering support in other MySQL source trees or distributions. In MySQL Cluster NDB 7.2 source distributions, it is enabled by default. See [Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”](#), and [Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”](#), for more information.

To exclude a storage engine from the build, use `-DWITHOUT_engine_STORAGE_ENGINE=1`. Examples:

```
-DWITHOUT_EXAMPLE_STORAGE_ENGINE=1
-DWITHOUT_FEDERATED_STORAGE_ENGINE=1
-DWITHOUT_PARTITION_STORAGE_ENGINE=1
```

If neither `-DWITH_engine_STORAGE_ENGINE` nor `-DWITHOUT_engine_STORAGE_ENGINE` are specified for a given storage engine, the engine is built as a shared module, or excluded if it cannot be built as a shared module.

Feature Options

- `-DCOMPILATION_COMMENT=string`

A descriptive comment about the compilation environment.

- `-DDEFAULT_CHARSET=charset_name`

The server character set. By default, MySQL uses the `latin1` (cp1252 West European) character set.

`charset_name` may be one of `binary`, `armscii8`, `ascii`, `big5`, `cp1250`, `cp1251`, `cp1256`, `cp1257`, `cp850`, `cp852`, `cp866`, `cp932`, `dec8`, `eucjpms`, `euckr`, `gb2312`, `gbk`, `geostd8`, `greek`, `hebrew`, `hp8`, `keybcs2`, `koi8r`, `koi8u`, `latin1`, `latin2`, `latin5`, `latin7`, `macce`, `macroman`, `sjis`, `swe7`, `tis620`, `ucs2`, `ujis`, `utf8`, `utf8mb4`, `utf16`, `utf32`. The permissible character sets are listed in the `cmake/character_sets.cmake` file as the value of `CHARSETS_AVAILABLE`.

This value can be set at server startup with the `--character_set_server` [488] option.

- `-DDEFAULT_COLLATION=collation_name`

The server collation. By default, MySQL uses `latin1_swedish_ci`. Use the `SHOW COLLATION` statement to determine which collations are available for each character set.

This value can be set at server startup with the `--collation_server` [489] option.

- `-DENABLE_DEBUG_SYNC=bool`

Whether to compile the Debug Sync facility into the server. This facility is used for testing and debugging. This option is enabled by default, but has no effect unless MySQL is configured with debugging enabled. If debugging is enabled and you want to disable Debug Sync, use `-DENABLE_DEBUG_SYNC=0` [154].

When compiled in, Debug Sync is disabled by default at runtime. To enable it, start `mysqld` with the `--debug-sync-timeout=N` [440] option, where `N` is a timeout value greater than 0. (The default value is 0, which disables Debug Sync.) `N` becomes the default timeout for individual synchronization points.

For a description of the Debug Sync facility and how to use synchronization points, see [MySQL Internals: Test Synchronization](#).

- `-DENABLE_DOWNLOADS=bool`

Whether to download optional files. For example, with this option enabled, `CMake` downloads the Google Test distribution that is used by the test suite to run unit tests.

- `-DENABLE_DTRACE=bool`

Whether to include support for DTrace probes. For information about DTrace, see [Section 5.4, "Tracing mysqld Using DTrace"](#)

- `-DENABLE_GCOV=bool`

Whether to include gcov support (Linux only).

- `-DENABLED_LOCAL_INFILE=bool`

Whether to enable `LOCAL` capability in the client library for `LOAD DATA INFILE`.

This option controls client-side `LOCAL` capability, but the capability can be set on the server side at server startup with the `--local-infile` [513] option. See Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”.

- `-DENABLED_PROFILING=bool`

Whether to enable query profiling code (for the `SHOW PROFILE` and `SHOW PROFILES` statements).

- `-DMYSQL_MAINTAINER_MODE=bool`

Whether to enable a MySQL maintainer-specific development environment. If enabled, this option causes compiler warnings to become errors.

- `-DMYSQL_PROJECT_NAME=name`

For Windows or Mac OS X, the project name to incorporate into the project file name. This option was added in MySQL 5.5.21.

- `-DMYSQL_TCP_PORT=port_num`

The port number on which the server listens for TCP/IP connections. The default is 3306.

This value can be set at server startup with the `--port` [458] option.

- `-DMYSQL_UNIX_ADDR=file_name`

The Unix socket file path on which the server listens for socket connections. This must be an absolute path name. The default is `/tmp/mysql.sock`.

This value can be set at server startup with the `--socket` [464] option.

- `-DWITH_DEBUG=bool`

Whether to include debugging support.

Configuring MySQL with debugging support enables you to use the `--debug="d,parser_debug"` [439] option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

- `-DWITH_EMBEDDED_SERVER=bool`

Whether to build the `libmysqld` embedded server library.

- `-DWITH_EXTRA_CHARSETS=name`

Which extra character sets to include:

- `all`: All character sets. This is the default.
- `complex`: Complex character sets.
- `none`: No extra character sets.

- `-DWITH_LIBWRAP=bool`

Whether to include `libwrap` (TCP wrappers) support.

- `-DWITH_READLINE=bool`

Whether to use the `readline` library bundled with the distribution.

- `-DWITH_SSL=ssl_type`

The type of SSL support to include, if any:

- `no`: No SSL support. This is the default.
- `yes`: Use the system SSL library if present, else the library bundled with the distribution.
- `bundled`: Use the SSL library bundled with the distribution.
- `system`: Use the system SSL library.

For information about using SSL support, see [Section 6.3.8, “Using SSL for Secure Connections”](#).

- `-DWITH_ZLIB=zlib_type`

Some features require that the server be built with compression library support, such as the `COMPRESS()` [1185] and `UNCOMPRESS()` [1189] functions, and compression of the client/server protocol. The `WITH_ZLIB` [156] indicates the source of `zlib` support:

- `bundled`: Use the `zlib` library bundled with the distribution.
- `system`: Use the system `zlib` library. This is the default.

Compiler Flags

To specify compiler flags, set the `CFLAGS` and `CXXFLAGS` environment variables before running `CMake`. Example:

```
shell> CFLAGS=-DDISABLE_GRANT_OPTIONS
shell> CXXFLAGS=-DDISABLE_GRANT_OPTIONS
shell> export CFLAGS CXXFLAGS
shell> cmake [options]
```

The following flags control configuration features:

- `DISABLE_GRANT_OPTIONS`

If this flag is defined, it causes the `--bootstrap` [436], `--skip-grant-tables` [461], and `--init-file` [445] options for `mysqld` to be disabled.

- `HAVE_EMBEDDED_PRIVILEGE_CONTROL`

By default, authentication for connections to the embedded server is disabled. To enable connection authentication, define this flag.

CMake Options for Compiling MySQL Cluster

The following options are for use when building MySQL Cluster NDB 7.2 or later. These options are supported only with the MySQL Cluster NDB 7.2 and later MySQL Cluster sources; they are not supported when using sources from the MySQL 5.5 Server tree.

- `-DMEMCACHED_HOME=path`

Perform the build using the memcached (version 1.6 or later) installed in the system directory indicated by `path`. Files from this installation that are used in the build include the memcached

binary, header files, and libraries, as well as the `memcached_utilities` library and the header file `engine_testapp.h`.

You must leave this option unset when building `ndbmemcache` using the bundled memcached sources (`WITH_BUNDLED_MEMCACHED` [157] option); in other words, the bundled sources are used by default).

This option was added in MySQL Cluster NDB 7.2.2.

While additional CMake options—such as for SASL authorization and for providing `dtrace` support—are available for use when compiling `memcached` from external sources, these options are currently not enabled for the `memcached` sources bundled with MySQL Cluster.

- `-DWITH_BUNDLED_LIBEVENT={ON|OFF}`

Use the `libevent` included in the MySQL Cluster sources when building MySQL Cluster with `ndbmemcached` support (MySQL Cluster NDB 7.2.2 and later). Enabled by default. OFF causes the system's `libevent` to be used instead.

- `-DWITH_BUNDLED_MEMCACHED={ON|OFF}`

Build the `memcached` sources included in the MySQL Cluster source tree (MySQL Cluster NDB 7.2.3 and later), then use the resulting `memcached` server when building the `ndbmemcache` engine. In this case, `make install` places the `memcached` binary in the installation `bin` directory, and the `ndbmemcache` engine shared object file `ndb_engine.so` in the installation `lib` directory.

This option is ON by default.

- `-DWITH_CLASSPATH=path`

Sets the classpath for building MySQL Cluster Connector for Java. Default is empty. In MySQL Cluster NDB 7.2.9 and later, this option is ignored if `-DWITH_NDB_JAVA=OFF` [157] is used.

- `-DWITH_ERROR_INSERT={ON|OFF}`

Enables error injection in the NDB kernel. For testing only; not intended for use in building production binaries. Default is OFF.

- `-DWITH_NDBCLUSTER_STORAGE_ENGINE={ON|OFF}`

Build and link in support for the NDB (NDBCLUSTER) storage engine in `mysqld`. Default is ON.

- `-DWITH_NDBCLUSTER={ON|OFF}`

This is an alias for `WITH_NDBCLUSTER_STORAGE_ENGINE` [157].

- `-DWITH_NDBMTD={ON|OFF}`

Build the multithreaded data node executable `ndbmtd`. Default is ON.

- `-DWITH_NDB_BINLOG={ON|OFF}`

Enable binary logging by default in the `mysqld` built using this option. ON by default.

- `-DWITH_NDB_DEBUG={ON|OFF}`

Enable building the debug versions of the MySQL Cluster binaries. OFF by default.

- `-DWITH_NDB_JAVA={ON|OFF}`

Enable building MySQL Cluster with Java support, including `ClusterJ`.

This option was added in MySQL Cluster NDB 7.2.9, and is ON by default. If you do not wish to compile MySQL Cluster with Java support, you must disable it explicitly by specifying `-DWITH_NDB_JAVA=OFF` when running `CMake`. Otherwise, if Java cannot be found, configuration of the build fails.

- `-DWITH_NDB_PORT=port`

Causes the MySQL Cluster management server (`ndb_mgmd`) that is built to use this `port` by default. If this option is unset, the resulting management server tries to use port 1186 by default.

- `-DWITH_NDB_TEST={ON|OFF}`

If enabled, include a set of NDB API test programs. The default is OFF.

2.10.5. Dealing with Problems Compiling MySQL

The solution to many problems involves reconfiguring. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run these commands on Unix before re-running `CMake`:

```
shell> make clean
shell> rm CMakeCache.txt
```

Or, on Windows:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

If you build out of the source tree, remove and recreate your build directory before re-running `CMake`. For instructions on building outside of the source tree, see [How to Build MySQL Server with CMake](#).

On some systems, warnings may occur due to differences in system include files. The following list describes other problems that have been found to occur most often when compiling MySQL:

- To define flags to be used by your C or C++ compilers, specify them using the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

To see what flags you might need to specify, invoke `mysql_config` with the `--cflags` [\[394\]](#) option.

- If compilation fails, check whether the `MYSQL_MAINTAINER_MODE` [\[155\]](#) option is enabled. This mode causes compiler warnings to become errors, so disabling it may enable compilation to proceed.

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

```
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

Or:

```
make: file `Makefile' line 18: Must be a separator (:
```

Or:

```
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome `make` programs.

GNU `make` 3.75 is known to work.

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pregenerated copy. However, if you do need to re-create it, you might encounter this error:

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

This is a sign that your version of `yacc` is deficient. You probably need to install `bison` (the GNU version of `yacc`) and use that instead.

Versions of `bison` older than 1.75 may report this error:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

For information about acquiring or updating tools, see the system requirements in [Section 2.10, "Installing MySQL from Source"](#).

2.10.6. MySQL Configuration and Third-Party Tools

Third-party tools that need to determine the MySQL version from the MySQL source can read the `VERSION` file in the top-level source directory. The file lists the pieces of the version separately. For example, if the version is 5.5.8, the file looks like this:

```
MYSQL_VERSION_MAJOR=5
MYSQL_VERSION_MINOR=5
MYSQL_VERSION_PATCH=8
MYSQL_VERSION_EXTRA=
```

If the source is not for a General Availability (GA) release, the `MYSQL_VERSION_EXTRA` value will be nonempty. For example, the value for a Release Candidate release would look like this:

```
MYSQL_VERSION_EXTRA=rc
```

To construct a five-digit number from the version components, use this formula:

```
MYSQL_VERSION_MAJOR*10000 + MYSQL_VERSION_MINOR*100 + MYSQL_VERSION_PATCH
```

2.11. Postinstallation Setup and Testing

After installing MySQL, there are some issues that you should address. For example, on Unix, you should initialize the data directory and create the MySQL grant tables. On all platforms, an important security concern is that the initial accounts in the grant tables have no passwords. You should assign passwords to prevent unauthorized access to the MySQL server. Optionally, you can create time zone tables to enable recognition of named time zones.

The following sections include postinstallation procedures that are specific to Windows systems and to Unix systems. Another section, [Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”](#), applies to all platforms; it describes what to do if you have trouble getting the server to start. [Section 2.11.2, “Securing the Initial MySQL Accounts”](#), also applies to all platforms. You should follow its instructions to make sure that you have properly protected your MySQL accounts by assigning passwords to them.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in [Section 6.2, “The MySQL Access Privilege System”](#), and [Section 6.3, “MySQL User Account Management”](#).

2.11.1. Unix Postinstallation Procedures

After installing MySQL on Unix, you must initialize the grant tables, start the server, and make sure that the server works satisfactorily. You may also wish to arrange for the server to be started and stopped automatically when your system starts and stops. You should also assign passwords to the accounts in the grant tables.

On Unix, the grant tables are set up by the `mysql_install_db` program. For some installation methods, this program is run for you automatically if an existing database cannot be found.

- If you install MySQL on Linux using RPM distributions, the server RPM runs `mysql_install_db`.
- Using the native packaging system on many platforms, including Debian Linux, Ubuntu Linux, Gentoo Linux and others, the `mysql_install_db` command is run for you.
- If you install MySQL on Mac OS X using a PKG distribution, the installer runs `mysql_install_db`.

For other platforms and installation types, including generic binary and source installs, you will need to run `mysql_install_db` yourself.

The following procedure describes how to initialize the grant tables (if that has not previously been done) and start the server. It also suggests some commands that you can use to test whether the server is accessible and working properly. For information about starting and stopping the server automatically, see [Section 2.11.1.2, “Starting and Stopping MySQL Automatically”](#).

After you complete the procedure and have the server running, you should assign passwords to the accounts created by `mysql_install_db` and perhaps restrict access to test databases. For instructions, see [Section 2.11.2, “Securing the Initial MySQL Accounts”](#).

In the examples shown here, the server runs under the user ID of the `mysql` login account. This assumes that such an account exists. Either create the account if it does not exist, or substitute the name of a different existing login account that you plan to use for running the server. For information about creating the account, see [Creating a mysql System User and Group](#), in [Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#).

1. Change location into the top-level directory of your MySQL installation, represented here by `BASEDIR`:

```
shell> cd BASEDIR
```

`BASEDIR` is the installation directory for your MySQL instance. It is likely to be something like `/usr/local/mysql` or `/usr/local`. The following steps assume that you have changed location to this directory.

You will find several files and subdirectories in the `BASEDIR` directory. The most important for installation purposes are the `bin` and `scripts` subdirectories:

- The `bin` directory contains client programs and the server. You should add the full path name of this directory to your `PATH` environment variable so that your shell finds the MySQL programs properly. See [Section 2.13, “Environment Variables”](#).
 - The `scripts` directory contains the `mysql_install_db` script used to initialize the `mysql` database containing the grant tables that store the server access permissions.
2. If necessary, ensure that the distribution contents are accessible to `mysql`. If you installed the distribution as `mysql`, no further action is required. If you installed the distribution as `root`, its contents will be owned by `root`. Change its ownership to `mysql` by executing the following commands as `root` in the installation directory. The first command changes the owner attribute of the files to the `mysql` user. The second changes the group attribute to the `mysql` group.

```
shell> chown -R mysql .
shell> chgrp -R mysql .
```

3. If necessary, run the `mysql_install_db` program to set up the initial MySQL grant tables containing the privileges that determine how users are permitted to connect to the server. You will need to do this if you used a distribution type for which the installation procedure does not run the program for you.

```
shell> scripts/mysql_install_db --user=mysql
```

Typically, `mysql_install_db` needs to be run only the first time you install MySQL, so you can skip this step if you are upgrading an existing installation. However, `mysql_install_db` does not overwrite any existing privilege tables, so it should be safe to run in any circumstances.

It might be necessary to specify other options such as `--basedir` [\[266\]](#) or `--datadir` [\[266\]](#) if `mysql_install_db` does not identify the correct locations for the installation directory or data directory. For example:

```
shell> scripts/mysql_install_db --user=mysql \
    --basedir=/opt/mysql/mysql \
    --datadir=/opt/mysql/mysql/data
```

The `mysql_install_db` script creates the server's data directory with `mysql` as the owner. Under the data directory, it creates directories for the `mysql` database that holds the grant tables and the `test` database that you can use to test MySQL. The script also creates privilege table entries for `root` and anonymous-user accounts. The accounts have no passwords initially. [Section 2.11.2, “Securing the Initial MySQL Accounts”](#), describes the initial privileges. Briefly, these privileges permit the MySQL `root` user to do anything, and permit anybody to create or use databases with a name of `test` or starting with `test_`. See [Section 6.2, “The MySQL Access Privilege System”](#), for a complete listing and description of the grant tables.

It is important to make sure that the database directories and files are owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this if you run `mysql_install_db` as `root`, include the `--user` [\[267\]](#) option as shown. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` [\[267\]](#) option from the command.

If you do not want to have the `test` database, you can remove it after starting the server, using the instructions in [Section 2.11.2, “Securing the Initial MySQL Accounts”](#).

If you have trouble with `mysql_install_db` at this point, see [Section 2.11.1.1, “Problems Running `mysql_install_db`”](#).

4. Most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory:

```
shell> chown -R root .
shell> chown -R mysql data
```

5. If the plugin directory (the directory named by the `plugin_dir` [537] system variable) is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` [537] read only to the server or by setting `--secure-file-priv` [460] to a directory where `SELECT` writes can be made safely.
6. If you installed MySQL using a source distribution, you may want to optionally copy one of the provided configuration files from the `support-files` directory into your `/etc` directory. There are different sample configuration files for different use cases, server types, and CPU and RAM configurations. If you want to use one of these standard files, you should copy it to `/etc/my.cnf`, or `/etc/mysql/my.cnf` and edit and check the configuration before starting your MySQL server for the first time.

If you do not copy one of the standard configuration files, the MySQL server will be started with the default settings.

If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `mysql.server` script itself, and in [Section 2.11.1.2, “Starting and Stopping MySQL Automatically”](#).

7. Start the MySQL server:

```
shell> bin/mysqld_safe --user=mysql &
```

It is important that the MySQL server be run using an unprivileged (non-`root`) login account. To ensure this if you run `mysqld_safe` as `root`, include the `--user` [258] option as shown. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` [258] option from the command.

For further instructions for running MySQL as an unprivileged user, see [Section 6.1.5, “How to Run MySQL as a Normal User”](#).

If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the `host_name.err` file in the data directory).

If you neglected to create the grant tables by running `mysql_install_db` before proceeding to this step, the following message appears in the error log file when you start the server:

```
mysqld: Can't find file: 'host.frm'
```

This error also occurs if you run `mysql_install_db` as `root` without the `--user` [267] option. Remove the `data` directory and run `mysql_install_db` with the `--user` [267] option as described previously.

If you have other problems starting the server, see [Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”](#). For more information about `mysqld_safe`, see [Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”](#).

- Use `mysqladmin` to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

The output from `mysqladmin version` varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell> bin/mysqladmin version
mysqladmin Ver 14.12 Distrib 5.5.31, for pc-linux-gnu on i686
...

Server version          5.5.31
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 14 days 5 hours 5 min 21 sec

Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
Queries per second avg: 0.000
```

To see what else you can do with `mysqladmin`, invoke it with the `--help` [\[297\]](#) option.

- Verify that you can shut down the server:

```
shell> bin/mysqladmin -u root shutdown
```

- Verify that you can start the server again. Do this by using `mysqld_safe` or by invoking `mysqld` directly. For example:

```
shell> bin/mysqld_safe --user=mysql &
```

If `mysqld_safe` fails, see [Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”](#).

- Run some simple tests to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```
shell> bin/mysqlshow
+-----+
|      Databases      |
+-----+
| information_schema |
| mysql              |
| test               |
+-----+

shell> bin/mysqlshow mysql
Database: mysql
+-----+
|      Tables        |
+-----+
| columns_priv      |
| db                |
| event             |
| func              |
| help_category     |
| help_keyword      |
```

```

| help_relation
| help_topic
| host
| plugin
| proc
| procs_priv
| servers
| tables_priv
| time_zone
| time_zone_leap_second
| time_zone_name
| time_zone_transition
| time_zone_transition_type
| user
+-----+
shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
+-----+-----+-----+
| host | db | user |
+-----+-----+-----+
| % | test | |
| % | test_% | |
+-----+-----+-----+
    
```

12. There is a benchmark suite in the `sql-bench` directory (under the MySQL installation directory) that you can use to compare how MySQL performs on different platforms. The benchmark suite is written in Perl. It requires the Perl DBI module that provides a database-independent interface to the various databases, and some other additional Perl modules:

```

DBI
DBD:mysql
Data:Dumper
Data:ShowTable
    
```

These modules can be obtained from CPAN (<http://www.cpan.org/>). See also [Section 2.14.1, "Installing Perl on Unix"](#).

The `sql-bench/Results` directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

```

shell> cd sql-bench
shell> perl run-all-tests
    
```

If you do not have the `sql-bench` directory, you probably installed MySQL using RPM files other than the source RPM. (The source RPM includes the `sql-bench` benchmark directory.) In this case, you must first install the benchmark suite before you can use it. There are separate benchmark RPM files named `mysql-bench-VERSION.i386.rpm` that contain benchmark code and data.

If you have a source distribution, there are also tests in its `tests` subdirectory that you can run. For example, to run `auto_increment.tst`, execute this command from the top-level directory of your source distribution:

```

shell> mysql -vvf test < ./tests/auto_increment.tst
    
```

The expected result of the test can be found in the `./tests/auto_increment.res` file.

13. At this point, you should have the server running. However, none of the initial MySQL accounts have a password, and the server permits permissive access to test databases. To tighten security, follow the instructions in [Section 2.11.2, "Securing the Initial MySQL Accounts"](#).

The MySQL 5.5 installation procedure creates time zone tables in the `mysql` database but does not populate them. To do so, use the instructions in [Section 10.6, "MySQL Server Time Zone Support"](#).

To make it more convenient to invoke programs installed in the `bin` directory under the installation directory, you can add that directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. See [Section 4.2.4, “Setting Environment Variables”](#).

You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD: :mysql` Perl modules. See [Section 4.6.13, “mysql_setpermission — Interactively Set Permissions in Grant Tables”](#). For Perl module installation instructions, see [Section 2.14, “Perl Installation Notes”](#).

If you would like to use `mysqlaccess` and have the MySQL distribution in some nonstandard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `bin/mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL = '/usr/local/bin/mysql'; # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a `Broken pipe` error will occur when you run `mysqlaccess`.

2.11.1.1. Problems Running `mysql_install_db`

The purpose of the `mysql_install_db` script is to generate new MySQL privilege tables. It does not overwrite existing MySQL privilege tables, and it does not affect any other data.

If you want to re-create your privilege tables, first stop the `mysqld` server if it is running. Then rename the `mysql` directory under the data directory to save it, and then run `mysql_install_db`. Suppose that your current directory is the MySQL installation directory and that `mysql_install_db` is located in the `bin` directory and the data directory is named `data`. To rename the `mysql` database and re-run `mysql_install_db`, use these commands.

```
shell> mv data/mysql data/mysql.old
shell> scripts/mysql_install_db --user=mysql
```

When you run `mysql_install_db`, you might encounter the following problems:

- **`mysql_install_db` fails to install the grant tables**

You may find that `mysql_install_db` fails to install the grant tables and terminates after displaying the following messages:

```
Starting mysqld daemon with databases from XXXXXX
mysqld ended
```

In this case, you should examine the error log file very carefully. The log should be located in the directory `XXXXXX` named by the error message and should indicate why `mysqld` did not start. If you do not understand what happened, include the log when you post a bug report. See [Section 1.7, “How to Report Bugs or Problems”](#).

- **There is a `mysqld` process running**

This indicates that the server is running, in which case the grant tables have probably been created already. If so, there is no need to run `mysql_install_db` at all because it needs to be run only once (when you install MySQL the first time).

- **Installing a second `mysqld` server does not work when one server is running**

This can happen when you have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation, but you want to create a second

installation for testing purposes. Generally the problem that occurs when you try to run a second server is that it tries to use a network interface that is in use by the first server. In this case, you should see one of the following error messages:

```
Can't start server: Bind on TCP/IP port:
Address already in use
Can't start server: Bind on unix socket...
```

For instructions on setting up multiple servers, see [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#).

- **You do not have write access to the `/tmp` directory**

If you do not have write access to create temporary files or a Unix socket file in the default location (the `/tmp` directory) or the `TMP_DIR` environment variable, if it has been set, an error occurs when you run `mysql_install_db` or the `mysqld` server.

You can specify different locations for the temporary directory and Unix socket file by executing these commands prior to starting `mysql_install_db` or `mysqld`, where *some_tmp_dir* is the full path name to some directory for which you have write permission:

```
shell> TMPDIR=/some_tmp_dir/
shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
shell> export TMPDIR MYSQL_UNIX_PORT
```

Then you should be able to run `mysql_install_db` and start the server with these commands:

```
shell> scripts/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
```

If `mysql_install_db` is located in the `scripts` directory, modify the first command to `scripts/mysql_install_db`.

See [Section C.5.4.5, “How to Protect or Change the MySQL Unix Socket File”](#), and [Section 2.13, “Environment Variables”](#).

There are some alternatives to running the `mysql_install_db` script provided in the MySQL distribution:

- If you want the initial privileges to be different from the standard defaults, you can modify `mysql_install_db` before you run it. However, it is preferable to use `GRANT` and `REVOKE` to change the privileges *after* the grant tables have been set up. In other words, you can run `mysql_install_db`, and then use `mysql -u root mysql` to connect to the server as the MySQL `root` user so that you can issue the necessary `GRANT` and `REVOKE` statements.

If you want to install MySQL on several machines with the same privileges, you can put the `GRANT` and `REVOKE` statements in a file and execute the file as a script using `mysql` after running `mysql_install_db`. For example:

```
shell> scripts/mysql_install_db --user=mysql
shell> bin/mysql -u root < your_script_file
```

By doing this, you can avoid having to issue the statements manually on each machine.

- It is possible to re-create the grant tables completely after they have previously been created. You might want to do this if you are just learning how to use `GRANT` and `REVOKE` and have made so many modifications after running `mysql_install_db` that you want to wipe out the tables and start over.

To re-create the grant tables, remove all the `.frm`, `.MYI`, and `.MYD` files in the `mysql` database directory. Then run the `mysql_install_db` script again.

- You can start `mysqld` manually using the `--skip-grant-tables` [461] option and add the privilege information yourself using `mysql`:

```
shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
shell> bin/mysql mysql
```

From `mysql`, manually execute the SQL commands contained in `mysql_install_db`. Make sure that you run `mysqladmin flush-privileges` or `mysqladmin reload` afterward to tell the server to reload the grant tables.

Note that by not using `mysql_install_db`, you not only have to populate the grant tables manually, you also have to create them first.

2.11.1.2. Starting and Stopping MySQL Automatically

Generally, you start the `mysqld` server in one of these ways:

- Invoke `mysqld` directly. This works on any platform.
- Run the MySQL server as a Windows service. The service can be set to start the server automatically when Windows starts, or as a manual service that you start on request. For instructions, see [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).
- Invoke `mysqld_safe`, which tries to determine the proper options for `mysqld` and then runs it with those options. This script is used on Unix and Unix-like systems. See [Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”](#).
- Invoke `mysql.server`. This script is used primarily at system startup and shutdown on systems that use System V-style run directories (that is, `/etc/init.d` and run-level specific directories), where it usually is installed under the name `mysql`. The `mysql.server` script starts the server by invoking `mysqld_safe`. See [Section 4.3.3, “mysql.server — MySQL Server Startup Script”](#).
- On Mac OS X, install a separate MySQL Startup Item package to enable the automatic startup of MySQL on system startup. The Startup Item starts the server by invoking `mysql.server`. See [Section 2.4.3, “Installing the MySQL Startup Item”](#), for details. A MySQL Preference Pane also provides control for starting and stopping MySQL through the System Preferences, see [Section 2.4.4, “Installing and Using the MySQL Preference Pane”](#).
- Use the Solaris/OpenSolaris service management framework (SMF) system to initiate and control MySQL startup. For more information, see [Section 2.7.2, “Installing MySQL on OpenSolaris using IPS”](#).

The `mysqld_safe` and `mysql.server` scripts, Windows server, Solaris/OpenSolaris SMF, and the Mac OS X Startup Item (or MySQL Preference Pane) can be used to start the server manually, or automatically at system startup time. `mysql.server` and the Startup Item also can be used to stop the server.

To start or stop the server manually using the `mysql.server` script, invoke it with `start` or `stop` arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

Before `mysql.server` starts the server, it changes location to the MySQL installation directory, and then invokes `mysqld_safe`. If you want the server to run as some specific user, add an appropriate `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file, as shown later in this section. (It is possible that you will need to edit `mysql.server` if you've installed a binary distribution of MySQL in a nonstandard location. Modify it to change location into the proper directory before it runs `mysqld_safe`. If you do this, your modified version of `mysql.server` may be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.)

`mysql.server stop` stops the server by sending a signal to it. You can also stop the server manually by executing `mysqladmin shutdown`.

To start and stop MySQL automatically on your server, you need to add start and stop commands to the appropriate places in your `/etc/rc*` files.

If you use the Linux server RPM package (`MySQL-server-VERSION.rpm`), or a native Linux package installation, the `mysql.server` script may be installed in the `/etc/init.d` directory with the name `mysql`. See Section 2.5.1, “Installing MySQL from RPM Packages on Linux”, for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as `mysqld`.

If you install MySQL from a source distribution or using a binary distribution format that does not install `mysql.server` automatically, you can install it manually. The script can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree.

To install `mysql.server` manually, copy it to the `/etc/init.d` directory with the name `mysql`, and then make it executable. Do this by changing location into the appropriate directory where `mysql.server` is located and executing these commands:

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```



Note

Older Red Hat systems use the `/etc/rc.d/init.d` directory rather than `/etc/init.d`. Adjust the preceding commands accordingly. Alternatively, first create `/etc/init.d` as a symbolic link that points to `/etc/rc.d/init.d`:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use `chkconfig`:

```
shell> chkconfig --add mysql
```

On some Linux systems, the following command also seems to be necessary to fully enable the `mysql` script:

```
shell> chkconfig --level 345 mysql on
```

On FreeBSD, startup scripts generally should go in `/usr/local/etc/rc.d/`. The `rc(8)` manual page states that scripts in this directory are executed only if their basename matches the `*.sh` shell file name pattern. Any other files or directories present within the directory are silently ignored. In other words, on FreeBSD, you should install the `mysql.server` script as `/usr/local/etc/rc.d/mysql.server.sh` to enable automatic startup.

As an alternative to the preceding setup, some operating systems also use `/etc/rc.local` or `/etc/init.d/boot.local` to start additional services on startup. To start up MySQL using this method, you could append a command like the one following to the appropriate startup file:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

For other systems, consult your operating system documentation to see how to install startup scripts.

You can add options for `mysql.server` in a global `/etc/my.cnf` file. A typical `/etc/my.cnf` file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

The `mysql.server` script supports the following options: `basedir`, `datadir`, and `pid-file`. If specified, they *must* be placed in an option file, not on the command line. `mysql.server` supports only `start` and `stop` as command-line arguments.

The following table shows which option groups the server and each startup script read from option files.

Table 2.15. MySQL Startup scripts and supported server option groups

Script	Option Groups
<code>mysqld</code>	<code>[mysqld]</code> , <code>[server]</code> , <code>[mysqld-major_version]</code>
<code>mysqld_safe</code>	<code>[mysqld]</code> , <code>[server]</code> , <code>[mysqld_safe]</code>
<code>mysql.server</code>	<code>[mysqld]</code> , <code>[mysql.server]</code> , <code>[server]</code>

`[mysqld-major_version]` means that groups with names like `[mysqld-5.1]` and `[mysqld-5.5]` are read by servers having versions 5.1.x, 5.5.x, and so forth. This feature can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, `mysql.server` also reads the `[mysql_server]` group and `mysqld_safe` also reads the `[safe_mysqld]` group. However, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead when using MySQL 5.5.

For more information on MySQL configuration files and their structure and contents, see [Section 4.2.3.3, “Using Option Files”](#).

2.11.1.3. Starting and Troubleshooting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server on Unix. If you are using Windows, see [Section 2.3.8, “Troubleshooting a Microsoft Windows MySQL Server Installation”](#).

If you have problems starting the server, here are some things to try:

- Check the error log to see why the server does not start.
- Specify any special options needed by the storage engines you are using.
- Make sure that the server knows where to find the data directory.
- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must be set such that the server can read and modify them.
- Verify that the network interfaces the server wants to use are available.

Some storage engines have options that control their behavior. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables ([InnoDB](#), [NDB](#)), be sure that you have them configured the way you want before starting the server:

If you are using [InnoDB](#) tables, see [Section 14.3.2, “Configuring InnoDB”](#).

Storage engines will use default option values if you specify none, but it is recommended that you review the available options and specify explicit values for those for which the defaults are not appropriate for your installation.

When the `mysqld` server starts, it changes location to the data directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

The data directory location is hardwired in when the server is compiled. This is where the server looks for the data directory by default. If the data directory is located somewhere else on your system, the server will not work properly. You can determine what the default path settings are by invoking `mysqld` with the `--verbose` [467] and `--help` [434] options.

If the default locations do not match the MySQL installation layout on your system, you can override them by specifying options to `mysqld` or `mysqld_safe` on the command line or in an option file.

To specify the location of the data directory explicitly, use the `--datadir` [439] option. However, normally you can tell `mysqld` the location of the base directory under which MySQL is installed and it looks for the data directory there. You can do this with the `--basedir` [435] option.

To check the effect of specifying path options, invoke `mysqld` with those options followed by the `--verbose` [467] and `--help` [434] options. For example, if you change location into the directory where `mysqld` is installed and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

You can specify other options such as `--datadir` [439] as well, but `--verbose` [467] and `--help` [434] must be the last options.

Once you determine the path settings you want, start the server without `--verbose` [467] and `--help` [434].

If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

```
shell> mysqladmin variables
```

Or:

```
shell> mysqladmin -h host_name variables
```

`host_name` is the name of the MySQL server host.

If you get `Errcode 13` (which means `Permission denied`) when starting `mysqld`, this means that the privileges of the data directory or its contents do not permit server access. In this case, you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as `root`, but this raises security issues and should be avoided.

On Unix, change location into the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

```
shell> ls -la /usr/local/mysql/var
```

If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named `mysql`, use these commands:

```
shell> chown -R mysql /usr/local/mysql/var
```

```
shell> chgrp -R mysql /usr/local/mysql/var
```

If it possible that even with correct ownership, MySQL may fail to start up if there is other security software running on your system that manages application access to various parts of the file system. In this case, you may need to reconfigure that software to enable `mysqld` to access the directories it uses during normal operation.

If the server fails to start up correctly, check the error log. Log files are located in the data directory (typically `C:\Program Files\MySQL\MySQL Server 5.5\data` on Windows, `/usr/local/mysql/data` for a Unix binary distribution, and `/usr/local/var` for a Unix source distribution). Look in the data directory for files with names of the form `host_name.err` and `host_name.log`, where `host_name` is the name of your server host. Then examine the last few lines of these files. On Unix, you can use `tail` to display them:

```
shell> tail host_name.err
shell> tail host_name.log
```

The error log should contain information that indicates why the server could not start.

If either of the following errors occur, it means that some other program (perhaps another `mysqld` server) is using the TCP/IP port or Unix socket file that `mysqld` is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

Use `ps` to determine whether you have another `mysqld` server running. If so, shut down the server before starting `mysqld` again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#).)

If no other server is running, try to execute the command `telnet your_host_name tcp_ip_port_number`. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you do not get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that `mysqld` is trying to use. You will need to track down what program this is and disable it, or else tell `mysqld` to listen to a different port with the `--port [458]` option. In this case, you will also need to specify the port number for client programs when connecting to the server using TCP/IP.

Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to permit access to the port.

If the server starts but you cannot connect to it, you should make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1 localhost
```

If you cannot get `mysqld` to start, you can try to make a trace file to find the problem by using the `--debug [439]` option. See [MySQL Internals: Porting to Other Systems](#).

2.11.2. Securing the Initial MySQL Accounts

Part of the MySQL installation process is to set up the `mysql` database that contains the grant tables:

- Windows distributions contain preinitialized grant tables.
- On Unix, the `mysql_install_db` program populates the grant tables. Some installation methods run this program for you. Others require that you execute it manually. For details, see [Section 2.11.1, “Unix Postinstallation Procedures”](#).

The `mysql.user` grant table defines the initial MySQL user accounts and their access privileges:

- Some accounts have the user name `root`. These are superuser accounts that have all privileges and can do anything. The initial `root` account passwords are empty, so anyone can connect to the MySQL server as `root` *without a password* and be granted all privileges.
- On Windows, `root` accounts are created that permit connections from the local host only. Connections can be made by specifying the host name `localhost`, the IP address `127.0.0.1`, or the IPv6 address `::1`. If the user selects the **Enable root access from remote machines** option during installation, the Windows installer creates another `root` account that permits connections from any host.
- On Unix, each `root` account permits connections from the local host. Connections can be made by specifying the host name `localhost`, the IP address `127.0.0.1`, the IPv6 address `::1`, or the actual host name or IP address.

An attempt to connect to the host `127.0.0.1` normally resolves to the `localhost` account. However, this fails if the server is run with the `--skip-name-resolve` [461] option, so the `127.0.0.1` account is useful in that case. The `::1` account is used for IPv6 connections.

- Some accounts are for anonymous users. These have an empty user name. The anonymous accounts have no password, so anyone can use them to connect to the MySQL server.
- On Windows, there is one anonymous account that permits connections from the local host. Connections can be made by specifying a host name of `localhost`.
- On Unix, each anonymous account permits connections from the local host. Connections can be made by specifying a host name of `localhost` for one of the accounts, or the actual host name or IP address for the other.

To display which accounts exist in the `mysql.user` table and check whether their passwords are empty, use the following statement:

```
mysql> SELECT User, Host, Password FROM mysql.user;
```

User	Host	Password
root	localhost	
root	myhost.example.com	
root	127.0.0.1	
root	::1	
	localhost	
	myhost.example.com	

This output indicates that there are several `root` and anonymous-user accounts, none of which have passwords. The output might differ on your system, but the presence of accounts with empty passwords means that your MySQL installation is unprotected until you do something about it:

- You should assign a password to each MySQL `root` account.
- If you want to prevent clients from connecting as anonymous users without a password, you should either assign a password to each anonymous account or else remove the accounts.

In addition, the `mysql.db` table contains rows that permit all accounts to access the `test` database and other databases with names that start with `test_`. This is true even for accounts that otherwise have no special privileges such as the default anonymous accounts. This is convenient for testing but inadvisable on production servers. Administrators who want database access restricted only to accounts that have permissions granted explicitly for that purpose should remove these `mysql.db` table rows.

The following instructions describe how to set up passwords for the initial MySQL accounts, first for the `root` accounts, then for the anonymous accounts. The instructions also cover how to remove the anonymous accounts, should you prefer not to permit anonymous access at all, and describe how to remove permissive access to test databases. Replace `newpwd` in the examples with the password that you want to use. Replace `host_name` with the name of the server host. You can determine this name from the output of the preceding `SELECT` statement. For the output shown, `host_name` is `myhost.example.com`.



Note

For additional information about setting passwords, see [Section 6.3.5, “Assigning Account Passwords”](#). If you forget your `root` password after setting it, see [Section C.5.4.1, “How to Reset the Root Password”](#).

You might want to defer setting the passwords until later, to avoid the need to specify them while you perform additional setup or testing. However, be sure to set them before using your installation for production purposes.

To set up additional accounts, see [Section 6.3.2, “Adding User Accounts”](#).

Assigning `root` Account Passwords

The `root` account passwords can be set several ways. The following discussion demonstrates three methods:

- Use the `SET PASSWORD` statement
- Use the `UPDATE` statement
- Use the `mysqladmin` command-line client program

To assign passwords using `SET PASSWORD`, connect to the server as `root` and issue a `SET PASSWORD` statement for each `root` account listed in the `mysql.user` table. Be sure to encrypt the password using the `PASSWORD()` [1188] function.

For Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@':::1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('newpwd');
```

The last statement is unnecessary if the `mysql.user` table has no `root` account with a host value of `%`.

For Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@':::1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

You can also use a single statement that assigns a password to all `root` accounts by using `UPDATE` to modify the `mysql.user` table directly. This method works on any platform:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
-> WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

To assign passwords to the `root` accounts using `mysqladmin`, execute the following commands:

```
shell> mysqladmin -u root password "newpwd"
shell> mysqladmin -u root -h host_name password "newpwd"
```

Those commands apply both to Windows and to Unix. The double quotation marks around the password are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter.

The `mysqladmin` method of setting the `root` account passwords does not work for the `'root'@'127.0.0.1'` or `'root'@':::1'` account. Use the `SET PASSWORD` method shown earlier.

After the `root` passwords have been set, you must supply the appropriate password whenever you connect as `root` to the server. For example, to shut down the server with `mysqladmin`, use this command:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

Assigning Anonymous Account Passwords

The `mysql` commands in the following instructions include a `-p` option based on the assumption that you have set the `root` account passwords using the preceding instructions and must specify that password when connecting to the server.

To assign passwords to the anonymous accounts, connect to the server as `root`, then use either `SET PASSWORD` or `UPDATE`. Be sure to encrypt the password using the `PASSWORD()` [1188] function.

To use `SET PASSWORD` on Windows, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
```

To use `SET PASSWORD` on Unix, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('newpwd');
```

To set the anonymous-user account passwords with a single `UPDATE` statement, do this (on any platform):

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
-> WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

Removing Anonymous Accounts

If you prefer to remove any anonymous accounts rather than assigning them passwords, do so as follows on Windows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DROP USER '@'localhost';
```

On Unix, remove the anonymous accounts like this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DROP USER '@'localhost';
mysql> DROP USER '@'host_name';
```

Securing Test Databases

By default, the `mysql.db` table contains rows that permit access by any user to the `test` database and other databases with names that start with `test_`. (These rows have an empty `User` column value, which for access-checking purposes matches any user name.) This means that such databases can be used even by accounts that otherwise possess no privileges. If you want to remove any-user access to test databases, do so as follows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DELETE FROM mysql.db WHERE Db LIKE 'test%';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the privilege change remains unnoticed by the server until you restart it.

With the preceding change, only users who have global database privileges or privileges granted explicitly for the `test` database can use it. However, if you do not want the database to exist at all, drop it:

```
mysql> DROP DATABASE test;
```



Note

On Windows, you can also perform the process described in this section using the Configuration Wizard (see [Section 2.3.6.11, “The Security Options Dialog”](#)). On other platforms, the MySQL distribution includes `mysql_secure_installation`, a command-line utility that automates much of the process of securing a MySQL installation.

2.12. Upgrading or Downgrading MySQL

2.12.1. Upgrading MySQL

As a general rule, to upgrade from one release series to another, go to the next series rather than skipping a series. To upgrade from a release series previous to MySQL 5.1, upgrade to each successive release series in turn until you have reached MySQL 5.1, and then proceed with the upgrade to MySQL 5.5. For example, if you currently are running MySQL 5.0 and wish to upgrade to a newer series, upgrade to MySQL 5.1 first before upgrading to 5.5, and so forth. For information on upgrading to MySQL 5.1, see the *MySQL 5.1 Reference Manual*.

There is a special case for upgrading to MySQL 5.5, which is that there was a short-lived MySQL 5.4 development series. This series is no longer being worked on, but to accommodate users of both series, this section includes one subsection for users upgrading from MySQL 5.1 to 5.5 and another for users upgrading from MySQL 5.4 to 5.5.

To upgrade to MySQL 5.5, use the items in the following checklist as a guide:

- Before any upgrade, back up your databases, including the `mysql` database that contains the grant tables. See [Section 7.2, “Database Backup Methods”](#).
- Read *all* the notes in [Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”](#), or [Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”](#), depending on whether you currently use MySQL 5.1 or 5.4. These notes enable you to identify upgrade issues that apply to your current MySQL installation. Some incompatibilities discussed in that section require your attention *before* upgrading. Others should be dealt with *after* upgrading.
- Read the [Release Notes](#) as well, which provide information about features that are new in MySQL 5.5 or differ from those found in earlier MySQL releases.
- After upgrading to a new version of MySQL, run `mysql_upgrade` (see [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#)). This program checks your tables, and attempts to repair them if necessary. It also updates your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. (Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.)

`mysql_upgrade` does not upgrade the contents of the help tables. For upgrade instructions, see [Section 5.1.10, “Server-Side Help”](#).

- If you run MySQL Server on Windows, see [Section 2.3.9, “Upgrading MySQL on Windows”](#).
- If you use replication, see [Section 16.4.3, “Upgrading a Replication Setup”](#), for information on upgrading your replication setup.
- If you upgrade an installation originally produced by installing multiple RPM packages, it is best to upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.
- If you have created a user-defined function (UDF) with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. The same is true if the new version of MySQL implements a built-in function with the same name as an existing stored function. See [Section 9.2.4, “Function Name Parsing and Resolution”](#), for the rules describing how the server interprets references to different kinds of functions.

For upgrades between versions of a MySQL release series that has reached General Availability status, you can move the MySQL format files and data files between different versions on systems with the same architecture. For upgrades to a version of a MySQL release series that is in development status, that is not necessarily true. Use of development releases is at your own risk.

If you are cautious about using new versions, you can always rename your old `mysqld` before installing a newer one. For example, if you are using a version of MySQL 5.1 and want to upgrade to 5.5, rename your current server from `mysqld` to `mysqld-5.1`. If your new `mysqld` then does something unexpected, you can simply shut it down and restart with your old `mysqld`.

If problems occur, such as that the new `mysqld` server does not start or that you cannot connect without a password, verify that you do not have an old `my.cnf` file from your previous installation. You can check this with the `--print-defaults [248]` option (for example, `mysqld --print-defaults`). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.

If, after an upgrade, you experience problems with compiled client programs, such as [Commands out of sync](#) or unexpected core dumps, you probably have used old header or library files when compiling

your programs. In this case, you should check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries. Recompile might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example from `libmysqlclient.so.15` to `libmysqlclient.so.16`).

If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, you might find it useful to create a “dummy” database instance for assessing what conversions might be needed and the work involved to perform them. Make a copy of your MySQL instance that contains a full copy of the `mysql` database, plus all other databases without data. Run your upgrade procedure on this dummy instance to see what actions might be needed so that you can better evaluate the work involved when performing actual data conversion on your original database instance.

It is a good idea to rebuild and reinstall the Perl `DBD:mysql` module whenever you install a new release of MySQL. The same applies to other MySQL interfaces as well, such as PHP `mysql` extensions and the Python `MySQLdb` module.

2.12.1.1. Upgrading from MySQL 5.1 to 5.5



Note

It is good practice to back up your data before installing any new version of software. Although MySQL works very hard to ensure a high level of quality, you should protect your data by making a backup.

To upgrade to 5.5 from any previous version, MySQL recommends that you dump your tables with `mysqldump` before upgrading and reload the dump file after upgrading. Use the `--all-databases` [313] option to include all databases in the dump. If your databases include stored programs, use the `--routines` [320] and `--events` [316] options as well.

In general, you should do the following when upgrading from MySQL 5.1 to 5.5:

- Read *all* the items in these sections to see whether any of them might affect your applications:
 - [Section 2.12.1, “Upgrading MySQL”](#), has general update information.
 - The items in the change lists provided later in this section enable you to identify upgrade issues that apply to your current MySQL installation. Some incompatibilities discussed there require your attention *before* upgrading. Others should be dealt with *after* upgrading.
 - The MySQL 5.5 [Release Notes](#) describe significant new features you can use in 5.5 or that differ from those found in earlier MySQL releases. Some of these changes may result in incompatibilities.

Note particularly any changes that are marked **Known issue** or **Incompatible change**. These incompatibilities with earlier versions of MySQL may require your attention *before you upgrade*. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases. If any upgrade issue applicable to your installation involves an incompatibility that requires special handling, follow the instructions given in the incompatibility description. Sometimes this involves dumping and reloading tables, or use of a statement such as `CHECK TABLE` or `REPAIR TABLE`.

For dump and reload instructions, see [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#). Any procedure that involves `REPAIR TABLE` with the `USE_FRM` option *must* be done before upgrading. Use of this statement with a version of MySQL different from the one used to create the table (that is, using it after upgrading) may damage the table. See [Section 13.7.2.5, “REPAIR TABLE Syntax”](#).

- Before upgrading to a new version of MySQL, [Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”](#), to see whether changes to table formats or to character sets or collations were made between your current version of MySQL and the version to which you are upgrading. If so and these changes result in an incompatibility between MySQL versions, you will need to upgrade the affected tables using the instructions in [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#).
- After upgrading to a new version of MySQL, run `mysql_upgrade` (see [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#)). This program checks your tables, and attempts to repair them if necessary. It also updates your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. (Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.)

`mysql_upgrade` does not upgrade the contents of the help tables. For upgrade instructions, see [Section 5.1.10, “Server-Side Help”](#).

- If you run MySQL Server on Windows, see [Section 2.3.9, “Upgrading MySQL on Windows”](#).
- If you use replication, see [Section 16.4.3, “Upgrading a Replication Setup”](#), for information on upgrading your replication setup.

If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, you might find it useful to create a “dummy” database instance for assessing what conversions might be needed and the work involved to perform them. Make a copy of your MySQL instance that contains a full copy of the `mysql` database, plus all other databases without data. Run your upgrade procedure on this dummy instance to see what actions might be needed so that you can better evaluate the work involved when performing actual data conversion on your original database instance.

The following lists describe changes that may affect applications and that you should watch out for when upgrading from MySQL 5.1 to 5.5.

Configuration Changes

- **Incompatible change:** The `InnoDB Plugin` is included in MySQL 5.5 releases. It becomes the built-in version of `InnoDB` in MySQL Server, replacing the version previously included as the built-in `InnoDB` engine. `InnoDB Plugin` is also available in MySQL 5.1 as of 5.1.38, but it is an optional storage engine that must be enabled explicitly using two server options:

```
[mysqld]
ignore-builtin-innodb
plugin-load=innodb=ha_innodb_plugin.so
```

If you were using `InnoDB Plugin` in MySQL 5.1 by means of those options, you must remove them after an upgrade to 5.5 or the server will fail to start.

In addition, in `InnoDB Plugin`, the `innodb_file_io_threads` system variable has been removed and replaced with `innodb_read_io_threads` [1568] and `innodb_write_io_threads` [1575]. If you upgrade from MySQL 5.1 to MySQL 5.5 and previously explicitly set `innodb_file_io_threads` at server startup, you must change your configuration. Either remove any reference to `innodb_file_io_threads` or replace it with references to `innodb_read_io_threads` [1568] and `innodb_write_io_threads` [1575].

- **Incompatible change:** In MySQL 5.5, the server includes a plugin services interface that complements the plugin API. The services interface enables server functionality to be exposed as a “service” that plugins can access through a function-call interface. The `libmysqservices` library provides access to the available services and dynamic plugins now must be linked against this library (use the `-lmysqservices` flag). For an example showing how to configure for `CMake`, see [Section 23.2.5, “MySQL Services for Plugins”](#).

Server Changes

- **Incompatible change:** As of MySQL 5.5.3, due to work done for Bug #989, `FLUSH TABLES` is not permitted when there is an active `LOCK TABLES ... READ`. To provide a workaround for this restriction, `FLUSH TABLES` has a new variant, `FLUSH TABLES tbl_list WITH READ LOCK`, that enables tables to be flushed and locked in a single operation. As a result of this change, applications that previously used this statement sequence to lock and flush tables will fail:

```
LOCK TABLES tbl_list READ;
FLUSH TABLES tbl_list;
```

Such applications should now use this statement instead:

```
FLUSH TABLES tbl_list WITH READ LOCK;
```

- **Incompatible change:** As of MySQL 5.5.7, the server requires that a new grant table, `proxies_priv`, be present in the `mysql` database. If you are upgrading to 5.5.7 from a previous MySQL release rather than performing a new installation, the server will find that this table is missing and exit during startup with the following message:

```
Table 'mysql.proxies_priv' doesn't exist
```

To create the `proxies_priv` table, start the server with the `--skip-grant-tables` [461] option to cause it to skip the normal grant table checks, then run `mysql_upgrade`. For example:

```
shell> mysqld --skip-grant-tables &
shell> mysql_upgrade
```

Then stop the server and restart it normally.

You can specify other options on the `mysqld` command line if necessary. Alternatively, if your installation is configured so that the server normally reads options from an option file, use the `--defaults-file` [248] option to specify the file (enter each command on a single line):

```
shell> mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
--skip-grant-tables &
shell> mysql_upgrade
```

With the `--skip-grant-tables` [461] option, the server does no password or privilege checking, so any client can connect and effectively have all privileges. For additional security, use the `--skip-networking` [461] option as well to prevent remote clients from connecting.



Note

This problem is fixed in MySQL 5.5.8; the server treats a missing `proxies_priv` table as equivalent to an empty table. However, after starting the server, you should still run `mysql_upgrade` to create the table.

- **Incompatible change:** As of MySQL 5.5.7, `InnoDB` always uses the fast truncation technique, equivalent to `DROP TABLE` and `CREATE TABLE`. It no longer performs a row-by-row delete for tables with parent-child foreign key relationships. `TRUNCATE TABLE` returns an error for such tables. Modify your SQL to issue `DELETE FROM table_name` for such tables instead.
- **Incompatible change:** Prior to MySQL 5.5.7, if you flushed the logs using `FLUSH LOGS` or `mysqladmin flush-logs` and `mysqld` was writing the error log to a file (for example, if it was started with the `--log-error` [448] option), it renames the current log file with the suffix `-old`, then created a new empty log file. This had the problem that a second log-flushing operation thus caused the original error log file to be lost unless you saved it under a different name. For example, you could use the following commands to save the file:

```
shell> mysqladmin flush-logs
shell> mv host_name.err-old backup-directory
```

To avoid the preceding file-loss problem, no renaming occurs as of MySQL 5.5.7; the server merely closes and reopens the log file. To rename the file, you can do so manually before flushing. Then flushing the logs reopens a new file with the original file name. For example, you can rename the file and create a new one using the following commands:

```
shell> mv host_name.err host_name.err-old
shell> mysqladmin flush-logs
shell> mv host_name.err-old backup-directory
```

- **Incompatible change:** As of MySQL 5.5.6, handling of `CREATE TABLE IF NOT EXISTS ... SELECT` statements has been changed for the case that the destination table already exists:
 - Previously, for `CREATE TABLE IF NOT EXISTS ... SELECT`, MySQL produced a warning that the table exists, but inserted the rows and wrote the statement to the binary log anyway. By contrast, `CREATE TABLE ... SELECT` (without `IF NOT EXISTS`) failed with an error, but MySQL inserted no rows and did not write the statement to the binary log.
 - MySQL now handles both statements the same way when the destination table exists, in that neither statement inserts rows or is written to the binary log. The difference between them is that MySQL produces a warning when `IF NOT EXISTS` is present and an error when it is not.

This change in handling of `IF NOT EXISTS` results in an incompatibility for statement-based replication from a MySQL 5.1 master with the original behavior and a MySQL 5.5 slave with the new behavior. Suppose that `CREATE TABLE IF NOT EXISTS ... SELECT` is executed on the master and the destination table exists. The result is that rows are inserted on the master but not on the slave. (Row-based replication does not have this problem.)

To address this issue, statement-based binary logging for `CREATE TABLE IF NOT EXISTS ... SELECT` is changed in MySQL 5.1 as of 5.1.51:

- If the destination table does not exist, there is no change: The statement is logged as is.
- If the destination table does exist, the statement is logged as the equivalent pair of `CREATE TABLE IF NOT EXISTS` and `INSERT ... SELECT` statements. (If the `SELECT` in the original statement is preceded by `IGNORE` or `REPLACE`, the `INSERT` becomes `INSERT IGNORE` or `REPLACE`, respectively.)

This change provides forward compatibility for statement-based replication from MySQL 5.1 to 5.5 because when the destination table exists, the rows will be inserted on both the master and slave. To take advantage of this compatibility measure, the 5.1 server must be at least 5.1.51 and the 5.5 server must be at least 5.5.6.

To upgrade an existing 5.1-to-5.5 replication scenario, upgrade the master first to 5.1.51 or higher. Note that this differs from the usual replication upgrade advice of upgrading the slave first.

A workaround for applications that wish to achieve the original effect (rows inserted regardless of whether the destination table exists) is to use `CREATE TABLE IF NOT EXISTS` and `INSERT ... SELECT` statements rather than `CREATE TABLE IF NOT EXISTS ... SELECT` statements.

Along with the change just described, the following related change was made: Previously, if an existing view was named as the destination table for `CREATE TABLE IF NOT EXISTS ... SELECT`, rows were inserted into the underlying base table and the statement was written to the binary log. As of MySQL 5.1.51 and 5.5.6, nothing is inserted or logged.

- **Incompatible change:** Prior to MySQL 5.5.6, if the server was started with `character_set_server` set to `utf16`, it crashed during full-text stopwords initialization. Now the stopwords file is loaded and searched using `latin1` if `character_set_server` is `ucs2`, `utf16`, or `utf32`. If any table was created with `FULLTEXT` indexes while the server character set was `ucs2`, `utf16`, or `utf32`, it should be repaired using this statement:

```
REPAIR TABLE tbl_name QUICK;
```

- **Incompatible change:** As of MySQL 5.5.5, all numeric operators and functions on integer, floating-point and `DECIMAL` values throw an “out of range” error (`ER_DATA_OUT_OF_RANGE [4026]`) rather than returning an incorrect value or `NULL`, when the result is out of the supported range for the corresponding data type. See [Section 11.2.6, “Out-of-Range and Overflow Handling”](#).
- **Incompatible change:** In very old versions of MySQL (prior to 4.1), the `TIMESTAMP` data type supported a display width, which was silently ignored beginning with MySQL 4.1. This is deprecated in MySQL 5.1, and removed altogether in MySQL 5.5. These changes in behavior can lead to two problem scenarios when trying to use `TIMESTAMP(N)` columns with a MySQL 5.5 or later server:
 - When importing a dump file (for example, one created using `mysqldump`) created in a MySQL 5.0 or earlier server into a server from a newer release series, a `CREATE TABLE` or `ALTER TABLE` statement containing `TIMESTAMP(N)` causes the import to fail with a syntax error.

To fix this problem, edit the dump file in a text editor to replace any instances of `TIMESTAMP(N)` with `TIMESTAMP` prior to importing the file. Be sure to use a plain text editor for this, and not a word processor; otherwise, the result is almost certain to be unusable for importing into the MySQL server.

- When trying replicate any `CREATE TABLE` or `ALTER TABLE` statement containing `TIMESTAMP(N)` from a master MySQL server that supports the `TIMESTAMP(N)` syntax to a MySQL 5.5.3 or newer slave, the statement causes replication to fail. Similarly, when you try to restore from a binary log written by a server that supports `TIMESTAMP(N)` to a MySQL 5.5.3 or newer server, any `CREATE TABLE` or `ALTER TABLE` statement containing `TIMESTAMP(N)` causes the backup to fail. This holds true regardless of the logging format.

It may be possible to fix such issues using a hex editor, by replacing any width arguments used with `TIMESTAMP`, and the parentheses containing them, with space characters (hexadecimal `20`). Be sure to use a programmer's binary hex editor and not a regular text editor or word processor for this; otherwise, the result is almost certain to be a corrupted binary log file. To guard against accidental corruption of the binary log, you should always work on a copy of the file rather than the original.

You should try to handle potential issues of these types proactively by updating with `ALTER TABLE` any `TIMESTAMP(N)` columns in your databases so that they use `TIMESTAMP` instead, before performing any upgrades.

- **Incompatible change:** As of MySQL 5.5.3, the Unicode implementation has been extended to provide support for supplementary characters that lie outside the Basic Multilingual Plane (BMP). Noteworthy features:
 - `utf16` and `utf32` character sets have been added. These correspond to the UTF-16 and UTF-32 encodings of the Unicode character set, and they both support supplementary characters.
 - The `utf8mb4` character set has been added. This is similar to `utf8`, but its encoding allows up to four bytes per character to enable support for supplementary characters.
 - The `ucs2` character set is essentially unchanged except for the inclusion of some newer BMP characters.

In most respects, upgrading to MySQL 5.5 should present few problems with regard to Unicode usage, although there are some potential areas of incompatibility. These are the primary areas of concern:

- For the variable-length character data types ([VARCHAR](#) and the [TEXT](#) types), the maximum length in characters is less for [utf8mb4](#) columns than for [utf8](#) columns.
- For all character data types ([CHAR](#), [VARCHAR](#), and the [TEXT](#) types), the maximum number of characters that can be indexed is less for [utf8mb4](#) columns than for [utf8](#) columns.

Consequently, if you want to upgrade tables from [utf8](#) to [utf8mb4](#) to take advantage of supplementary-character support, it may be necessary to change some column or index definitions.

For additional details about the new Unicode character sets and potential incompatibilities, see [Section 10.1.10, “Unicode Support”](#), and [Section 10.1.11, “Upgrading from Previous to Current Unicode Support”](#).

- **Incompatible change:** As of MySQL 5.5.3, the server includes [dtoa](#), a library for conversion between strings and numbers by David M. Gay. In MySQL, this library provides the basis for improved conversion between string or [DECIMAL](#) values and approximate-value ([FLOAT](#) or [DOUBLE](#)) numbers.

Because the conversions produced by this library differ in some cases from previous results, the potential exists for incompatibilities in applications that rely on previous results. For example, applications that depend on a specific exact result from previous conversions might need adjustment to accommodate additional precision.

For additional information about the properties of [dtoa](#) conversions, see [Section 12.2, “Type Conversion in Expression Evaluation”](#).

- **Incompatible change:** In MySQL 5.5, several changes were made regarding the language and character set of error messages:
 - The [--language \[446\]](#) option for specifying the directory for the error message file is now deprecated. The new [lc_messages_dir \[512\]](#) and [lc_messages \[512\]](#) system variables should be used instead, and the server treats [--language \[446\]](#) as an alias for [lc_messages_dir \[512\]](#).
 - The [language \[510\]](#) system variable has been removed and replaced with the new [lc_messages_dir \[512\]](#) and [lc_messages \[512\]](#) system variables. [lc_messages_dir \[512\]](#) has only a global value and is read only. [lc_messages \[512\]](#) has global and session values and can be modified at runtime, so the error message language can be changed while the server is running, and individual clients each can have a different error message language by changing their session [lc_messages \[512\]](#) value to a different locale name.
 - Error messages previously were constructed in a mix of character sets. This issue is resolved by constructing error messages internally within the server using UTF-8 and returning them to the client in the character set specified by the [character_set_results \[488\]](#) system variable. The content of error messages therefore may in some cases differ from the messages returned previously.

For more information, see [Section 10.2, “Setting the Error Message Language”](#), and [Section 10.1.6, “Character Set for Error Messages”](#).

SQL Changes

- **Incompatible change:** Previously, the parser accepted an [INTO](#) clause in nested [SELECT](#) statements, which is invalid because such statements must return their results to the outer context. As of MySQL 5.5.3, this syntax is no longer permitted and statements that use it must be changed.

- **Incompatible change:** In MySQL 5.5.3, several changes were made to alias resolution in multiple-table `DELETE` statements so that it is no longer possible to have inconsistent or ambiguous table aliases.
- In MySQL 5.1.23, alias declarations outside the `table_references` part of the statement were disallowed for the `USING` variant of multiple-table `DELETE` syntax, to reduce the possibility of ambiguous aliases that could lead to ambiguous statements that have unexpected results such as deleting rows from the wrong table.

As of MySQL 5.5.3, alias declarations outside `table_references` are disallowed for all multiple-table `DELETE` statements. Alias declarations are permitted only in the `table_references` part.

Incorrect:

```
DELETE FROM t1 AS a2 USING t1 AS a1 INNER JOIN t2 AS a2;
DELETE t1 AS a2 FROM t1 AS a1 INNER JOIN t2 AS a2;
```

Correct:

```
DELETE FROM t1 USING t1 AS a1 INNER JOIN t2 AS a2;
DELETE t1 FROM t1 AS a1 INNER JOIN t2 AS a2;
```

- Previously, for alias references in the list of tables from which to delete rows in a multiple-table delete, the default database is used unless one is specified explicitly. For example, if the default database is `db1`, the following statement does not work because the unqualified alias reference `a2` is interpreted as having a database of `db1`:

```
DELETE a1, a2 FROM db1.t1 AS a1 INNER JOIN db2.t2 AS a2
WHERE a1.id=a2.id;
```

To correctly match an alias that refers to a table outside the default database, you must explicitly qualify the reference with the name of the proper database:

```
DELETE a1, db2.a2 FROM db1.t1 AS a1 INNER JOIN db2.t2 AS a2
WHERE a1.id=a2.id;
```

As of MySQL 5.5.3, alias resolution does not require qualification and alias references should not be qualified with the database name. Qualified names are interpreted as referring to tables, not aliases.

Statements containing alias constructs that are no longer permitted must be rewritten.

- Some keywords may be reserved in MySQL 5.5 that were not reserved in MySQL 5.1. See [Section 9.3](#), “Reserved Words”.

2.12.1.2. Upgrading from MySQL 5.4 to 5.5

This section is for the special case of upgrading to MySQL 5.5 from the short-lived MySQL 5.4 development series, which is no longer being worked on.



Note

It is good practice to back up your data before installing any new version of software. Although MySQL works very hard to ensure a high level of quality, you should protect your data by making a backup.

To upgrade to 5.5 from any previous version, MySQL recommends that you dump your tables with `mysqldump` before upgrading and reload the dump file after upgrading. Use the `--all-databases` [313] option to include all databases in the dump. If your databases include stored programs, use the `--routines` [320] and `--events` [316] options as well.

In general, you should do the following when upgrading from MySQL 5.4 to 5.5:

- Read *all* the items in these sections to see whether any of them might affect your applications:
 - [Section 2.12.1, “Upgrading MySQL”](#), has general update information.
 - The items in the change lists provided later in this section enable you to identify upgrade issues that apply to your current MySQL installation. Some incompatibilities discussed there require your attention *before* upgrading. Others should be dealt with *after* upgrading.
 - The MySQL 5.5 [Release Notes](#) describe significant new features you can use in 5.5 or that differ from those found in earlier MySQL releases. Some of these changes may result in incompatibilities.

Note particularly any changes that are marked **Known issue** or **Incompatible change**. These incompatibilities with earlier versions of MySQL may require your attention *before you upgrade*. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases. If any upgrade issue applicable to your installation involves an incompatibility that requires special handling, follow the instructions given in the incompatibility description. Sometimes this involves dumping and reloading tables, or use of a statement such as `CHECK TABLE` or `REPAIR TABLE`.

For dump and reload instructions, see [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#). Any procedure that involves `REPAIR TABLE` with the `USE_FRM` option *must* be done before upgrading. Use of this statement with a version of MySQL different from the one used to create the table (that is, using it after upgrading) may damage the table. See [Section 13.7.2.5, “REPAIR TABLE Syntax”](#).

- Before upgrading to a new version of MySQL, [Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”](#), to see whether changes to table formats or to character sets or collations were made between your current version of MySQL and the version to which you are upgrading. If so and these changes result in an incompatibility between MySQL versions, you will need to upgrade the affected tables using the instructions in [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#).
- After upgrading to a new version of MySQL, run `mysql_upgrade` (see [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#)). This program checks your tables, and attempts to repair them if necessary. It also updates your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. (Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.)

`mysql_upgrade` does not upgrade the contents of the help tables. For upgrade instructions, see [Section 5.1.10, “Server-Side Help”](#).

- If you run MySQL Server on Windows, see [Section 2.3.9, “Upgrading MySQL on Windows”](#).
- If you use replication, see [Section 16.4.3, “Upgrading a Replication Setup”](#), for information on upgrading your replication setup.

If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, you might find it useful to create a “dummy” database instance for assessing what conversions might be needed and the work involved to perform them. Make a copy of your MySQL instance that contains a full copy of the `mysql` database, plus all other databases without data. Run your upgrade procedure on this dummy instance to see what actions might be needed so that you can better evaluate the work involved when performing actual data conversion on your original database instance.

The following lists describe changes that may affect applications and that you should watch out for when upgrading from MySQL 5.4 to 5.5.

Configuration Changes

- **Incompatible change:** In MySQL 5.5, the server includes a plugin services interface that complements the plugin API. The services interface enables server functionality to be exposed as a “service” that plugins can access through a function-call interface. The `libmysqlservices` library provides access to the available services and dynamic plugins now must be linked against this library (use the `-lmysqlservices` flag). For an example showing how to configure for CMake, see [Section 23.2.5, “MySQL Services for Plugins”](#).

Server Changes

- **Incompatible change:** As of MySQL 5.5.7, the server requires that a new grant table, `proxies_priv`, be present in the `mysql` database. If you are upgrading from a previous MySQL release rather than performing a new installation, the server will find that this table is missing and exit during startup with the following message:

```
Table 'mysql.proxies_priv' doesn't exist
```

To create the `proxies_priv` table, start the server with the `--skip-grant-tables` [\[461\]](#) option to cause it to skip the normal grant table checks, then run `mysql_upgrade`. For example:

```
shell> mysqld --skip-grant-tables &
shell> mysql_upgrade
```

Then stop the server and restart it normally.

You can specify other options on the `mysqld` command line if necessary. Alternatively, if your installation is configured so that the server normally reads options from an option file, use the `--defaults-file` [\[248\]](#) option to specify the file (enter each command on a single line):

```
shell> mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
--skip-grant-tables &
shell> mysql_upgrade
```

With the `--skip-grant-tables` [\[461\]](#) option, the server does no password or privilege checking, so any client can connect and effectively have all privileges. For additional security, use the `--skip-networking` [\[461\]](#) option as well to prevent remote clients from connecting.

- **Incompatible change:** As of MySQL 5.5.3, the Unicode implementation has been extended to provide support for supplementary characters that lie outside the Basic Multilingual Plane (BMP). Noteworthy features:
 - `utf16` and `utf32` character sets have been added. These correspond to the UTF-16 and UTF-32 encodings of the Unicode character set, and they both support supplementary characters.
 - The `utf8mb4` character set has been added. This is similar to `utf8`, but its encoding allows up to four bytes per character to enable support for supplementary characters.
 - The `ucs2` character set is essentially unchanged except for the inclusion of some newer BMP characters.

In most respects, upgrading to MySQL 5.5 should present few problems with regard to Unicode usage, although there are some potential areas of incompatibility. These are the primary areas of concern:

- For the variable-length character data types (`VARCHAR` and the `TEXT` types), the maximum length in characters is less for `utf8mb4` columns than for `utf8` columns.

- For all character data types (`CHAR`, `VARCHAR`, and the `TEXT` types), the maximum number of characters that can be indexed is less for `utf8mb4` columns than for `utf8` columns.

Consequently, if you want to upgrade tables from `utf8` to `utf8mb4` to take advantage of supplementary-character support, it may be necessary to change some column or index definitions.

For additional details about the new Unicode character sets and potential incompatibilities, see [Section 10.1.10, “Unicode Support”](#), and [Section 10.1.11, “Upgrading from Previous to Current Unicode Support”](#).

- **Incompatible change:** As of MySQL 5.5.3, the server includes `dtoa`, a library for conversion between strings and numbers by David M. Gay. In MySQL, this library provides the basis for improved conversion between string or `DECIMAL` values and approximate-value (`FLOAT/DOUBLE`) numbers.

Because the conversions produced by this library differ in some cases from previous results, the potential exists for incompatibilities in applications that rely on previous results. For example, applications that depend on a specific exact result from previous conversions might need adjustment to accommodate additional precision.

For additional information about the properties of `dtoa` conversions, see [Section 12.2, “Type Conversion in Expression Evaluation”](#).

- **Incompatible change:** In MySQL 5.5, several changes were made regarding the language and character set of error messages:
 - The `--language [446]` option for specifying the directory for the error message file is now deprecated. The new `lc_messages_dir [512]` and `lc_messages [512]` system variables should be used instead, and the server treats `--language [446]` as an alias for `lc_messages_dir [512]`.
 - The `language [510]` system variable has been removed and replaced with the new `lc_messages_dir [512]` and `lc_messages [512]` system variables. `lc_messages_dir [512]` has only a global value and is read only. `lc_messages [512]` has global and session values and can be modified at runtime, so the error message language can be changed while the server is running, and individual clients each can have a different error message language by changing their session `lc_messages [512]` value to a different locale name.
 - Error messages previously were constructed in a mix of character sets. This issue is resolved by constructing error messages internally within the server using UTF-8 and returning them to the client in the character set specified by the `character_set_results [488]` system variable. The content of error messages therefore may in some cases differ from the messages returned previously.

For more information, see [Section 10.2, “Setting the Error Message Language”](#), and [Section 10.1.6, “Character Set for Error Messages”](#).

- Before MySQL 5.1.36, plugin options were boolean options (see [Section 4.2.3.2, “Program Option Modifiers”](#)). If you upgrade to MySQL 5.5 from a version older than 5.1.36 and previously used options of the form `--plugin_name=0` or `--plugin_name=1`, the equivalent options are now `--plugin_name=OFF` and `--plugin_name=ON`, respectively. You also have the choice of requiring plugins to start successfully by using `--plugin_name=FORCE` or `--plugin_name=FORCE_PLUS_PERMANENT`.

SQL Changes

- **Incompatible change:** Previously, the parser accepted an `INTO` clause in nested `SELECT` statements, which is invalid because such statements must return their results to the outer context. As of MySQL 5.5.3, this syntax is no longer permitted and statements that use it must be changed.
- Some keywords may be reserved in MySQL 5.5 that were not reserved in MySQL 5.4. See [Section 9.3, “Reserved Words”](#).

2.12.2. Downgrading MySQL

This section describes what you should do to downgrade to an older MySQL version in the unlikely case that the previous version worked better than the new one.

If you are downgrading within the same release series (for example, from 5.1.13 to 5.1.12) the general rule is that you just have to install the new binaries on top of the old ones. There is no need to do anything with the databases. As always, however, it is always a good idea to make a backup.

The following items form a checklist of things you should do whenever you perform a downgrade:

- Read the upgrading section for the release series from which you are downgrading to be sure that it does not have any features you really need. See [Section 2.12.1, “Upgrading MySQL”](#).
- If there is a downgrading section for that version, you should read that as well.
- To see which new features were added between the version to which you are downgrading and your current version, see the [Release Notes](#).
- Check [Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”](#), to see whether changes to table formats or to character sets or collations were made between your current version of MySQL and the version to which you are downgrading. If so and these changes result in an incompatibility between MySQL versions, you will need to downgrade the affected tables using the instructions in [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#).

In most cases, you can move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL.

If you downgrade from one release series to another, there may be incompatibilities in table storage formats. In this case, use `mysqldump` to dump your tables before downgrading. After downgrading, reload the dump file using `mysql` or `mysqlimport` to re-create your tables. For examples, see [Section 2.12.5, “Copying MySQL Databases to Another Machine”](#).

A typical symptom of a downward-incompatible table format change when you downgrade is that you cannot open tables. In that case, use the following procedure:

1. Stop the older MySQL server that you are downgrading to.
2. Restart the newer MySQL server you are downgrading from.
3. Dump any tables that were inaccessible to the older server by using `mysqldump` to create a dump file.
4. Stop the newer MySQL server and restart the older one.
5. Reload the dump file into the older server. Your tables should be accessible.

It might also be the case that system tables in the `mysql` database have changed and that downgrading introduces some loss of functionality or requires some adjustments. Here are some examples:

- Trigger creation requires the `TRIGGER` privilege as of MySQL 5.1. In MySQL 5.0, there is no `TRIGGER` privilege and `SUPER` is required instead. If you downgrade from MySQL 5.1 to 5.0, you will need to give the `SUPER` privilege to those accounts that had the `TRIGGER` privilege in 5.1.
- Triggers were added in MySQL 5.0, so if you downgrade from 5.0 to 4.1, you cannot use triggers at all.
- The `mysql.proc.comment` column definition changed between MySQL 5.1 and 5.5. After a downgrade from 5.5 to 5.1, this table is seen as corrupt and in need of repair. To work around this problem, execute `mysql_upgrade` from the version of MySQL to which you downgraded.

2.12.2.1. Downgrading to MySQL 5.1

When downgrading to MySQL 5.1 from MySQL 5.5, you should keep in mind the following issues relating to features found in MySQL 5.5, but not in MySQL 5.1:

- **InnoDB.** MySQL 5.5 uses `InnoDB Plugin` as the built-in version of `InnoDB`. MySQL 5.1 includes `InnoDB Plugin` as of 5.1.38, but as an option that must be enabled explicitly. See the [Release Notes](#) for MySQL 5.1.38.

2.12.3. Checking Whether Tables or Indexes Must Be Rebuilt

A binary upgrade or downgrade is one that installs one version of MySQL “in place” over an existing version, without dumping and reloading tables:

1. Stop the server for the existing version if it is running.
2. Install a different version of MySQL. This is an upgrade if the new version is higher than the original version, a downgrade if the version is lower.
3. Start the server for the new version.

In many cases, the tables from the previous version of MySQL can be used without problem by the new version. However, sometimes changes occur that require tables or table indexes to be rebuilt, as described in this section. If you have tables that are affected by any of the issues described here, rebuild the tables or indexes as necessary using the instructions given in [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#).

Table Incompatibilities

After a binary upgrade to MySQL 5.1 from a MySQL 5.0 installation that contains `ARCHIVE` tables, accessing those tables causes the server to crash, even if you have run `mysql_upgrade` or `CHECK TABLE ... FOR UPGRADE`. To work around this problem, use `mysqldump` to dump all `ARCHIVE` tables before upgrading, and reload them into MySQL 5.1 after upgrading. The same problem occurs for binary downgrades from MySQL 5.1 to 5.0.

The upgrade problem is fixed in MySQL 5.6.4: The server can open `ARCHIVE` tables created in MySQL 5.0. However, it remains the recommended upgrade procedure to dump 5.0 `ARCHIVE` tables before upgrading and reload them after upgrading.

Index Incompatibilities

In MySQL 5.6.3, the length limit for index prefix keys is increased from 767 bytes to 3072 bytes, for `InnoDB` tables using `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED`. See [Section 14.3.15, “Limits on InnoDB Tables”](#) for details. This change is also backported to MySQL 5.5.14. If you downgrade from one of these releases or higher, to an earlier release with a lower length limit, the index prefix keys could be truncated at 767 bytes or the downgrade could fail. This issue could only occur if the configuration option `innodb_large_prefix [1560]` was enabled on the server being downgraded.

If you perform a binary upgrade without dumping and reloading tables, you cannot upgrade directly from MySQL 4.1 to 5.1 or higher. This occurs due to an incompatible change in the [MyISAM](#) table index format in MySQL 5.0. Upgrade from MySQL 4.1 to 5.0 and repair all [MyISAM](#) tables. Then upgrade from MySQL 5.0 to 5.1 and check and repair your tables.

Modifications to the handling of character sets or collations might change the character sort order, which causes the ordering of entries in any index that uses an affected character set or collation to be incorrect. Such changes result in several possible problems:

- Comparison results that differ from previous results
- Inability to find some index values due to misordered index entries
- Misordered [ORDER BY](#) results
- Tables that [CHECK TABLE](#) reports as being in need of repair

The solution to these problems is to rebuild any indexes that use an affected character set or collation, either by dropping and re-creating the indexes, or by dumping and reloading the entire table. In some cases, it is possible to alter affected columns to use a different collation. For information about rebuilding indexes, see [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#).

To check whether a table has indexes that must be rebuilt, consult the following list. It indicates which versions of MySQL introduced character set or collation changes that require indexes to be rebuilt. Each entry indicates the version in which the change occurred and the character sets or collations that the change affects. If the change is associated with a particular bug report, the bug number is given.

The list applies both for binary upgrades and downgrades. For example, Bug #27877 was fixed in MySQL 5.1.24 and 5.4.0, so it applies to upgrades from versions older than 5.1.24 to 5.1.24 or newer, and to downgrades from 5.1.24 or newer to versions older than 5.1.24.

In many cases, you can use [CHECK TABLE ... FOR UPGRADE](#) to identify tables for which index rebuilding is required. It will report this message:

```
Table upgrade required.
Please do "REPAIR TABLE `tbl_name`" or dump/reload to fix it!
```

In these cases, you can also use [mysqlcheck --check-upgrade](#) or [mysql_upgrade](#), which execute [CHECK TABLE](#). However, the use of [CHECK TABLE](#) applies only after upgrades, not downgrades. Also, [CHECK TABLE](#) is not applicable to all storage engines. For details about which storage engines [CHECK TABLE](#) supports, see [Section 13.7.2.2, “CHECK TABLE Syntax”](#).

These changes cause index rebuilding to be necessary:

- MySQL 5.1.24, 5.4.0 (Bug #27877)

Affects indexes that use the [utf8_general_ci](#) or [ucs2_general_ci](#) collation for columns that contain 'ß' LATIN SMALL LETTER SHARP S (German). The bug fix corrected an error in the original collations but introduced an incompatibility such that 'ß' compares equal to characters with which it previously compared different.

Affected tables can be detected by [CHECK TABLE ... FOR UPGRADE](#) as of MySQL 5.1.30, 5.4.0 (see Bug #40053).

A workaround for this issue is implemented as of MySQL 5.1.62, 5.5.21, and 5.6.5. The workaround involves altering affected columns to use the [utf8_general_mysql500_ci](#) and [ucs2_general_mysql500_ci](#) collations, which preserve the original pre-5.1.24 ordering of [utf8_general_ci](#) and [ucs2_general_ci](#).

- MySQL 5.0.48, 5.1.23 (Bug #27562)

Affects indexes that use the `ascii_general_ci` collation for columns that contain any of these characters: ``` GRAVE ACCENT, `[` LEFT SQUARE BRACKET, `\` REVERSE SOLIDUS, `]` RIGHT SQUARE BRACKET, `~` TILDE

Affected tables can be detected by `CHECK TABLE ... FOR UPGRADE` as of MySQL 5.1.29, 5.4.0 (see Bug #39585).

- MySQL 5.0.48, 5.1.21 (Bug #29461)

Affects indexes for columns that use any of these character sets: `ucjpm`, `uckr`, `gb2312`, `latin7`, `macce`, `ujis`

Affected tables can be detected by `CHECK TABLE ... FOR UPGRADE` as of MySQL 5.1.29, 5.4.0 (see Bug #39585).

2.12.4. Rebuilding or Repairing Tables or Indexes

This section describes how to rebuild a table. This can be necessitated by changes to MySQL such as how data types are handled or changes to character set handling. For example, an error in a collation might have been corrected, necessitating a table rebuild to update the indexes for character columns that use the collation. (For examples, see [Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”](#).) It might also be that a table repair or upgrade should be done as indicated by a table check operation such as that performed by `CHECK TABLE`, `mysqlcheck`, or `mysql_upgrade`.

Methods for rebuilding a table include dumping and reloading it, or using `ALTER TABLE` or `REPAIR TABLE`.



Note

If you are rebuilding tables because a different version of MySQL will not handle them after a binary (in-place) upgrade or downgrade, you must use the dump-and-reload method. Dump the tables *before* upgrading or downgrading using your original version of MySQL. Then reload the tables *after* upgrading or downgrading.

If you use the dump-and-reload method of rebuilding tables only for the purpose of rebuilding indexes, you can perform the dump either before or after upgrading or downgrading. Reloading still must be done afterward.

To rebuild a table by dumping and reloading it, use `mysqldump` to create a dump file and `mysql` to reload the file:

```
shell> mysqldump db_name t1 > dump.sql
shell> mysql db_name < dump.sql
```

To rebuild all the tables in a single database, specify the database name without any following table name:

```
shell> mysqldump db_name > dump.sql
shell> mysql db_name < dump.sql
```

To rebuild all tables in all databases, use the `--all-databases` [\[313\]](#) option:

```
shell> mysqldump --all-databases > dump.sql
shell> mysql < dump.sql
```

To rebuild a table with `ALTER TABLE`, use a “null” alteration; that is, an `ALTER TABLE` statement that “changes” the table to use the storage engine that it already has. For example, if `t1` is a `MyISAM` table, use this statement:

```
mysql> ALTER TABLE t1 ENGINE = MyISAM;
```

If you are not sure which storage engine to specify in the `ALTER TABLE` statement, use `SHOW CREATE TABLE` to display the table definition.

If you must rebuild a table because a table checking operation indicates that the table is corrupt or needs an upgrade, you can use `REPAIR TABLE` if that statement supports the table's storage engine. For example, to repair a `MyISAM` table, use this statement:

```
mysql> REPAIR TABLE t1;
```

For storage engines such as `InnoDB` that `REPAIR TABLE` does not support, use `mysqldump` to create a dump file and `mysql` to reload the file, as described earlier.

For specifics about which storage engines `REPAIR TABLE` supports, see [Section 13.7.2.5, “REPAIR TABLE Syntax”](#).

`mysqlcheck --repair` provides command-line access to the `REPAIR TABLE` statement. This can be a more convenient means of repairing tables because you can use the `--databases [304]` or `--all-databases [303]` option to repair all tables in specific databases or all databases, respectively:

```
shell> mysqlcheck --repair --databases db_name ...
shell> mysqlcheck --repair --all-databases
```

For incompatibilities introduced in MySQL 5.1.24 by the fix for Bug #27877 that corrected the `utf8_general_ci` and `ucs2_general_ci` collations, a workaround is implemented as of MySQL 5.1.62, 5.5.21, and 5.6.5. Upgrade to one of those versions, then convert each affected table using one of the following methods. In each case, the workaround altering affected columns to use the `utf8_general_mysql500_ci` and `ucs2_general_mysql500_ci` collations, which preserve the original pre-5.1.24 ordering of `utf8_general_ci` and `ucs2_general_ci`.

- To convert an affected table after a binary upgrade that leaves the table files in place, alter the table to use the new collation. Suppose that the table `t1` contains one or more problematic `utf8` columns. To convert the table at the table level, use a statement like this:

```
ALTER TABLE t1
CONVERT TO CHARACTER SET utf8 COLLATE utf8_general_mysql500_ci;
```

To apply the change on a column-specific basis, use a statement like this (be sure to repeat the column definition as originally specified except for the `COLLATE` clause):

```
ALTER TABLE t1
MODIFY c1 CHAR(N) CHARACTER SET utf8 COLLATE utf8_general_mysql500_ci;
```

- To upgrade the table using a dump and reload procedure, dump the table using `mysqldump`, modify the `CREATE TABLE` statement in the dump file to use the new collation, and reload the table.

After making the appropriate changes, `CHECK TABLE` should report no error.

2.12.5. Copying MySQL Databases to Another Machine

You can copy the `.frm`, `.MYI`, and `.MYD` files for `MyISAM` tables between different architectures that support the same floating-point format. (MySQL takes care of any byte-swapping issues.) See [Section 14.5, “The MyISAM Storage Engine”](#).

In cases where you need to transfer databases between different architectures, you can use `mysqldump` to create a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the `mysql` client.

Use `mysqldump --help` to see what options are available.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --compress db_name | mysql db_name
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
shell> mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
shell> mysqladmin create db_name
shell> gunzip < db_name.gz | mysql db_name
```

You can also use `mysqldump` and `mysqlimport` to transfer the database. For large tables, this is much faster than simply using `mysqldump`. In the following commands, `DUMPDIR` represents the full path name of the directory you use to store the output from `mysqldump`.

First, create the directory for the output files and dump the database:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the `DUMPDIR` directory to some corresponding directory on the target machine and load the files into MySQL there:

```
shell> mysqladmin create db_name           # create database
shell> cat DUMPDIR/*.sql | mysql db_name   # create tables in database
shell> mysqlimport db_name DUMPDIR/*.txt  # load data into tables
```

Do not forget to copy the `mysql` database because that is where the grant tables are stored. You might have to run commands as the MySQL `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush-privileges` so that the server reloads the grant table information.

2.13. Environment Variables

This section lists all the environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Note that any options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables.

In many cases, it is preferable to use an option file instead of environment variables to modify the behavior of MySQL. See [Section 4.2.3.3, “Using Option Files”](#).

Variable	Description
CXX	The name of your C++ compiler (for running <code>CMake</code>).
CC	The name of your C compiler (for running <code>CMake</code>).
CFLAGS	Flags for your C compiler (for running <code>CMake</code>).
CXXFLAGS	Flags for your C++ compiler (for running <code>CMake</code>).
DBI_USER	The default user name for Perl DBI.
DBI_TRACE	Trace options for Perl DBI.
HOME	The default path for the <code>mysql</code> history file is <code>\$HOME/.mysql_history</code> .
LD_RUN_PATH	Used to specify the location of <code>libmysqlclient.so</code> .
LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN	Enable <code>mysql_clear_password</code> authentication plugin; see Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin” .
MYSQL_DEBUG	Debug trace options when debugging.
MYSQL_GROUP_SUFFIX	Option group suffix value (like specifying <code>--defaults-group-suffix [248]</code>).
MYSQL_HISTFILE	The path to the <code>mysql</code> history file. If this variable is set, its value overrides the default for <code>\$HOME/.mysql_history</code> .
MYSQL_HOME	The path to the directory in which the server-specific <code>my.cnf</code> file resides.
MYSQL_HOST	The default host name used by the <code>mysql</code> command-line client.
MYSQL_PS1	The command prompt to use in the <code>mysql</code> command-line client.
MYSQL_PWD	The default password when connecting to <code>mysqld</code> . Note that using this is insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security” .
MYSQL_TCP_PORT	The default TCP/IP port number.
MYSQL_UNIX_PORT	The default Unix socket file name; used for connections to <code>localhost</code> .
PATH	Used by the shell to find MySQL programs.
TMPDIR	The directory where temporary files are created.
TZ	This should be set to your local time zone. See Section C.5.4.6, “Time Zone Problems” .
UMASK	The user-file creation mode when creating files. See note following table.
UMASK_DIR	The user-directory creation mode when creating directories. See note following table.
USER	The default user name on Windows when connecting to <code>mysqld</code> .

For information about the `mysql` history file, see [Section 4.5.1.3, “mysql History File”](#).

The `UMASK` and `UMASK_DIR` variables, despite their names, are used as modes, not masks:

- If `UMASK` is set, `mysqld` uses `($\$UMASK$ | 0600)` as the mode for file creation, so that newly created files have a mode in the range from 0600 to 0666 (all values octal).
- If `UMASK_DIR` is set, `mysqld` uses `($\$UMASK_DIR$ | 0700)` as the base mode for directory creation, which then is AND-ed with `($\sim(\sim\$UMASK \& 0666)$)`, so that newly created directories have a mode in the range from 0700 to 0777 (all values octal). The AND operation may remove read and write permissions from the directory mode, but not execute permissions.

MySQL assumes that the value for `UMASK` or `UMASK_DIR` is in octal if it starts with a zero.

2.14. Perl Installation Notes

The Perl `DBI` module provides a generic interface for database access. You can write a `DBI` script that works with many different database engines without change. To use `DBI`, you must install the `DBI` module, as well as a DataBase Driver (DBD) module for each type of database server you want to access. For MySQL, this driver is the `DBD::mysql` module.

Perl, and the `DBD::MySQL` module for `DBI` must be installed if you want to run the MySQL benchmark scripts; see [Section 8.12.2, “The MySQL Benchmark Suite”](#). They are also required for the MySQL Cluster `ndb_size.pl` utility; see [Section 17.4.22, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”](#).



Note

Perl support is not included with MySQL distributions. You can obtain the necessary modules from <http://search.cpan.org> for Unix, or by using the ActiveState `ppm` program on Windows. The following sections describe how to do this.

The `DBI/DBD` interface requires Perl 5.6.0, and 5.6.1 or later is preferred. `DBI` *does not work* if you have an older version of Perl. You should use `DBD::mysql` 4.009 or higher. Although earlier versions are available, they do not support the full functionality of MySQL 5.5.

2.14.1. Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. If you install MySQL from RPM files on Linux, be sure to install the developer RPM as well. The client programs are in the client RPM, but client programming support is in the developer RPM.

The files you need for Perl support can be obtained from the CPAN (Comprehensive Perl Archive Network) at <http://search.cpan.org>.

The easiest way to install Perl modules on Unix is to use the `CPAN` module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The `DBD::mysql` installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and `ODBC` on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

`DBI` requires the `Data::Dumper` module. It may be installed; if not, you should install it before installing `DBI`.

It is also possible to download the module distributions in the form of compressed `tar` archives and build the modules manually. For example, to unpack and build a `DBI` distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

```
shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
```

This command creates a directory named `DBI-VERSION`.

2. Change location into the top-level directory of the unpacked distribution:

```
shell> cd DBI-VERSION
```

3. Build the distribution and compile everything:

```
shell> perl Makefile.PL
shell> make
shell> make test
shell> make install
```

The `make test` command is important because it verifies that the module is working. Note that when you run that command during the `DBD:mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD:mysql` distribution whenever you install a new release of MySQL. This ensures that the latest versions of the MySQL client libraries are installed correctly.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: <http://servers.digitaldaze.com/extensions/perl/modules.html#modules>

Look under the heading “Installing New Modules that Require Locally Installed Modules.”

2.14.2. Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL `DBD` module with ActiveState Perl:

1. Get ActiveState Perl from <http://www.activestate.com/Products/ActivePerl/> and install it.
2. Open a console window.
3. If necessary, set the `HTTP_proxy` variable. For example, you might try a setting like this:

```
C:\> set HTTP_proxy=my.proxy.com:3128
```

4. Start the PPM program:

```
C:\> C:\perl\bin\ppm.pl
```

5. If you have not previously done so, install `DBI`:

```
ppm> install DBI
```

6. If this succeeds, run the following command:

```
ppm> install DBD-mysql
```

This procedure should work with ActiveState Perl 5.6 or newer.

If you cannot get the procedure to work, you should install the ODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn", $user, $password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.14.3. Problems Using the Perl `DBI/DBD` Interface

If Perl reports that it cannot find the `../mysql/mysql.so` module, the problem is probably that Perl cannot locate the `libmysqlclient.so` shared library. You should be able to fix this problem by one of the following methods:

- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `/usr/lib` or `/lib`).
- Modify the `-L` options used to compile `DBD:mysql` to reflect the actual location of `libmysqlclient.so`.
- On Linux, you can add the path name of the directory where `libmysqlclient.so` is located to the `etc/ld.so.conf` file.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable. Some systems use `LD_LIBRARY_PATH` instead.

Note that you may also need to modify the `-L` options if there are other libraries that the linker fails to find. For example, if the linker cannot find `libc` because it is in `/lib` and the link command specifies `-L/usr/lib`, change the `-L` option to `-L/lib` or add `-L/lib` to the existing link command.

If you get the following errors from `DBD:mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the `mysql.so` library gets built (check the output from `make` for `mysql.so` when you compile the Perl client). The `-L` option should specify the path name of the directory where `libgcc.a` is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

You may see the following error from `DBD:mysql` when you run the tests:

```
t/00base.....install_driver(mysql) failed:
Can't load '../blib/arch/auto/DBD/mysql/mysql.so' for module DBD:mysql:
../blib/arch/auto/DBD/mysql/mysql.so: undefined symbol:
uncompress at /usr/lib/perl5/5.00503/i586-linux/DynaLoader.pm line 169.
```

This means that you need to include the `-lz` compression library on the link line. That can be done by changing the following line in the file `lib/DBD/mysql/Install.pm`:

```
$sysliblist .= " -lm";
```

Change that line to:

```
$sysliblist .= " -lm -lz";
```

After this, you *must* run `make realclean` and then proceed with the installation from the beginning.

Chapter 3. Tutorial

Table of Contents

3.1. Connecting to and Disconnecting from the Server	197
3.2. Entering Queries	198
3.3. Creating and Using a Database	201
3.3.1. Creating and Selecting a Database	202
3.3.2. Creating a Table	203
3.3.3. Loading Data into a Table	204
3.3.4. Retrieving Information from a Table	206
3.4. Getting Information About Databases and Tables	219
3.5. Using <code>mysql</code> in Batch Mode	220
3.6. Examples of Common Queries	221
3.6.1. The Maximum Value for a Column	222
3.6.2. The Row Holding the Maximum of a Certain Column	222
3.6.3. Maximum of Column per Group	223
3.6.4. The Rows Holding the Group-wise Maximum of a Certain Column	223
3.6.5. Using User-Defined Variables	224
3.6.6. Using Foreign Keys	224
3.6.7. Searching on Two Keys	226
3.6.8. Calculating Visits Per Day	226
3.6.9. Using <code>AUTO_INCREMENT</code>	226
3.7. Using MySQL with Apache	229

This chapter provides a tutorial introduction to MySQL by showing how to use the `mysql` client program to create and use a simple database. `mysql` (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that enables you to connect to a MySQL server, run queries, and view the results. `mysql` may also be used in batch mode: you place your queries in a file beforehand, then tell `mysql` to execute the contents of the file. Both ways of using `mysql` are covered here.

To see a list of options provided by `mysql`, invoke it with the `--help [276]` option:

```
shell> mysql --help
```

This chapter assumes that `mysql` is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as [Chapter 5, MySQL Server Administration](#).)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1. Connecting to and Disconnecting from the Server

To connect to the server, you will usually need to provide a MySQL user name when you invoke `mysql` and, most likely, a password. If the server runs on a machine other than the one where you log in, you will

also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: *****
```

`host` and `user` represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The `*****` represents your password; enter it when `mysql` displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.5.31-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The `mysql>` prompt tells you that `mysql` is ready for you to enter commands.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell> mysql -u user -p
```

If, when you attempt to log in, you get an error message such as `ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)`, it means that the MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of [Chapter 2, *Installing and Upgrading MySQL*](#) that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see [Section C.5.2, "Common Errors When Using MySQL Programs"](#).

Some MySQL installations permit users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking `mysql` without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing `QUIT` (or `\q`) at the `mysql>` prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control+D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the `mysql>` prompt.

3.2. Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that is okay. At this point, it is more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data

from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how `mysql` works.

Here is a simple command that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----+-----+
| VERSION() | CURRENT_DATE |
+-----+-----+
| 5.5.0-m2-log | 2009-05-04 |
+-----+-----+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about `mysql`:

- A command normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a command, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another command.
- `mysql` displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), `mysql` labels the column using the expression itself.
- `mysql` shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the "rows in set" line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here is another query. It demonstrates that you can use `mysql` as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
| 0.70710678118655 | 25 |
+-----+-----+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+-----+
| VERSION() |
+-----+
| 5.5.0-m2-log |
+-----+
1 row in set (0.00 sec)
```

```
+-----+
| NOW() |
+-----+
| 2009-05-04 15:15:00 |
+-----+
1 row in set (0.00 sec)
```

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here is a simple multiple-line statement:

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
+-----+-----+
| USER() | CURRENT_DATE |
+-----+-----+
| jon@localhost | 2005-10-11 |
+-----+-----+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what `mysql` is waiting for.

If you decide you do not want to execute a command that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
-> USER()
-> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in.

Prompt	Meaning
<code>mysql></code>	Ready for new command.
<code>-></code>	Waiting for next line of multiple-line command.
<code>'></code>	Waiting for next line, waiting for completion of a string that began with a single quote (" ").
<code>"></code>	Waiting for next line, waiting for completion of a string that began with a double quote (" ").
<code>`></code>	Waiting for next line, waiting for completion of an identifier that began with a backtick (" `").
<code>/*></code>	Waiting for next line, waiting for completion of a comment that began with <code>/*</code> .

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```
mysql> SELECT USER()
->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` executes it:

```
mysql> SELECT USER()
      -> ;
+-----+
| USER() |
+-----+
| jon@localhost |
+-----+
```

The `'>` and `">` prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either `"` or `'` characters (for example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you have entered a line containing a string that begins with a `"` or `'` quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
      '>
```

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quotation mark.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
      '> '\c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new command.

The ``>` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and ``>` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by `mysql`—including a line containing `QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current command.

3.3. Creating and Using a Database

Once you know how to enter commands, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to perform the following operations:

- Create a database
- Create a table

- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL Web site. It is available in both compressed `tar` file and Zip formats at <http://dev.mysql.com/doc/>.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql    |
| test     |
| tmp      |
+-----+
```

The `mysql` database describes user access privileges. The `test` database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; `SHOW DATABASES` does not show databases that you have no privileges for if you do not have the `SHOW DATABASES` privilege. See [Section 13.7.5.15, “SHOW DATABASES Syntax”](#).

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

`USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours `menagerie`. The administrator needs to execute a command like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where `your_mysql_name` is the MySQL user name assigned to you and `your_client_host` is the host from which you connect to the server.

3.3.1. Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case sensitive (unlike SQL keywords), so you must always refer to your database as `menagerie`, not as `Menagerie`, `MENAGERIE`, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases

and tables using the same lettercase throughout a given query. However, for a variety of reasons, the recommended best practice is always to use the same lettercase that was used when the database was created.)



Note

If you get an error such as `ERROR 1044 (42000): Access denied for user 'monty'@'localhost' to database 'menagerie'` when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see [Section 6.2, “The MySQL Access Privilege System”](#).

Creating a database does not select it for use; you must do that explicitly. To make `menagerie` the current database, use this command:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a `mysql` session. You can do this by issuing a `USE` statement as shown in the example. Alternatively, you can select the database on the command line when you invoke `mysql`. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: *****
```



Important

`menagerie` in the command just shown is **not** your password. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (for example, as `-pmypassword`, not as `-p mypassword`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.



Note

You can see at any time which database is currently selected using `SELECT DATABASE() [1193]`.

3.3.2. Creating a Table

Creating the database is the easy part, but at this point it is empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed

value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`. You can normally pick any length from `1` to `65535`, whatever seems most reasonable to you. If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It is simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
+-----+
| Tables in menagerie |
+-----+
| pet                |
+-----+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | varchar(20) | YES  |     | NULL    |       |
| owner | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex   | char(1)    | YES  |     | NULL    |       |
| birth | date       | YES  |     | NULL    |       |
| death | date       | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see [Chapter 11, Data Types](#).

3.3.3. Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in `'YYYY-MM-DD'` format; this may be different from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler      Gwen      bird      \N      1997-12-09      \N
```

To load the text file `pet.txt` into the `pet` table, use this statement:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

If you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use this statement instead:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
-> LINES TERMINATED BY '\r\n';
```

(On an Apple machine running OS X, you would likely want to use `LINES TERMINATED BY '\r'`.)

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt` properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See [Section 6.1.6, "Security Issues with `LOAD DATA LOCAL`"](#), for information on how to change this.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose that Diane gets a new hamster named "Puffball." You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
-> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

String and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

3.3.4. Retrieving Information from a Table

The `SELECT` statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

what_to_select indicates what you want to see. This can be a list of columns, or `*` to indicate “all columns.” *which_table* indicates the table from which you want to retrieve data. The `WHERE` clause is optional. If it is present, *conditions_to_satisfy* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1. Selecting All Data

The simplest form of `SELECT` retrieves everything from a table:

```
mysql> SELECT * FROM pet;
```

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	NULL
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL
Fang	Benny	dog	m	1990-08-27	NULL
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	NULL
Whistler	Gwen	bird	NULL	1997-12-09	NULL
Slim	Benny	snake	m	1996-04-29	NULL
Puffball	Diane	hamster	f	1999-03-30	NULL

This form of `SELECT` is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file `pet.txt` to correct the error, then empty the table and reload it using `DELETE` and `LOAD DATA`:

```
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
```

However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an `UPDATE` statement:

```
mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
```

The `UPDATE` changes only the record in question and does not require you to reload the table.

3.3.4.2. Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the `WHERE` clause from the `SELECT` statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
```

name	owner	species	sex	birth	death
Bowser	Diane	dog	m	1989-08-31	1995-07-29

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as `'bowser'`, `'BOWSER'`, and so forth. The query result is the same.

You can specify conditions on any column, not just `name`. For example, if you want to know which animals were born during or after 1998, test the `birth` column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
```

name	owner	species	sex	birth	death
Chirpy	Gwen	bird	f	1998-09-11	NULL
Puffball	Diane	hamster	f	1999-03-30	NULL

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
```

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	NULL

The preceding query uses the `AND` [1094] logical operator. There is also an `OR` [1095] operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
```

name	owner	species	sex	birth	death
Chirpy	Gwen	bird	f	1998-09-11	NULL
Whistler	Gwen	bird	NULL	1997-12-09	NULL
Slim	Benny	snake	m	1996-04-29	NULL

`AND` [1094] and `OR` [1095] may be intermixed, although `AND` [1094] has higher precedence than `OR` [1095]. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
-> OR (species = 'dog' AND sex = 'f');
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

3.3.4.3. Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
+-----+-----+
| name   | birth   |
+-----+-----+
| Fluffy | 1993-02-04 |
| Claws  | 1994-03-17 |
| Buffy  | 1989-05-13 |
| Fang   | 1990-08-27 |
| Bowser | 1989-08-31 |
| Chirpy | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim   | 1996-04-29 |
| Puffball | 1999-03-30 |
+-----+-----+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+-----+
| owner |
+-----+
| Harold |
| Gwen  |
| Harold |
| Benny |
| Diane |
| Gwen  |
| Gwen  |
| Benny |
| Diane |
+-----+
```

Notice that the query simply retrieves the `owner` column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+-----+
| owner |
+-----+
| Benny |
| Diane |
| Gwen  |
| Harold |
+-----+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
-> WHERE species = 'dog' OR species = 'cat';
+-----+-----+-----+
| name   | species | birth   |
+-----+-----+-----+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
+-----+-----+-----+
```

Buffy	dog	1989-05-13
Fang	dog	1990-08-27
Bowser	dog	1989-08-31

3.3.4.4. Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It is often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+-----+-----+
| name   | birth   |
+-----+-----+
| Buffy  | 1989-05-13 |
| Bowser | 1989-08-31 |
| Fang   | 1990-08-27 |
| Fluffy | 1993-02-04 |
| Claws  | 1994-03-17 |
| Slim   | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy | 1998-09-11 |
| Puffball | 1999-03-30 |
+-----+-----+
```

On character type columns, sorting—like all other comparison operations—is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY [1168]` like so: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+-----+-----+
| name   | birth   |
+-----+-----+
| Puffball | 1999-03-30 |
| Chirpy  | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim    | 1996-04-29 |
| Claws   | 1994-03-17 |
| Fluffy  | 1993-02-04 |
| Fang    | 1990-08-27 |
| Bowser  | 1989-08-31 |
| Buffy   | 1989-05-13 |
+-----+-----+
```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
-> ORDER BY species, birth DESC;
+-----+-----+-----+
| name   | species | birth   |
+-----+-----+-----+
| Chirpy | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws  | cat     | 1994-03-17 |
+-----+-----+-----+
```

Fluffy	cat	1993-02-04
Fang	dog	1990-08-27
Bowser	dog	1989-08-31
Buffy	dog	1989-05-13
Puffball	hamster	1999-03-30
Slim	snake	1996-04-29

The `DESC` keyword applies only to the column name immediately preceding it (`birth`); it does not affect the `species` column sort order.

3.3.4.5. Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, compute the difference in the year part of the current date and the birth date, then subtract one if the current date occurs earlier in the calendar year than the birth date. The following query shows, for each pet, the birth date, the current date, and the age in years.

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
-> AS age
-> FROM pet;
```

name	birth	CURDATE()	age
Fluffy	1993-02-04	2003-08-19	10
Claws	1994-03-17	2003-08-19	9
Buffy	1989-05-13	2003-08-19	14
Fang	1990-08-27	2003-08-19	12
Bowser	1989-08-31	2003-08-19	13
Chirpy	1998-09-11	2003-08-19	4
Whistler	1997-12-09	2003-08-19	5
Slim	1996-04-29	2003-08-19	7
Puffball	1999-03-30	2003-08-19	4

Here, `YEAR()` [1152] pulls out the year part of a date and `RIGHT()` [1107] pulls off the rightmost five characters that represent the `MM-DD` (calendar year) part of the date. The part of the expression that compares the `MM-DD` values evaluates to 1 or 0, which adjusts the year difference down a year if `CURDATE()` [1135] occurs earlier in the year than `birth`. The full expression is somewhat ungainly, so an *alias* (`age`) is used to make the output column label more meaningful.

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an `ORDER BY name` clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
-> AS age
-> FROM pet ORDER BY name;
```

name	birth	CURDATE()	age
Bowser	1989-08-31	2003-08-19	13
Buffy	1989-05-13	2003-08-19	14
Chirpy	1998-09-11	2003-08-19	4
Claws	1994-03-17	2003-08-19	9
Fang	1990-08-27	2003-08-19	12
Fluffy	1993-02-04	2003-08-19	10

Puffball	1999-03-30	2003-08-19	4
Slim	1996-04-29	2003-08-19	7
Whistler	1997-12-09	2003-08-19	5

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
-> AS age
-> FROM pet ORDER BY age;
```

name	birth	CURDATE()	age
Chirpy	1998-09-11	2003-08-19	4
Puffball	1999-03-30	2003-08-19	4
Whistler	1997-12-09	2003-08-19	5
Slim	1996-04-29	2003-08-19	7
Claws	1994-03-17	2003-08-19	9
Fluffy	1993-02-04	2003-08-19	10
Fang	1990-08-27	2003-08-19	12
Bowser	1989-08-31	2003-08-19	13
Buffy	1989-05-13	2003-08-19	14

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
-> (YEAR(death)-YEAR(birth)) - (RIGHT(death,5)<RIGHT(birth,5))
-> AS age
-> FROM pet WHERE death IS NOT NULL ORDER BY age;
```

name	birth	death	age
Bowser	1989-08-31	1995-07-29	5

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See [Section 3.3.4.6, “Working with NULL Values”](#).

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several functions for extracting parts of dates, such as `YEAR()` [1152], `MONTH()` [1143], and `DAYOFMONTH()` [1140]. `MONTH()` [1143] is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)` [1143]:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
```

name	birth	MONTH(birth)
Fluffy	1993-02-04	2
Claws	1994-03-17	3
Buffy	1989-05-13	5
Fang	1990-08-27	8
Bowser	1989-08-31	8
Chirpy	1998-09-11	9
Whistler	1997-12-09	12
Slim	1996-04-29	4
Puffball	1999-03-30	3

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is 4 and you can look for animals born in May (month 5) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-----+-----+
| name | birth |
+-----+-----+
| Buffy | 1989-05-13 |
+-----+-----+
```

There is a small complication if the current month is December. You cannot merely add one to the month number (12) and look for animals born in month 13, because there is no such month. Instead, you look for animals born in January (month 1).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` [1135] enables you to add a time interval to a given date. If you add a month to the value of `CURDATE()` [1135], then extract the month part with `MONTH()` [1143], the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
-> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add 1 to get the next month after the current one after using the modulo function (`MOD`) to wrap the month value to 0 if it is currently 12:

```
mysql> SELECT name, birth FROM pet
-> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

`MONTH()` [1143] returns a number between 1 and 12. And `MOD(something, 12)` [1127] returns a number between 0 and 11. So the addition has to be after the `MOD()` [1127], otherwise we would go from November (11) to January (1).

3.3.4.6. Working with NULL Values

The `NULL` value can be surprising until you get used to it. Conceptually, `NULL` means “a missing unknown value” and it is treated somewhat differently from other values.

To test for `NULL`, use the `IS NULL` [1091] and `IS NOT NULL` [1091] operators, as shown here:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----+-----+
| 1 IS NULL | 1 IS NOT NULL |
+-----+-----+
|          0 |              1 |
+-----+-----+
```

You cannot use arithmetic comparison operators such as `=` [1089], `<` [1090], or `<>` [1090] to test for `NULL`. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+-----+-----+-----+-----+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+-----+-----+-----+-----+
|      NULL |      NULL |      NULL |      NULL |
+-----+-----+-----+-----+
```

Because the result of any arithmetic comparison with `NULL` is also `NULL`, you cannot obtain any meaningful results from such comparisons.

In MySQL, 0 or `NULL` means false and anything else means true. The default truth value from a boolean operation is 1.

This special treatment of `NULL` is why, in the previous section, it was necessary to determine which animals are no longer alive using `death IS NOT NULL` instead of `death <> NULL`.

Two `NULL` values are regarded as equal in a `GROUP BY`.

When doing an `ORDER BY`, `NULL` values are presented first if you do `ORDER BY ... ASC` and last if you do `ORDER BY ... DESC`.

A common error when working with `NULL` is to assume that it is not possible to insert a zero or an empty string into a column defined as `NOT NULL`, but this is not the case. These are in fact values, whereas `NULL` means “not having a value.” You can test this easily enough by using `IS [NOT] NULL` as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----+-----+-----+-----+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----+-----+-----+-----+
|          0 |                1 |           0 |                1 |
+-----+-----+-----+-----+
```

Thus it is entirely possible to insert a zero or empty string into a `NOT NULL` column, as these are in fact `NOT NULL`. See [Section C.5.5.3, “Problems with NULL Values”](#).

3.3.4.7. Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as `vi`, `grep`, and `sed`.

SQL pattern matching enables you to use “`_`” to match any single character and “`%`” to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. You do not use `=` or `<>` when you use SQL patterns; use the `LIKE [1111]` or `NOT LIKE [1113]` comparison operators instead.

To find names beginning with “`b`”:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+-----+-----+-----+-----+-----+-----+
| name   | owner | species | sex | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Buffy  | Harold | dog     | f   | 1989-05-13 | NULL       |
| Bowser | Diane | dog     | m   | 1989-08-31 | 1995-07-29 |
+-----+-----+-----+-----+-----+-----+
```

To find names ending with “`fy`”:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+-----+-----+-----+-----+-----+-----+
| name   | owner | species | sex | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Fluffy | Harold | cat     | f   | 1993-02-04 | NULL       |
| Buffy  | Harold | dog     | f   | 1989-05-13 | NULL       |
+-----+-----+-----+-----+-----+-----+
```

To find names containing a “`w`”:

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+-----+-----+-----+-----+-----+-----+
| name   | owner | species | sex | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Claws  | Gwen  | cat     | m   | 1994-03-17 | NULL       |
| Bowser | Diane | dog     | m   | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen | bird    | NULL | 1997-12-09 | NULL       |
+-----+-----+-----+-----+-----+-----+
```

To find names containing exactly five characters, use five instances of the “_” pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the [REGEXP \[1114\]](#) and [NOT REGEXP \[1114\]](#) operators (or [RLIKE \[1114\]](#) and [NOT RLIKE \[1114\]](#), which are synonyms).

The following list describes some characteristics of extended regular expressions:

- “.” matches any single character.
- A character class “[...]” matches any character within the brackets. For example, “[abc]” matches “a”, “b”, or “c”. To name a range of characters, use a dash. “[a-z]” matches any letter, whereas “[0-9]” matches any digit.
- “*” matches zero or more instances of the thing preceding it. For example, “x*” matches any number of “x” characters, “[0-9]*” matches any number of digits, and “. *” matches any number of anything.
- A [REGEXP \[1114\]](#) pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a [LIKE \[1111\]](#) pattern match, which succeeds only if the pattern matches the entire value.)
- To anchor a pattern so that it must match the beginning or end of the value being tested, use “^” at the beginning or “\$” at the end of the pattern.

To demonstrate how extended regular expressions work, the [LIKE \[1111\]](#) queries shown previously are rewritten here to use [REGEXP \[1114\]](#).

To find names beginning with “b”, use “^” to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
```

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29

If you really want to force a [REGEXP \[1114\]](#) comparison to be case sensitive, use the [BINARY \[1168\]](#) keyword to make one of the strings a binary string. This query matches only lowercase “b” at the beginning of a name:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

To find names ending with “fy”, use “\$” to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
```

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

To find names containing a “w”, use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29
Whistler	Gwen	bird	NULL	1997-12-09	NULL

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

To find names containing exactly five characters, use “^” and “\$” to match the beginning and end of the name, and five instances of “.” in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

You could also write the previous query using the `{n}` (“repeat-*n*-times”) operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

[Section 12.5.2, “Regular Expressions”](#), provides more information about the syntax for regular expressions.

3.3.4.8. Counting Rows

Databases are often used to answer the question, “How often does a certain type of data occur in a table?” For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as “How many rows are in the `pet` table?” because there is one record per pet. `COUNT(*)` [1204] counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
```

COUNT(*)
9

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` [1204] if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
```

owner	COUNT(*)
Benny	2
Diane	2
Gwen	3

```
| Harold |      2 |
+-----+-----+
```

The preceding query uses `GROUP BY` to group all records for each `owner`. The use of `COUNT()` [1204] in conjunction with `GROUP BY` is useful for characterizing your data under various groupings. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+-----+-----+
| species | COUNT(*) |
+-----+-----+
| bird   |      2 |
| cat    |      2 |
| dog    |      3 |
| hamster|      1 |
| snake  |      1 |
+-----+-----+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+-----+-----+
| sex   | COUNT(*) |
+-----+-----+
| NULL  |      1 |
| f     |      4 |
| m     |      4 |
+-----+-----+
```

(In this output, `NULL` indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+-----+-----+-----+
| species | sex   | COUNT(*) |
+-----+-----+-----+
| bird   | NULL  |      1 |
| bird   | f     |      1 |
| cat    | f     |      1 |
| cat    | m     |      1 |
| dog    | f     |      1 |
| dog    | m     |      2 |
| hamster| f     |      1 |
| snake  | m     |      1 |
+-----+-----+-----+
```

You need not retrieve an entire table when you use `COUNT()` [1204]. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
-> WHERE species = 'dog' OR species = 'cat'
-> GROUP BY species, sex;
+-----+-----+-----+
| species | sex   | COUNT(*) |
+-----+-----+-----+
| cat    | f     |      1 |
| cat    | m     |      1 |
| dog    | f     |      1 |
| dog    | m     |      2 |
+-----+-----+-----+
```

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
-> WHERE sex IS NOT NULL
-> GROUP BY species, sex;
+-----+-----+-----+
| species | sex | COUNT(*) |
+-----+-----+-----+
| bird   | f   | 1         |
| cat    | f   | 1         |
| cat    | m   | 1         |
| dog    | f   | 1         |
| dog    | m   | 2         |
| hamster| f   | 1         |
| snake  | m   | 1         |
+-----+-----+-----+
```

If you name columns to select in addition to the `COUNT()` [1204] value, a `GROUP BY` clause should be present that names those same columns. Otherwise, the following occurs:

- If the `ONLY_FULL_GROUP_BY` [615] SQL mode is enabled, an error occurs:

```
mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

- If `ONLY_FULL_GROUP_BY` [615] is not enabled, the query is processed by treating all rows as a single group, but the value selected for each named column is indeterminate. The server is free to select the value from any row:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
+-----+-----+
| owner | COUNT(*) |
+-----+-----+
| Harold | 8         |
+-----+-----+
1 row in set (0.00 sec)
```

See also [Section 12.16.3, “MySQL Extensions to `GROUP BY`”](#).

3.3.4.9. Using More Than one Table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs to contain the following information:

- The pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
-> type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it is easiest to load the initial records by creating a tab-delimited text file containing the following information.

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
-> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS age,
-> remark
-> FROM pet INNER JOIN event
-> ON pet.name = event.name
-> WHERE event.type = 'litter';
```

```
+-----+-----+-----+
| name | age | remark |
+-----+-----+-----+
| Fluffy | 2 | 4 kittens, 3 female, 1 male |
| Buffy | 4 | 5 puppies, 2 female, 3 male |
| Buffy | 5 | 3 puppies, 3 female |
+-----+-----+-----+
```

There are several things to note about this query:

- The `FROM` clause joins two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a `name` column. The query uses an `ON` clause to match up records in the two tables based on the `name` values.

The query uses an `INNER JOIN` to combine the tables. An `INNER JOIN` permits rows from either table to appear in the result if and only if both tables meet the conditions specified in the `ON` clause. In this example, the `ON` clause specifies that the `name` column in the `pet` table must match the `name` column in the `event` table. If a name appears in one table but not the other, the row will not appear in the result because the condition in the `ON` clause fails.

- Because the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the `pet` table with itself to produce candidate pairs of males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
-> FROM pet AS p1 INNER JOIN pet AS p2
-> ON p1.species = p2.species AND p1.sex = 'f' AND p2.sex = 'm';
```

name	sex	name	sex	species
Fluffy	f	Claws	m	cat
Buffy	f	Fang	m	dog
Buffy	f	Bowser	m	dog

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4. Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` [1193] function:

```
mysql> SELECT DATABASE();
```

DATABASE()
menagerie

If you have not yet selected any database, the result is `NULL`.

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this command:

```
mysql> SHOW TABLES;
```

Tables_in_menagerie
event
pet

The name of the column in the output produced by this statement is always `Tables_in_db_name`, where `db_name` is the name of the database. See [Section 13.7.5.38, “SHOW TABLES Syntax”](#), for more information.

If you want to find out about the structure of a table, the `DESCRIBE` statement is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
```

Field	Type	Null	Key	Default	Extra
name	varchar(20)	YES		NULL	
owner	varchar(20)	YES		NULL	
species	varchar(20)	YES		NULL	
sex	char(1)	YES		NULL	
birth	date	YES		NULL	
death	date	YES		NULL	

`Field` indicates the column name, `Type` is the data type for the column, `NULL` indicates whether the column can contain `NULL` values, `Key` indicates whether the column is indexed, and `Default` specifies the column's default value. `Extra` displays special information about columns: If a column was created with the `AUTO_INCREMENT` option, the value will be `auto_increment` rather than empty.

`DESC` is a short form of `DESCRIBE`. See [Section 13.8.1, “DESCRIBE Syntax”](#), for more information.

You can obtain the `CREATE TABLE` statement necessary to create an existing table using the `SHOW CREATE TABLE` statement. See [Section 13.7.5.12, “SHOW CREATE TABLE Syntax”](#).

If you have indexes on a table, `SHOW INDEX FROM tbl_name` produces information about them. See [Section 13.7.5.23, “SHOW INDEX Syntax”](#), for more about this statement.

3.5. Using `mysql` in Batch Mode

In the previous sections, you used `mysql` interactively to enter queries and view the results. You can also run `mysql` in batch mode. To do this, put the commands you want to run in a file, then tell `mysql` to read its input from the file:

```
shell> mysql < batch-file
```

If you are running `mysql` under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: *****
```

When you use `mysql` this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force [278]` command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script enables you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line commands or multiple-statement sequences of commands. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

```
shell> mysql < batch-file | more
```

- You can catch the output in a file for further processing:

```
shell> mysql < batch-file > mysql.out
```

- You can distribute your script to other people so that they can also run the commands.
- Some situations do not allow for interactive use, for example, when you run a query from a [cron](#) job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when `mysql` is run interactively:

```
+-----+
| species |
+-----+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+-----+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the commands that are executed, use `mysql -vvv`.

You can also use scripts from the `mysql` prompt by using the `source` command or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

See [Section 4.5.1.5, “Executing SQL Statements from a Text File”](#), for more information.

3.6. Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then `(article, dealer)` is a primary key for the records.

Start the command-line tool `mysql` and select a database:

```
shell> mysql your-database-name
```

(In most MySQL installations, you can use the database named `test`).

You can create and populate the example table with these statements:

```
CREATE TABLE shop (
```

```

article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
dealer CHAR(20) DEFAULT ' ' NOT NULL,
price DOUBLE(16,2) DEFAULT '0.00' NOT NULL,
PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
(1, 'A', 3.45), (1, 'B', 3.99), (2, 'A', 10.99), (3, 'B', 1.45),
(3, 'C', 1.69), (3, 'D', 1.25), (4, 'D', 19.95);

```

After issuing the statements, the table should have the following contents:

```

SELECT * FROM shop;

```

article	dealer	price
0001	A	3.45
0001	B	3.99
0002	A	10.99
0003	B	1.45
0003	C	1.69
0003	D	1.25
0004	D	19.95

3.6.1. The Maximum Value for a Column

“What is the highest item number?”

```

SELECT MAX(article) AS article FROM shop;

```

article
4

3.6.2. The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

```

SELECT article, dealer, price
FROM shop
WHERE price=(SELECT MAX(price) FROM shop);

```

article	dealer	price
0004	D	19.95

Other solutions are to use a **LEFT JOIN** or to sort all rows descending by price and get only the first row using the MySQL-specific **LIMIT** clause:

```

SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;

```

**Note**

If there were several most expensive articles, each with a price of 19.95, the `LIMIT` solution would show only one of them.

3.6.3. Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article;
```

article	price
0001	3.99
0002	10.99
0003	1.69
0004	19.95

3.6.4. The Rows Holding the Group-wise Maximum of a Certain Column

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article);
```

article	dealer	price
0001	B	3.99
0002	A	10.99
0003	C	1.69
0004	D	19.95

The preceding example uses a correlated subquery, which can be inefficient (see [Section 13.2.10.7, “Correlated Subqueries”](#)). Other possibilities for solving the problem are to use an uncorrelated subquery in the `FROM` clause or a `LEFT JOIN`:

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
ON s1.article = s2.article AND s1.price = s2.price;
```

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL;
```

The `LEFT JOIN` works on the basis that when `s1.price` is at its maximum value, there is no `s2.price` with a greater value and the `s2` rows values will be `NULL`. See [Section 13.2.9.2, “JOIN Syntax”](#).

3.6.5. Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See [Section 9.4, “User-Defined Variables”](#).)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
```

article	dealer	price
0003	D	1.25
0004	D	19.95



Note

It is also possible to store the name of a database object such as a table or a column in a user variable and then to use this variable in an SQL statement; however, this requires the use of a prepared statement. See [Section 13.5, “SQL Syntax for Prepared Statements”](#), for more information.

3.6.6. Using Foreign Keys

In MySQL, [InnoDB](#) tables support checking of foreign key constraints. See [Section 14.3, “The InnoDB Storage Engine”](#), and [Section 1.8.5.4, “Foreign Key Differences”](#).

A foreign key constraint is not required merely to join two tables. For storage engines other than [InnoDB](#), it is possible when defining a column to use a `REFERENCES tbl_name(col_name)` clause, which has no actual effect, and *serves only as a memo or comment to you that the column which you are currently defining is intended to refer to a column in another table*. It is extremely important to realize when using this syntax that:

- MySQL does not perform any sort of `CHECK` to make sure that `col_name` actually exists in `tbl_name` (or even that `tbl_name` itself exists).
- MySQL does not perform any sort of action on `tbl_name` such as deleting rows in response to actions taken on rows in the table which you are defining; in other words, this syntax induces no `ON DELETE` or `ON UPDATE` behavior whatsoever. (Although you can write an `ON DELETE` or `ON UPDATE` clause as part of the `REFERENCES` clause, it is also ignored.)
- This syntax creates a *column*; it does **not** create any sort of index or key.

You can use a column so created as a join column, as shown here:

```
CREATE TABLE person (
  id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  name CHAR(60) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE shirt (
  id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
  color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
  owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
  PRIMARY KEY (id)
);
```

```

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+-----+
| id | name          |
+----+-----+
|  1 | Antonio Paz   |
|  2 | Lilliana Angelovska |
+----+-----+

SELECT * FROM shirt;
+----+-----+-----+-----+
| id | style  | color | owner |
+----+-----+-----+-----+
|  1 | polo   | blue  | 1     |
|  2 | dress  | white | 1     |
|  3 | t-shirt| blue  | 1     |
|  4 | dress  | orange| 2     |
|  5 | polo   | red   | 2     |
|  6 | dress  | blue  | 2     |
|  7 | t-shirt| white | 2     |
+----+-----+-----+-----+

SELECT s.* FROM person p INNER JOIN shirt s
  ON s.owner = p.id
  WHERE p.name LIKE 'Lilliana%'
  AND s.color <> 'white';

+----+-----+-----+-----+
| id | style  | color | owner |
+----+-----+-----+-----+
|  4 | dress  | orange| 2     |
|  5 | polo   | red   | 2     |
|  6 | dress  | blue  | 2     |
+----+-----+-----+-----+

```

When used in this fashion, the [REFERENCES](#) clause is not displayed in the output of `SHOW CREATE TABLE` or `DESCRIBE`:

```

SHOW CREATE TABLE shirt\G
***** 1. row *****
Table: shirt
Create Table: CREATE TABLE `shirt` (
  `id` smallint(5) unsigned NOT NULL auto_increment,
  `style` enum('t-shirt','polo','dress') NOT NULL,
  `color` enum('red','blue','orange','white','black') NOT NULL,
  `owner` smallint(5) unsigned NOT NULL,
  PRIMARY KEY (`id`)

```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

The use of [REFERENCES](#) in this way as a comment or “reminder” in a column definition works with [MyISAM](#) tables.

3.6.7. Searching on Two Keys

An [OR \[1095\]](#) using a single key is well optimized, as is the handling of [AND \[1094\]](#).

The one tricky case is that of searching on two different keys combined with [OR \[1095\]](#):

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR field2_index = '1'
```

This case is optimized. See [Section 8.13.2, “Index Merge Optimization”](#).

You can also solve the problem efficiently by using a [UNION](#) that combines the output of two separate [SELECT](#) statements. See [Section 13.2.9.4, “UNION Syntax”](#).

Each [SELECT](#) searches only one key and can be optimized:

```
SELECT field1_index, field2_index
FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
FROM test_table WHERE field2_index = '1';
```

3.6.8. Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,
day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
(2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
GROUP BY year,month;
```

Which returns:

```
+-----+-----+-----+
| year | month | days |
+-----+-----+-----+
| 2000 | 01 | 3 |
| 2000 | 02 | 2 |
+-----+-----+-----+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9. Using [AUTO_INCREMENT](#)

The [AUTO_INCREMENT](#) attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
id MEDIUMINT NOT NULL AUTO_INCREMENT,
```

```

    name CHAR(30) NOT NULL,
    PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
  ('dog'),('cat'),('penguin'),
  ('lax'),('whale'),('ostrich');

SELECT * FROM animals;

```

Which returns:

```

+-----+-----+
| id | name |
+-----+-----+
| 1 | dog  |
| 2 | cat  |
| 3 | penguin |
| 4 | lax  |
| 5 | whale |
| 6 | ostrich |
+-----+-----+

```

No value was specified for the `AUTO_INCREMENT` column, so MySQL assigned sequence numbers automatically. You can also explicitly assign `NULL` or 0 to the column to generate sequence numbers.

You can retrieve the most recent `AUTO_INCREMENT` value with the `LAST_INSERT_ID()` [1194] SQL function or the `mysql_insert_id()` C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Use the smallest integer data type for the `AUTO_INCREMENT` column that is large enough to hold the maximum sequence value you will need. When the column reaches the upper limit of the data type, the next attempt to generate a sequence number fails. Use the `UNSIGNED` attribute if possible to allow a greater range. For example, if you use `TINYINT`, the maximum permissible sequence number is 127. For `TINYINT UNSIGNED`, the maximum is 255. See [Section 11.2.1, “Integer Types \(Exact Value\) - `INTEGER`, `INT`, `SMALLINT`, `TINYINT`, `MEDIUMINT`, `BIGINT`”](#) for the ranges of all the integer types.



Note

For a multiple-row insert, `LAST_INSERT_ID()` [1194] and `mysql_insert_id()` actually return the `AUTO_INCREMENT` key from the *first* of the inserted rows. This enables multiple-row inserts to be reproduced correctly on other servers in a replication setup.

If the `AUTO_INCREMENT` column is part of multiple indexes, MySQL generates sequence values using the index that begins with the `AUTO_INCREMENT` column, if there is one. For example, if the `animals` table contained indexes `PRIMARY KEY (grp, id)` and `INDEX (id)`, MySQL would ignore the `PRIMARY KEY` for generating sequence values. As a result, the table would contain a single sequence, not a sequence per `grp` value.

To start with an `AUTO_INCREMENT` value other than 1, set that value with `CREATE TABLE` or `ALTER TABLE`, like this:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

InnoDB Notes

For `InnoDB` tables, be careful if you modify the column containing the auto-increment value in the middle of a sequence of `INSERT` statements. For example, if you use an `UPDATE` statement to put a new, larger value in the auto-increment column, a subsequent `INSERT` could encounter a “Duplicate entry” error.

The test whether an auto-increment value is already present occurs if you do a `DELETE` followed by more `INSERT` statements, or when you `COMMIT` the transaction, but not after an `UPDATE` statement.

MyISAM Notes

For `MyISAM` tables, you can specify `AUTO_INCREMENT` on a secondary column in a multiple-column index. In this case, the generated value for the `AUTO_INCREMENT` column is calculated as `MAX(auto_increment_column) + 1 WHERE prefix=given-prefix [1205]`. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (
  grp ENUM('fish','mammal','bird') NOT NULL,
  id MEDIUMINT NOT NULL AUTO_INCREMENT,
  name CHAR(30) NOT NULL,
  PRIMARY KEY (grp,id)
) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES
  ('mammal','dog'),('mammal','cat'),
  ('bird','penguin'),('fish','lax'),('mammal','whale'),
  ('bird','ostrich');

SELECT * FROM animals ORDER BY grp,id;
```

Which returns:

```
+-----+-----+-----+
| grp   | id   | name  |
+-----+-----+-----+
| fish  | 1    | lax   |
| mammal| 1    | dog   |
| mammal| 2    | cat   |
| mammal| 3    | whale |
| bird  | 1    | penguin|
| bird  | 2    | ostrich|
+-----+-----+-----+
```

In this case (when the `AUTO_INCREMENT` column is part of a multiple-column index), `AUTO_INCREMENT` values are reused if you delete the row with the biggest `AUTO_INCREMENT` value in any group. This happens even for `MyISAM` tables, for which `AUTO_INCREMENT` values normally are not reused.

Further Reading

More information about `AUTO_INCREMENT` is available here:

- How to assign the `AUTO_INCREMENT` attribute to a column: [Section 13.1.17, “CREATE TABLE Syntax”](#), and [Section 13.1.7, “ALTER TABLE Syntax”](#).
- How `AUTO_INCREMENT` behaves depending on the `NO_AUTO_VALUE_ON_ZERO [613]` SQL mode: [Section 5.1.7, “Server SQL Modes”](#).
- How to use the `LAST_INSERT_ID()` [\[1194\]](#) function to find the row that contains the most recent `AUTO_INCREMENT` value: [Section 12.14, “Information Functions”](#).
- Setting the `AUTO_INCREMENT` value to be used: [Section 5.1.4, “Server System Variables”](#).
- `AUTO_INCREMENT` and replication: [Section 16.4.1.1, “Replication and AUTO_INCREMENT”](#).
- Server-system variables related to `AUTO_INCREMENT` (`auto_increment_increment [1865]` and `auto_increment_offset [1868]`) that can be used for replication: [Section 5.1.4, “Server System Variables”](#).

3.7. Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
    "%h",%{Y%m%d%H%M%S}t,%>s,"%b",\%{Content-Type}o\," \
    \%U",\%{Referer}i\,"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the `LogFormat` line writes to the log file.

Chapter 4. MySQL Programs

Table of Contents

4.1. Overview of MySQL Programs	232
4.2. Using MySQL Programs	236
4.2.1. Invoking MySQL Programs	236
4.2.2. Connecting to the MySQL Server	237
4.2.3. Specifying Program Options	241
4.2.4. Setting Environment Variables	253
4.3. MySQL Server and Server-Startup Programs	254
4.3.1. <code>mysqld</code> — The MySQL Server	254
4.3.2. <code>mysqld_safe</code> — MySQL Server Startup Script	254
4.3.3. <code>mysql.server</code> — MySQL Server Startup Script	259
4.3.4. <code>mysqld_multi</code> — Manage Multiple MySQL Servers	260
4.4. MySQL Installation-Related Programs	264
4.4.1. <code>comp_err</code> — Compile MySQL Error Message File	264
4.4.2. <code>mysqlbug</code> — Generate Bug Report	265
4.4.3. <code>mysql_install_db</code> — Initialize MySQL Data Directory	265
4.4.4. <code>mysql_plugin</code> — Configure MySQL Server Plugins	267
4.4.5. <code>mysql_secure_installation</code> — Improve MySQL Installation Security	269
4.4.6. <code>mysql_tzinfo_to_sql</code> — Load the Time Zone Tables	269
4.4.7. <code>mysql_upgrade</code> — Check and Upgrade MySQL Tables	270
4.5. MySQL Client Programs	272
4.5.1. <code>mysql</code> — The MySQL Command-Line Tool	272
4.5.2. <code>mysqladmin</code> — Client for Administering a MySQL Server	293
4.5.3. <code>mysqlcheck</code> — A Table Maintenance Program	300
4.5.4. <code>mysqldump</code> — A Database Backup Program	307
4.5.5. <code>mysqlimport</code> — A Data Import Program	325
4.5.6. <code>mysqlshow</code> — Display Database, Table, and Column Information	330
4.5.7. <code>mysqlslap</code> — Load Emulation Client	334
4.6. MySQL Administrative and Utility Programs	341
4.6.1. <code>innochecksum</code> — Offline InnoDB File Checksum Utility	341
4.6.2. <code>myisam_ftdump</code> — Display Full-Text Index information	342
4.6.3. <code>myisamchk</code> — MyISAM Table-Maintenance Utility	343
4.6.4. <code>myisamlog</code> — Display MyISAM Log File Contents	360
4.6.5. <code>myisampack</code> — Generate Compressed, Read-Only MyISAM Tables	361
4.6.6. <code>mysqlaccess</code> — Client for Checking Access Privileges	367
4.6.7. <code>mysqlbinlog</code> — Utility for Processing Binary Log Files	370
4.6.8. <code>mysqldumpslow</code> — Summarize Slow Query Log Files	384
4.6.9. <code>mysqlhotcopy</code> — A Database Backup Program	386
4.6.10. <code>mysql_convert_table_format</code> — Convert Tables to Use a Given Storage Engine	389
4.6.11. <code>mysql_find_rows</code> — Extract SQL Statements from Files	390
4.6.12. <code>mysql_fix_extensions</code> — Normalize Table File Name Extensions	391
4.6.13. <code>mysql_setpermission</code> — Interactively Set Permissions in Grant Tables	391
4.6.14. <code>mysql_waitpid</code> — Kill Process and Wait for Its Termination	392
4.6.15. <code>mysql_zap</code> — Kill Processes That Match a Pattern	393
4.7. MySQL Program Development Utilities	393
4.7.1. <code>msql2mysql</code> — Convert mSQL Programs for Use with MySQL	394
4.7.2. <code>mysql_config</code> — Get Compile Options for Compiling Clients	394
4.7.3. <code>my_print_defaults</code> — Display Options from Option Files	395
4.7.4. <code>resolve_stack_dump</code> — Resolve Numeric Stack Trace Dump to Symbols	396

4.8. Miscellaneous Programs	397
4.8.1. <code>pererror</code> — Explain Error Codes	397
4.8.2. <code>replace</code> — A String-Replacement Utility	397
4.8.3. <code>resolveip</code> — Resolve Host name to IP Address or Vice Versa	398

This chapter provides a brief overview of the MySQL command-line programs provided by Oracle Corporation. It also discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Finally, the chapter provides more detailed descriptions of individual programs, including which options they recognize.

4.1. Overview of MySQL Programs

There are many different programs in a MySQL installation. This section provides a brief overview of them. Later sections provide a more detailed description of each one, with the exception of MySQL Cluster programs. Each program's description indicates its invocation syntax and the options that it supports. [Chapter 17, MySQL Cluster NDB 7.2](#), describes programs specific to MySQL Cluster.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see [Chapter 2, Installing and Upgrading MySQL](#), for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install an additional package.

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a description of the program's different options. For example, try `mysql --help`.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. See [Section 4.2, "Using MySQL Programs"](#), for general information on invoking programs and specifying program options.

The MySQL server, `mysqld`, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that assist you in starting and stopping the server:

- `mysqld`

The SQL daemon (that is, the MySQL server). To use client programs, `mysqld` must be running, because clients gain access to databases by connecting to the server. See [Section 4.3.1, "mysqld — The MySQL Server"](#).

- `mysqld_safe`

A server startup script. `mysqld_safe` attempts to start `mysqld`. See [Section 4.3.2, "mysqld_safe — MySQL Server Startup Script"](#).

- `mysql.server`

A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes `mysqld_safe` to start the MySQL server. See [Section 4.3.3, "mysql.server — MySQL Server Startup Script"](#).

- `mysqld_multi`

A server startup script that can start or stop multiple servers installed on the system. See [Section 4.3.4, "mysqld_multi — Manage Multiple MySQL Servers"](#).

Several programs perform setup operations during MySQL installation or upgrading:

- `comp_err`

This program is used during the MySQL build/installation process. It compiles error message files from the error source files. See [Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”](#).

- `mysql_install_db`

This script creates the MySQL database and initializes the grant tables with default privileges. It is usually executed only once, when first installing MySQL on a system. See [Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”](#), [Section 2.11.1, “Unix Postinstallation Procedures”](#), and [Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”](#).

- `mysql_plugin`

This program configures MySQL server plugins. See [Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”](#).

- `mysql_secure_installation`

This program enables you to improve the security of your MySQL installation. SQL. See [Section 4.4.5, “`mysql_secure_installation` — Improve MySQL Installation Security”](#).

- `mysql_tzinfo_to_sql`

This program loads the time zone tables in the `mysql` database using the contents of the host system `zoneinfo` database (the set of files describing time zones). SQL. See [Section 4.4.6, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”](#).

- `mysql_upgrade`

This program is used after a MySQL upgrade operation. It checks tables for incompatibilities and repairs them if necessary, and updates the grant tables with any changes that have been made in newer versions of MySQL. See [Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”](#).

MySQL client programs that connect to the MySQL server:

- `mysql`

The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See [Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”](#).

- `mysqladmin`

A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. `mysqladmin` can also be used to retrieve version, process, and status information from the server. See [Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”](#).

- `mysqlcheck`

A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See [Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”](#).

- `mysqldump`

A client that dumps a MySQL database into a file as SQL, text, or XML. See [Section 4.5.4, “`mysqldump` — A Database Backup Program”](#).

- `mysqlimport`

A client that imports text files into their respective tables using `LOAD DATA INFILE`. See [Section 4.5.5, “mysqlimport — A Data Import Program”](#).

- `mysqlshow`

A client that displays information about databases, tables, columns, and indexes. See [Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”](#).

- `mysqlslap`

A client that is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients are accessing the server. See [Section 4.5.7, “mysqlslap — Load Emulation Client”](#).

MySQL administrative and utility programs:

- `innochecksum`

An offline InnoDB offline file checksum utility. See [Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”](#).

- `myisam_ftdump`

A utility that displays information about full-text indexes in MyISAM tables. See [Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”](#).

- `myisamchk`

A utility to describe, check, optimize, and repair MyISAM tables. See [Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

- `myisamlog, isamlog`

A utility that processes the contents of a MyISAM log file. See [Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”](#).

- `myisampack`

A utility that compresses MyISAM tables to produce smaller read-only tables. See [Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#).

- `mysqlaccess`

A script that checks the access privileges for a host name, user name, and database combination. See [Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”](#).

- `mysqlbinlog`

A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See [Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”](#).

- `mysqldumpslow`

A utility to read and summarize the contents of a slow query log. See [Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”](#).

- `mysqlhotcopy`

A utility that quickly makes backups of `MyISAM` tables while the server is running. See [Section 4.6.9, “mysqlhotcopy — A Database Backup Program”](#).

- `mysql_convert_table_format`

A utility that converts tables in a database to use a given storage engine. See [Section 4.6.10, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”](#).

- `mysql_find_rows`

A utility that reads files containing SQL statements (such as update logs) and extracts statements that match a given regular expression. See [Section 4.6.11, “mysql_find_rows — Extract SQL Statements from Files”](#).

- `mysql_fix_extensions`

A utility that converts the extensions for `MyISAM` table files to lowercase. This can be useful after transferring the files from a system with case-insensitive file names to a system with case-sensitive file names. See [Section 4.6.12, “mysql_fix_extensions — Normalize Table File Name Extensions”](#).

- `mysql_setpermission`

A utility for interactively setting permissions in the MySQL grant tables. See [Section 4.6.13, “mysql_setpermission — Interactively Set Permissions in Grant Tables”](#).

- `mysql_waitpid`

A utility that kills the process with a given process ID. See [Section 4.6.14, “mysql_waitpid — Kill Process and Wait for Its Termination”](#).

- `mysql_zap`

A utility that kills processes that match a pattern. See [Section 4.6.15, “mysql_zap — Kill Processes That Match a Pattern”](#).

MySQL program-development utilities:

- `msql2mysql`

A shell script that converts `mSQL` programs to MySQL. It doesn't handle every case, but it gives a good start when converting. See [Section 4.7.1, “msql2mysql — Convert mSQL Programs for Use with MySQL”](#).

- `mysql_config`

A shell script that produces the option values needed when compiling MySQL programs. See [Section 4.7.2, “mysql_config — Get Compile Options for Compiling Clients”](#).

- `my_print_defaults`

A utility that shows which options are present in option groups of option files. See [Section 4.7.3, “my_print_defaults — Display Options from Option Files”](#).

- `resolve_stack_dump`

A utility program that resolves a numeric stack trace dump to symbols. See [Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”](#).

Miscellaneous utilities:

- `perror`

A utility that displays the meaning of system or MySQL error codes. See [Section 4.8.1, “perror — Explain Error Codes”](#).

- `replace`

A utility program that performs string replacement in the input text. See [Section 4.8.2, “replace — A String-Replacement Utility”](#).

- `resolveip`

A utility program that resolves a host name to an IP address or vice versa. See [Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”](#).

Oracle Corporation also provides the [MySQL Workbench](#) GUI tool, which is used to administer MySQL servers and databases, to create, execute, and evaluate queries, and to migrate schemas and data from other relational database management systems for use with MySQL. Additional GUI tools include [MySQL Notifier for Microsoft Windows](#) and [MySQL for Excel](#).

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables.

Environment Variable	Meaning
<code>MYSQL_UNIX_PORT</code>	The default Unix socket file; used for connections to <code>localhost</code>
<code>MYSQL_TCP_PORT</code>	The default port number; used for TCP/IP connections
<code>MYSQL_PWD</code>	The default password
<code>MYSQL_DEBUG</code>	Debug trace options when debugging
<code>TMPDIR</code>	The directory where temporary tables and files are created

For a full list of environment variables used by MySQL programs, see [Section 2.13, “Environment Variables”](#).

Use of `MYSQL_PWD` is insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#).

4.2. Using MySQL Programs

4.2.1. Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. “`shell>`” represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are `$` for `sh`, `ksh`, or `bash`, `%` for `csh` or `tcsh`, and `C:\>` for the Windows `command.com` or `cmd.exe` command interpreters.

```
shell> mysql --user=root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump -u root personnel
```

Arguments that begin with a single or double dash (“-”, “--”) specify program options. Options typically indicate the type of connection a program should make to the server or affect its operational mode. Option syntax is described in [Section 4.2.3, “Specifying Program Options”](#).

Nonoption arguments (arguments with no leading dash) provide additional information to the program. For example, the `mysql` program interprets the first nonoption argument as a database name, so the command `mysql --user=root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program supports and describe the meaning of any additional nonoption arguments.

Some options are common to a number of programs. The most frequently used of these are the `--host` [239] (or `-h`), `--user` [240] (or `-u`), and `--password` [239] (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the user name and password of your MySQL account. All MySQL client programs understand these options; they enable you to specify which server to connect to and the account to use on that server. Other connection options are `--port` [239] (or `-P`) to specify a TCP/IP port number and `--socket` [240] (or `-S`) to specify a Unix socket file on Unix (or named pipe name on Windows). For more information on options that specify connection options, see [Section 4.2.2, “Connecting to the MySQL Server”](#).

You may find it necessary to invoke MySQL programs using the path name to the `bin` directory in which they are installed. This is likely to be the case if you get a “program not found” error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the path name of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. For example, if `mysql` is installed in `/usr/local/mysql/bin`, you can run the program by invoking it as `mysql`, and it is not necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in [Section 4.2.4, “Setting Environment Variables”](#).) After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.2.2. Connecting to the MySQL Server

For a client program to be able to connect to the MySQL server, it must use the proper connection parameters, such as the name of the host where the server is running and the user name and password of your MySQL account. Each connection parameter has a default value, but you can override them as necessary using program options specified either on the command line or in an option file.

The examples here use the `mysql` client program, but the principles apply to other clients such as `mysqldump`, `mysqladmin`, or `mysqlshow`.

This command invokes `mysql` without specifying any connection parameters explicitly:

```
shell> mysql
```

Because there are no parameter options, the default values apply:

- The default host name is `localhost`. On Unix, this has a special meaning, as described later.
- The default user name is `ODBC` on Windows or your Unix login name on Unix.
- No password is sent if neither `-p` nor `--password` [239] is given.
- For `mysql`, the first nonoption argument is taken as the name of the default database. If there is no such option, `mysql` does not select a default database.

To specify the host name and user name explicitly, as well as a password, supply appropriate options on the command line:

```
shell> mysql --host=localhost --user=myname --password=mypass mydb
shell> mysql -h localhost -u myname -pmypass mydb
```

For password options, the password value is optional:

- If you use a `-p` or `--password` [239] option and specify the password value, there must be *no space* between `-p` or `--password=` [239] and the password following it.
- If you use a `-p` or `--password` [239] option but do not specify the password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line. Other users on your system may be able to see a password specified on the command line by executing a command such as `ps auxw`. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#).

As just mentioned, including the password value on the command line can be a security risk. To avoid this problem, specify the `--password` or `-p` option without any following password value:

```
shell> mysql --host=localhost --user=myname --password mydb
shell> mysql -h localhost -u myname -p mydb
```

When the password option has no password value, the client program prints a prompt and waits for you to enter the password. (In these examples, `mydb` is *not* interpreted as a password because it is separated from the preceding password option by a space.)

On some systems, the library routine that MySQL uses to prompt for a password automatically limits the password to eight characters. That is a problem with the system library, not with MySQL. Internally, MySQL does not have any limit for the length of the password. To work around the problem, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

On Unix, MySQL programs treat the host name `localhost` specially, in a way that is likely different from what you expect compared to other network-based programs. For connections to `localhost`, MySQL programs attempt to connect to the local server by using a Unix socket file. This occurs even if a `--port` [239] or `-P` option is given to specify a port number. To ensure that the client makes a TCP/IP connection to the local server, use `--host` [239] or `-h` to specify a host name value of `127.0.0.1`, or the IP address or name of the local server. You can also specify the connection protocol explicitly, even for `localhost`, by using the `--protocol=TCP` [239] option. For example:

```
shell> mysql --host=127.0.0.1
shell> mysql --protocol=TCP
```

The `--protocol` [239] option enables you to establish a particular type of connection even when the other options would normally default to some other protocol.

If the server is configured to accept IPv6 connections, client can connect over IPv6 using `--host>:::1` [239]. See [Section 5.1.9, “IPv6 Support”](#).

On Windows, you can force a MySQL client to use a named-pipe connection by specifying the `--pipe` [239] or `--protocol=PIPE` [239] option, or by specifying `.` (period) as the host name. If named-pipe connections are not enabled, an error occurs. Use the `--socket` [240] option to specify the name of the pipe if you do not want to use the default pipe name.

Connections to remote servers always use TCP/IP. This command connects to the server running on `remote.example.com` using the default port number (3306):

```
shell> mysql --host=remote.example.com
```

To specify a port number explicitly, use the `--port` [239] or `-P` option:

```
shell> mysql --host=remote.example.com --port=13306
```

You can specify a port number for connections to a local server, too. However, as indicated previously, connections to `localhost` on Unix will use a socket file by default. You will need to force a TCP/IP connection as already described or any option that specifies a port number will be ignored.

For this command, the program uses a socket file on Unix and the `--port [239]` option is ignored:

```
shell> mysql --port=13306 --host=localhost
```

To cause the port number to be used, invoke the program in either of these ways:

```
shell> mysql --port=13306 --host=127.0.0.1
shell> mysql --port=13306 --protocol=TCP
```

The following list summarizes the options that can be used to control how client programs connect to the server:

- `--host=host_name [239]`, `-h host_name`

The host where the server is running. The default value is `localhost`.

- `--password[=pass_val] [239]`, `-p[pass_val]`

The password of the MySQL account. As described earlier, the password value is optional, but if given, there must be *no space* between `-p` or `--password= [239]` and the password following it. The default is to send no password.

- `--pipe [239]`, `-W`

On Windows, connect to the server using a named pipe. The server must be started with the `--enable-named-pipe [442]` option to enable named-pipe connections.

- `--port=port_num [239]`, `-P port_num`

The port number to use for the connection, for connections made using TCP/IP. The default port number is 3306.

- `--protocol={TCP | SOCKET | PIPE | MEMORY} [239]`

This option explicitly specifies a protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For example, connections on Unix to `localhost` are made using a Unix socket file by default:

```
shell> mysql --host=localhost
```

To force a TCP/IP connection to be used instead, specify a `--protocol [239]` option:

```
shell> mysql --host=localhost --protocol=TCP
```

The following table shows the permissible `--protocol [239]` option values and indicates the platforms on which each value may be used. The values are not case sensitive.

<code>--protocol [239]</code> Value	Connection Protocol	Permissible Operating Systems
TCP	TCP/IP connection to local or remote server	All

<code>--protocol [239]</code> Value	Connection Protocol	Permissible Operating Systems
SOCKET	Unix socket file connection to local server	Unix only
PIPE	Named-pipe connection to local or remote server	Windows only
MEMORY	Shared-memory connection to local server	Windows only

- `--shared-memory-base-name=name [240]`

On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

The server must be started with the `--shared-memory [460]` option to enable shared-memory connections.

- `--socket=file_name [240], -S file_name`

On Unix, the name of the Unix socket file to use, for connections made using a named pipe to a local server. The default Unix socket file name is `/tmp/mysql.sock`.

On Windows, the name of the named pipe to use, for connections to a local server. The default Windows pipe name is `MySQL`. The pipe name is not case sensitive.

The server must be started with the `--enable-named-pipe [442]` option to enable named-pipe connections.

- `--ssl*`

Options that begin with `--ssl [749]` are used for establishing a secure connection to the server using SSL, if the server is configured with SSL support. For details, see [Section 6.3.8.4, “SSL Command Options”](#).

- `--user=user_name [240], -u user_name`

The user name of the MySQL account you want to use. The default user name is `ODBC` on Windows or your Unix login name on Unix.

It is possible to specify different default values to be used when you make a connection so that you need not enter them on the command line each time you invoke a client program. This can be done in a couple of ways:

- You can specify connection parameters in the `[client]` section of an option file. The relevant section of the file might look like this:

```
[client]
host=host_name
user=user_name
password=your_pass
```

[Section 4.2.3.3, “Using Option Files”](#), discusses option files further.

- You can specify some connection parameters using environment variables. The host can be specified for `mysql` using `MYSQL_HOST`. The MySQL user name can be specified using `USER` (this is for Windows only). The password can be specified using `MYSQL_PWD`, although this is insecure; see [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). For a list of variables, see [Section 2.13, “Environment Variables”](#).

4.2.3. Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is common for options that apply to a specific invocation of the program.
- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables (see [Section 4.2.4, “Setting Environment Variables”](#)). This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose, but [Section 5.3.3, “Running Multiple MySQL Instances on Unix”](#), discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for the server and for client programs.

Options are processed in order, so if an option is specified multiple times, the last occurrence takes precedence. The following command causes `mysql` to connect to the server running on `localhost`:

```
shell> mysql -h example.com -h localhost
```

If conflicting or related options are given, later options take precedence over earlier options. The following command runs `mysql` in “no column names” mode:

```
shell> mysql --column-names --skip-column-names
```

MySQL programs determine which options are given first by examining environment variables, then by reading option files, and then by checking the command line. This means that environment variables have the lowest precedence and command-line options the highest.

You can take advantage of the way that MySQL programs process options by specifying default option values for a program in an option file. That enables you to avoid typing them each time you run the program while enabling you to override the defaults if necessary by using command-line options.

An option can be specified by writing it in full or as any unambiguous prefix. For example, the `--compress` [314] option can be given to `mysqldump` as `--compr`, but not as `--comp` because the latter is ambiguous:

```
shell> mysqldump --comp
mysqldump: ambiguous option '--comp' (compatible, compress)
```

Be aware that the use of option prefixes can cause problems in the event that new options are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

4.2.3.1. Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it is a short form or long form of the option name. Many options have both short and long forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.
- Option names are case sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)

- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` [239] indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an “=” sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=pass_val` [239] or as `--password` [239]. In the latter case (with no password value given), the program prompts you for the password. The password option also may be given in short form as `-ppass_val` or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*. The reason for this is that if a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

```
shell> mysql -ptest
shell> mysql -p test
```

The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.

- Within option names, dash (“-”) and underscore (“_”) may be used interchangeably. For example, `--skip-grant-tables` [461] and `--skip_grant_tables` [461] are equivalent. (However, the leading dashes cannot be given as underscores.)
- For options that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024² or 1024³. For example, the following command tells `mysqladmin` to ping the server 1024 times, sleeping 10 seconds between each ping:

```
mysql> mysqladmin --count=1K --sleep=10 ping
```

Option values that contain spaces must be quoted when given on the command line. For example, the `--execute` [278] (or `-e`) option can be used with `mysql` to pass SQL statements to the server. When this option is used, `mysql` executes the statements in the option value and exits. The statements must be enclosed by quotation marks. For example, you can use the following command to obtain a list of user accounts:

```
mysql> mysql -u root -p --execute="SELECT User, Host FROM mysql.user"
Enter password: *****
+-----+-----+
| User | Host |
+-----+-----+
| root | gigan |
| root | gigan |
| jon  | localhost |
| root | localhost |
+-----+-----+
shell>
```

Note that the long form (`--execute` [278]) is followed by an equal sign (=).

If you wish to use quoted values within a statement, you will either need to escape the inner quotation marks, or use a different type of quotation marks within the statement from those used to quote the statement itself. The capabilities of your command processor dictate your choices for whether you can use single or double quotation marks and the syntax for escaping quote characters. For example, if your command processor supports quoting with single or double quotation marks, you can use double quotation marks around the statement, and single quotation marks for any quoted values within the statement.

Multiple SQL statements may be passed in the option value on the command line, separated by semicolons:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: *****
+-----+
| VERSION() |
+-----+
| 5.1.5-alpha-log |
+-----+
+-----+
| NOW() |
+-----+
| 2006-01-05 21:19:04 |
+-----+
```

4.2.3.2. Program Option Modifiers

Some options are “boolean” and control behavior that can be turned on or off. For example, the `mysql` client supports a `--column-names` [277] option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

As of MySQL 5.5.10, the values `ON`, `TRUE`, `OFF`, and `FALSE` are also recognized for boolean options (not case sensitive).

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

`mysqld` enables a limit to be placed on how large client programs can set dynamic system variables. To do this, use a `--maximum` prefix with the variable name. For example, `--maximum-query_cache_size=4M` prevents any client from making the query cache size larger than 4MB.

4.2.3.3. Using Option Files

Most MySQL programs can read startup options from option files (also sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be

entered on the command line each time you run a program. For the MySQL server, MySQL provides a number of [preconfigured option files](#).

To determine whether a program reads option files, invoke it with the `--help` option. (For `mysqld`, use `--verbose` [467] and `--help` [434].) If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

On Windows, MySQL programs read startup options from the following files, in the specified order (top items are used first).

File Name	Purpose
<code>WINDIR\my.ini</code> , <code>WINDIR\my.cnf</code>	Global options
<code>C:\my.ini</code> , <code>C:\my.cnf</code>	Global options
<code>INSTALLDIR\my.ini</code> , <code>INSTALLDIR\my.cnf</code>	Global options
<code>defaults-extra-file</code>	The file specified with <code>--defaults-extra-file=path</code> [248], if any

`WINDIR` represents the location of your Windows directory. This is commonly `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

`INSTALLDIR` represents the MySQL installation directory. This is typically `C:\PROGRAMDIR\MySQL\MySQL 5.5 Server` where `PROGRAMDIR` represents the programs directory (usually `Program Files` on English-language versions of Windows), when MySQL 5.5 has been installed using the installation and configuration wizards. See [Section 2.3.3, "Installing MySQL on Microsoft Windows Using MySQL Installer"](#).

On Unix, Linux and Mac OS X, MySQL programs read startup options from the following files, in the specified order (top items are used first).

File Name	Purpose
<code>/etc/my.cnf</code>	Global options
<code>/etc/mysql/my.cnf</code>	Global options
<code>SYSCONFDIR/my.cnf</code>	Global options
<code>\$MYSQL_HOME/my.cnf</code>	Server-specific options
<code>defaults-extra-file</code>	The file specified with <code>--defaults-extra-file=path</code> [248], if any
<code>~/my.cnf</code>	User-specific options

`~` represents the current user's home directory (the value of `$HOME`).

`SYSCONFDIR` represents the directory specified with the `SYSCONFDIR` [153] option to `CMake` when MySQL was built. By default, this is the `etc` directory located under the compiled-in installation directory.

`MYSQL_HOME` is an environment variable containing the path to the directory in which the server-specific `my.cnf` file resides. If `MYSQL_HOME` is not set and you start the server using the `mysqld_safe` program, `mysqld_safe` attempts to set `MYSQL_HOME` as follows:

- Let `BASEDIR` and `DATADIR` represent the path names of the MySQL base directory and data directory, respectively.

- If there is a `my.cnf` file in `DATADIR` but not in `BASEDIR`, `mysqld_safe` sets `MYSQL_HOME` to `DATADIR`.
- Otherwise, if `MYSQL_HOME` is not set and there is no `my.cnf` file in `DATADIR`, `mysqld_safe` sets `MYSQL_HOME` to `BASEDIR`.

In MySQL 5.5, use of `DATADIR` as the location for `my.cnf` is deprecated.

Typically, `DATADIR` is `/usr/local/mysql/data` for a binary installation or `/usr/local/var` for a source installation. Note that this is the data directory location that was specified at configuration time, not the one specified with the `--datadir` [439] option when `mysqld` starts. Use of `--datadir` [439] at runtime has no effect on where the server looks for option files, because it looks for them before processing any options.

MySQL looks for option files in the order just described and reads any that exist. If an option file that you want to use does not exist, create it with a plain text editor.

If multiple instances of a given option are found, the last instance takes precedence. There is one exception: For `mysqld`, the *first* instance of the `--user` [466] option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.



Note

On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional as a security measure.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the `--help` option.

The syntax for specifying options in an option file is similar to command-line syntax (see [Section 4.2.3.1, “Using Options on the Command Line”](#)). However, in an option file, you omit the leading two dashes from the option name and you specify only one option per line. For example, `--quick` and `--host=localhost` on the command line should be specified as `quick` and `host=localhost` on separate lines in an option file. To specify an option of the form `--loose-opt_name` in an option file, write it as `loose-opt_name`.

Empty lines in option files are ignored. Nonempty lines can take any of the following forms:

- `#comment, ;comment`

Comment lines start with “#” or “;”. A “#” comment can start in the middle of a line as well.

- `[group]`

`group` is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given. Option group names are not case sensitive.

- `opt_name`

This is equivalent to `--opt_name` on the command line.

- `opt_name=value`

This is equivalent to `--opt_name=value` on the command line. In an option file, you can have spaces around the “=” character, something that is not true on the command line. You can optionally enclose the value within single quotation marks or double quotation marks, which is useful if the value contains a “#” comment character.

Leading and trailing spaces are automatically deleted from option names and values.

You can use the escape sequences “\b”, “\t”, “\n”, “\r”, “\\”, and “\s” in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters. The escaping rules in option files are:

- If a backslash is followed by a valid escape sequence character, the sequence is converted to the character represented by the sequence. For example, “\s” is converted to a space.
- If a backslash is not followed by a valid escape sequence character, it remains unchanged. For example, “\s” is retained as is.

The preceding rules mean that a literal backslash can be given as “\\”, or as “\” if it is not followed by a valid escape sequence character.

The rules for escape sequences in option files differ slightly from the rules for escape sequences in string literals in SQL statements. In the latter context, if “x” is not a value escape sequence character, “\x” becomes “x” rather than “\x”. See [Section 9.1.1, “String Literals”](#).

The escaping rules for option file values are especially pertinent for Windows path names, which use “\” as a path name separator. A separator in a Windows path name must be written as “\\” if it is followed by an escape sequence character. It can be written as “\” or “\” if it is not. Alternatively, “/” may be used in Windows path names and will be treated as “\”. Suppose that you want to specify a base directory of `C:\Program Files\MySQL\MySQL Server 5.5` in an option file. This can be done several ways. Some examples:

```
basedir="C:\Program Files\MySQL\MySQL Server 5.5"
basedir="C:\\Program Files\\MySQL\\MySQL Server 5.5"
basedir="C:/Program Files/MySQL/MySQL Server 5.5"
basedir=C:\\Program\\sFiles\\MySQL\\MySQL\\sServer\\s5.5
```

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the `[mysqld]` and `[mysql]` groups apply to the `mysqld` server and the `mysql` client program, respectively.

The `[client]` option group is read by all client programs (but *not* by `mysqld`). This enables you to specify options that apply to all clients. For example, `[client]` is the perfect group to use to specify the password that you use to connect to the server. (But make sure that the option file is readable and writable only by yourself, so that other people cannot find out your password.) Be sure not to put an option in the `[client]` group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

The preceding option file uses `var_name=value` syntax for the lines that set the `key_buffer_size` [508] and `max_allowed_packet` [519] variables.

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"

[mysql]
no-auto-rehash
connect_timeout=2

[mysqlhotcopy]
interactive_timeout
```

If you want to create option groups that should be read by `mysqld` servers from a specific MySQL release series only, you can do this by using groups with names of `[mysqld-5.1]`, `[mysqld-5.5]`, and so forth. The following group indicates that the `--new` option should be used only by MySQL servers with 5.5.x version numbers:

```
[mysqld-5.5]
new
```

It is possible to use `!include` directives in option files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, use the following directive:

```
!include /home/mydir/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, use this directive:

```
!includedir /home/mydir
```

There is no guarantee about the order in which the option files in the directory will be read.



Note

Currently, any files to be found and included using the `!includedir` directive on Unix operating systems *must* have file names ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Write the contents of an included option file like any other option file. That is, it should contain groups of options, each preceded by a `[group]` line that indicates the program to which the options apply.

While an included file is being processed, only those options in groups that the current program is looking for are used. Other groups are ignored. Suppose that a `my.cnf` file contains this line:

```
!include /home/mydir/myopt.cnf
```

And suppose that `/home/mydir/myopt.cnf` looks like this:

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

If `my.cnf` is processed by `mysqld`, only the `[mysqld]` group in `/home/mydir/myopt.cnf` is used. If the file is processed by `mysqladmin`, only the `[mysqladmin]` group is used. If the file is processed by any other program, no options in `/home/mydir/myopt.cnf` are used.

The `!includedir` directive is processed similarly except that all option files in the named directory are read.

4.2.3.4. Command-Line Options that Affect Option-File Handling

Most MySQL programs that support option files handle the following options. They affect option-file handling, so they must be given on the command line and not in an option file. To work properly, each of these options must immediately follow the command name, with these exceptions:

- `--print-defaults` [248] may be used immediately after `--defaults-file` [248] or `--defaults-extra-file` [248].
- On Windows, if the `--defaults-file` [248] and `--install` options are given, `--install` option must be first. See [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

When specifying file names, you should avoid the use of the “~” shell metacharacter because it might not be interpreted as you expect.

- `--defaults-extra-file=file_name` [248]

Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, the program exits with an error. Before MySQL 5.5.8, *file_name* must be the full path name to the file. As of MySQL 5.5.8, the name is interpreted relative to the current directory if given as a relative path name.

- `--defaults-file=file_name` [248]

Use only the given option file. If the file does not exist or is otherwise inaccessible, the program exits with an error. Before MySQL 5.5.8, *file_name* must be the full path name to the file. As of MySQL 5.5.8, the name is interpreted relative to the current directory if given as a relative path name.

- `--defaults-group-suffix=str` [248]

If this option is given, the program reads not only its usual option groups, but also groups with the usual names and a suffix of *str*. For example, the `mysql` client normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` [248] option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

- `--no-defaults` [248]

Do not read any option files. If a program does not start because it is reading unknown options from an option file, `--no-defaults` [248] can be used to prevent the program from reading them.

- `--print-defaults` [248]

Print the program name and all options that it gets from option files.

4.2.3.5. Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime using the `SET` statement. See [Section 13.7.4, “SET Syntax”](#), and [Section 5.1.5, “Using System Variables”](#).

Most of these program variables also can be set at server startup by using the same syntax that applies to specifying program options. For example, `mysql` has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for `mysql` to a value of 16MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase)

to indicate a multiplier of 1024, 1024² or 1024³. (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.)

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M
```

```
[mysqld]
key-buffer-size=512M
```

A variable can be specified by writing it in full or as any unambiguous prefix. For example, the `max_allowed_packet` variable can be set for `mysql` as `--max_a`, but not as `--max` because the latter is ambiguous:

```
shell> mysql --max=1000000
mysql: ambiguous option '--max=1000000' (max_allowed_packet, max_join_size)
```

Be aware that the use of variable prefixes can cause problems in the event that new variables are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```



Note

Before MySQL 4.0.2, the only syntax for setting program variables was `--set-variable=option=value` (or `set-variable=option=value` in option files). Underscores cannot be given as dashes, and the variable name must be specified in full. This syntax is deprecated and was removed in MySQL 5.5.3.

4.2.3.6. Option Defaults, Options Expecting Values, and the = Sign

By convention, long forms of options that assign a value are written with an equals (=) sign, like this:

```
shell> mysql --host=tonfisk --user=jon
```

For options that require a value (that is, not having a default value), the equal sign is not required, and so the following is also valid:

```
shell> mysql --host tonfisk --user jon
```

In both cases, the `mysql` client attempts to connect to a MySQL server running on the host named “tonfisk” using an account with the user name “jon”.

Due to this behavior, problems can occasionally arise when no value is provided for an option that expects one. Consider the following example, where a user connects to a MySQL server running on host `tonfisk` as user `jon`:

```
shell> mysql --host 85.224.35.45 --user jon
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.5.31 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| jon@%          |
+-----+
1 row in set (0.00 sec)
```

Omitting the required value for one of these option yields an error, such as the one shown here:

```
shell> mysql --host 85.224.35.45 --user
mysql: option '--user' requires an argument
```

In this case, `mysql` was unable to find a value following the `--user` [240] option because nothing came after it on the command line. However, if you omit the value for an option that is *not* the last option to be used, you obtain a different error that you may not be expecting:

```
shell> mysql --host --user jon
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

Because `mysql` assumes that any string following `--host` [239] on the command line is a host name, `--host` [239] `--user` [240] is interpreted as `--host=--user` [239], and the client attempts to connect to a MySQL server running on a host named “--user”.

Options having default values always require an equal sign when assigning a value; failing to do so causes an error. For example, the MySQL server `--log-error` [448] option has the default value `host_name.err`, where `host_name` is the name of the host on which MySQL is running. Assume that you are running MySQL on a computer whose host name is “tonfisk”, and consider the following invocation of `mysqld_safe`:

```
shell> mysqld_safe &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

After shutting down the server, restart it as follows:

```
shell> mysqld_safe --log-error &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

The result is the same, since `--log-error` [256] is not followed by anything else on the command line, and it supplies its own default value. (The `&` character tells the operating system to run MySQL in the

background; it is ignored by MySQL itself.) Now suppose that you wish to log errors to a file named `my-errors.err`. You might try starting the server with `--log-error my-errors`, but this does not have the intended effect, as shown here:

```
shell> mysqld_safe --log-error my-errors &
[1] 31357
shell> 080111 22:53:31 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080111 22:53:32 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended

[1]+  Done                  ./mysqld_safe --log-error my-errors
```

The server attempted to start using `/usr/local/mysql/var/tonfisk.err` as the error log, but then shut down. Examining the last few lines of this file shows the reason:

```
shell> tail /usr/local/mysql/var/tonfisk.err
080111 22:53:32 InnoDB: Started; log sequence number 0 46409
/usr/local/mysql/libexec/mysqld: Too many arguments (first extra is 'my-errors').
Use --verbose --help to get a list of available options
080111 22:53:32 [ERROR] Aborting

080111 22:53:32 InnoDB: Starting shutdown...
080111 22:53:34 InnoDB: Shutdown completed; log sequence number 0 46409
080111 22:53:34 [Note] /usr/local/mysql/libexec/mysqld: Shutdown complete

080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended
```

Because the `--log-error [256]` option supplies a default value, you must use an equal sign to assign a different value to it, as shown here:

```
shell> mysqld_safe --log-error=my-errors &
[1] 31437
shell> 080111 22:54:15 mysqld_safe Logging to '/usr/local/mysql/var/my-errors.err'.
080111 22:54:15 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var

shell>
```

Now the server has been started successfully, and is logging errors to the file `/usr/local/mysql/var/my-errors.err`.

Similar issues can arise when specifying option values in option files. For example, consider a `my.cnf` file that contains the following:

```
[mysql]

host
user
```

When the `mysql` client reads this file, these entries are parsed as `--host [278] --user [282]` or `--host=--user [278]`, with the result shown here:

```
shell> mysql
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

However, in option files, an equal sign is not assumed. Suppose the `my.cnf` file is as shown here:

```
[mysql]

user jon
```

Trying to start `mysql` in this case causes a different error:

```
shell> mysql
```

```
mysql: unknown option '--user jon'
```

A similar error would occur if you were to write `host tonfisk` in the option file rather than `host=tonfisk`. Instead, you must use the equal sign:

```
[mysql]
user=jon
```

Now the login attempt succeeds:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.5.31 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+-----+
| USER()          |
+-----+
| jon@localhost   |
+-----+
1 row in set (0.00 sec)
```

This is not the same behavior as with the command line, where the equals sign is not required:

```
shell> mysql --user jon --host tonfisk
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.5.31 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+-----+
| USER()          |
+-----+
| jon@tonfisk     |
+-----+
1 row in set (0.00 sec)
```

In MySQL 5.5, specifying an option requiring a value without a value in an option file causes the server to abort with an error. Suppose that `my.cnf` contains the following:

```
[mysqld]
log_error
relay_log
relay_log_index
```

This causes the server to fail on startup, as shown here:

```
shell> mysqld_safe &
090514 09:48:39 mysqld_safe Logging to '/home/jon/bin/mysql-5.5/var/tonfisk.err'.
090514 09:48:39 mysqld_safe Starting mysqld daemon with databases from /home/jon/bin/mysql-5.5/var
090514 09:48:39 mysqld_safe mysqld from pid file /home/jon/bin/mysql-5.5/var/tonfisk.pid ended
```

The `--log-error` [448] option does not require an argument; however, the `--relay-log` [1873] option requires one, as shown in the error log (which in the absence of a specified value, defaults to `datadir/hostname.err`):

```
shell> tail -n 3 ../var/tonfisk.err
```

```
090514 09:48:39 mysqld_safe Starting mysqld daemon with databases from /home/jon/bin/mysql-5.5/var
090514 9:48:39 [ERROR] /home/jon/bin/mysql-5.5/libexec/mysqld: option '--relay-log' requires an argument
090514 9:48:39 [ERROR] Aborting
```

This is a change from previous behavior, where the server would have interpreted the last two lines in the example `my.cnf` file as `--relay-log=relay_log_index` and created a relay log file using “`relay_log_index`” as the basename. (Bug #25192)

4.2.4. Setting Environment Variables

Environment variables can be set at the command prompt to affect the current invocation of your command processor, or set permanently to affect future invocations. To set a variable permanently, you can set it in a startup file or by using the interface provided by your system for this purpose. Consult the documentation for your command interpreter for specific details. [Section 2.13, “Environment Variables”](#), lists all environment variables that affect MySQL program operation.

To specify a value for an environment variable, use the syntax appropriate for your command processor. For example, on Windows, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for `sh`, `ksh`, `bash`, `zsh`, and so on) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For `csh` and `tcsh`, use `setenv` to make the shell variable available to the environment:

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, use the interface provided by your system or place the appropriate command or commands in a startup file that your command interpreter reads each time it starts.

On Windows, you can set environment variables using the System Control Panel (under Advanced).

On Unix, typical shell startup files are `.bashrc` or `.bash_profile` for `bash`, or `.tcshrc` for `tcsh`.

Suppose that your MySQL programs are installed in `/usr/local/mysql/bin` and that you want to make it easy to invoke these programs. To do this, set the value of the `PATH` environment variable to include that directory. For example, if your shell is `bash`, add the following line to your `.bashrc` file:

```
PATH=${PATH}:/usr/local/mysql/bin
```

`bash` uses different startup files for login and nonlogin shells, so you might want to add the setting to `.bashrc` for login shells and to `.bash_profile` for nonlogin shells to make sure that `PATH` is set regardless.

If your shell is `tcsh`, add the following line to your `.tcshrc` file:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If the appropriate startup file does not exist in your home directory, create it with a text editor.

After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.3. MySQL Server and Server-Startup Programs

This section describes `mysqld`, the MySQL server, and several programs that are used to start the server.

4.3.1. `mysqld` — The MySQL Server

`mysqld`, also known as MySQL Server, is the main program that does most of the work in a MySQL installation. MySQL Server manages access to the MySQL data directory that contains databases and tables. The data directory is also the default location for other information such as log files and status files.

When MySQL server starts, it listens for network connections from client programs and manages access to databases on behalf of those clients.

The `mysqld` program has many options that can be specified at startup. For a complete list of options, run this command:

```
shell> mysqld --verbose --help
```

MySQL Server also has a set of system variables that affect its operation as it runs. System variables can be set at server startup, and many of them can be changed at runtime to effect dynamic server reconfiguration. MySQL Server also has a set of status variables that provide information about its operation. You can monitor these status variables to access runtime performance characteristics.

For a full description of MySQL Server command options, system variables, and status variables, see [Section 5.1, “The MySQL Server”](#). For information about installing MySQL and setting up the initial configuration, see [Chapter 2, *Installing and Upgrading MySQL*](#).

4.3.2. `mysqld_safe` — MySQL Server Startup Script

`mysqld_safe` is the recommended way to start a `mysqld` server on Unix. `mysqld_safe` adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log file. A description of error logging is given later in this section.

`mysqld_safe` tries to start an executable named `mysqld`. To override the default behavior and specify explicitly the name of the server you want to run, specify a `--mysqld` [257] or `--mysqld-version` [257] option to `mysqld_safe`. You can also use `--ledir` [256] to indicate the directory where `mysqld_safe` should look for the server.

Many of the options to `mysqld_safe` are the same as the options to `mysqld`. See [Section 5.1.3, “Server Command Options”](#).

Options unknown to `mysqld_safe` are passed to `mysqld` if they are specified on the command line, but ignored if they are specified in the `[mysqld_safe]` group of an option file. See [Section 4.2.3.3, “Using Option Files”](#).

`mysqld_safe` reads all options from the `[mysqld]`, `[server]`, and `[mysqld_safe]` sections in option files. For example, if you specify a `[mysqld]` section like this, `mysqld_safe` will find and use the `--log-error` [256] option:

```
[mysqld]
log-error=error.log
```

For backward compatibility, `mysqld_safe` also reads `[safe_mysqld]` sections, although you should rename such sections to `[mysqld_safe]` in MySQL 5.5 installations.

`mysqld_safe` supports the following options. It also reads option files and supports the options for processing them described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.1. `mysqld_safe` Options

Format	Option File	Description
<code>--basedir=path [256]</code>	<code>basedir [256]</code>	The path to the MySQL installation directory
<code>--core-file-size=size [256]</code>	<code>core-file-size [256]</code>	The size of the core file that <code>mysqld</code> should be able to create
<code>--datadir=path [256]</code>	<code>datadir [256]</code>	The path to the data directory
<code>--defaults-extra-file=path [256]</code>	<code>defaults-extra-file [256]</code>	The name of an option file to be read in addition to the usual option files
<code>--defaults-file=file_name [256]</code>	<code>defaults-file [256]</code>	The name of an option file to be read instead of the usual option files
<code>--help [256]</code>		Display a help message and exit
<code>--ledir=path [256]</code>	<code>ledir [256]</code>	Use this option to indicate the path name to the directory where the server is located
<code>--log-error=file_name [256]</code>	<code>log-error [256]</code>	Write the error log to the given file
<code>--malloc-lib=[lib-name] [256]</code>	<code>malloc-lib [256]</code>	Alternative malloc library to use for <code>mysqld</code>
<code>--mysqld=prog_name [257]</code>	<code>mysqld [257]</code>	The name of the server program (in the <code>ledir</code> directory) that you want to start
<code>--mysqld-version=suffix [257]</code>	<code>mysqld-version [257]</code>	This option is similar to the <code>--mysqld</code> option, but you specify only the suffix for the server program name
<code>--nice=priority [257]</code>	<code>nice [257]</code>	Use the <code>nice</code> program to set the server's scheduling priority to the given value
<code>--no-defaults [257]</code>	<code>no-defaults [257]</code>	Do not read any option files
<code>--open-files-limit=count [257]</code>	<code>open-files-limit [257]</code>	The number of files that <code>mysqld</code> should be able to open
<code>--pid-file=file_name [257]</code>	<code>pid-file=file_name [257]</code>	The path name of the process ID file
<code>--port=number [257]</code>	<code>port [257]</code>	The port number that the server should use when listening for TCP/IP connections
<code>--skip-kill-mysqld [257]</code>	<code>skip-kill-mysqld [257]</code>	Do not try to kill stray <code>mysqld</code> processes
<code>--skip-syslog [258]</code>	<code>skip-syslog [258]</code>	Do not write error messages to <code>syslog</code> ; use error log file
<code>--socket=path [258]</code>	<code>socket [258]</code>	The Unix socket file that the server should use when listening for local connections
<code>--syslog [258]</code>	<code>syslog [258]</code>	Write error messages to <code>syslog</code>
<code>--timezone=timezone [258]</code>	<code>timezone [258]</code>	Set the TZ time zone environment variable to the given option value

Format	Option File	Description
<code>--user={user_name} user_id} [258]</code>	<code>user [258]</code>	Run the mysqld server as the user having the name <code>user_name</code> or the numeric user ID <code>user_id</code>

- `--help [256]`

Display a help message and exit.

- `--basedir=path [256]`

The path to the MySQL installation directory.

- `--core-file-size=size [256]`

The size of the core file that `mysqld` should be able to create. The option value is passed to `ulimit -c`.

- `--datadir=path [256]`

The path to the data directory.

- `--defaults-extra-file=path [256]`

The name of an option file to be read in addition to the usual option files. This must be the first option on the command line if it is used. If the file does not exist or is otherwise inaccessible, the server will exit with an error.

- `--defaults-file=file_name [256]`

The name of an option file to be read instead of the usual option files. This must be the first option on the command line if it is used.

- `--ledir=path [256]`

If `mysqld_safe` cannot find the server, use this option to indicate the path name to the directory where the server is located.

- `--log-error=file_name [256]`

Write the error log to the given file. See [Section 5.2.2, “The Error Log”](#).

- `--malloc-lib=[lib_name] [256]`

The name of the library to use for memory allocation instead of the system `malloc()` library. Any library can be used by specifying its path name, but there is a shortcut form to enable use of the `tcmalloc` library that is shipped with binary MySQL distributions for Linux in MySQL 5.5.

The `--malloc-lib [256]` option works by modifying the `LD_PRELOAD` environment value to affect dynamic linking to enable the loader to find the memory-allocation library when `mysqld` runs:

- If the option is not given, or is given without a value (`--malloc-lib= [256]`), `LD_PRELOAD` is not modified and no attempt is made to use `tcmalloc`.
- If the option is given as `--malloc-lib=tcmalloc [256]`, `mysqld_safe` looks for a `tcmalloc` library in `/usr/lib` and then in the MySQL `pkglibdir` location (for example, `/usr/local/mysql/lib` or whatever is appropriate). If `tmalloc` is found, its path name is added to the beginning of the `LD_PRELOAD` value for `mysqld`. If `tcmalloc` is not found, `mysqld_safe` aborts with an error.

- If the option is given as `--malloc-lib=/path/to/some/library` [256], that full path is added to the beginning of the `LD_PRELOAD` value. If the full path points to a nonexistent or unreadable file, `mysqld_safe` aborts with an error.
- For cases where `mysqld_safe` adds a path name to `LD_PRELOAD`, it adds the path to the beginning of any existing value the variable already has.

Linux users can use the `libtcmalloc_minimal.so` included in binary packages by adding these lines to the `my.cnf` file:

```
[mysqld_safe]
malloc-lib=tcmalloc
```

Those lines also suffice for users on any platform who have installed a `tcmalloc` package in `/usr/lib`. To use a specific `tcmalloc` library, specify its full path name. Example:

```
[mysqld_safe]
malloc-lib=/opt/lib/libtcmalloc_minimal.so
```

- `--mysqld=prog_name` [257]

The name of the server program (in the `ledir` directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution. If `mysqld_safe` cannot find the server, use the `--ledir` [256] option to indicate the path name to the directory where the server is located.

- `--mysqld-version=suffix` [257]

This option is similar to the `--mysqld` [257] option, but you specify only the suffix for the server program name. The basename is assumed to be `mysqld`. For example, if you use `--mysqld-version=debug` [257], `mysqld_safe` starts the `mysqld-debug` program in the `ledir` directory. If the argument to `--mysqld-version` [257] is empty, `mysqld_safe` uses `mysqld` in the `ledir` directory.

- `--nice=priority` [257]

Use the `nice` program to set the server's scheduling priority to the given value.

- `--no-defaults` [257]

Do not read any option files. This must be the first option on the command line if it is used.

- `--open-files-limit=count` [257]

The number of files that `mysqld` should be able to open. The option value is passed to `ulimit -n`. Note that you need to start `mysqld_safe` as `root` for this to work properly!

- `--pid-file=file_name` [257]

The path name of the process ID file.

- `--port=port_num` [257]

The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` system user.

- `--skip-kill-mysqld` [257]

Do not try to kill stray `mysqld` processes at startup. This option works only on Linux.

- `--socket=path` [258]

The Unix socket file that the server should use when listening for local connections.

- `--syslog` [258], `--skip-syslog` [258]

`--syslog` [258] causes error messages to be sent to `syslog` on systems that support the `logger` program. `--skip-syslog` suppresses the use of `syslog`; messages are written to an error log file.

When `syslog` is used, the `daemon.err` syslog priority/facility is used for all log messages.

- `--syslog-tag=tag` [258]

For logging to `syslog`, messages from `mysqld_safe` and `mysqld` are written with a tag of `mysqld_safe` and `mysqld`, respectively. To specify a suffix for the tag, use `--syslog-tag=tag` [258], which modifies the tags to be `mysqld_safe-tag` and `mysqld-tag`.

- `--timezone=timezone` [258]

Set the `TZ` time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- `--user={user_name|user_id}` [258]

Run the `mysqld` server as the user having the name `user_name` or the numeric user ID `user_id`. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

If you execute `mysqld_safe` with the `--defaults-file` [256] or `--defaults-extra-file` [256] option to name an option file, the option must be the first one given on the command line or the option file will not be used. For example, this command will not use the named option file:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

Instead, use the following command:

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

The `mysqld_safe` script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See [Section 2.1.5, “Installation Layouts”](#).) `mysqld_safe` expects one of the following conditions to be true:

- The server and databases can be found relative to the working directory (the directory from which `mysqld_safe` is invoked). For binary distributions, `mysqld_safe` looks under its working directory for `bin` and `data` directories. For source distributions, it looks for `libexec` and `var` directories. This condition should be met if you execute `mysqld_safe` from your MySQL installation directory (for example, `/usr/local/mysql` for a binary distribution).
- If the server and databases cannot be found relative to the working directory, `mysqld_safe` attempts to locate them by absolute path names. Typical locations are `/usr/local/libexec` and `/usr/local/var`. The actual locations are determined from the values configured into the distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because `mysqld_safe` tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run `mysqld_safe` from the MySQL installation directory:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

If `mysqld_safe` fails, even when invoked from the MySQL installation directory, you can specify the `--ledir` [256] and `--datadir` [256] options to indicate the directories in which the server and databases are located on your system.

Beginning with MySQL 5.5.21, `mysqld_safe` tries to use the `sleep` and `date` system utilities to determine how many times it has attempted to start this second, and—if these are present and this is greater than 5 times—is forced to wait 1 full second before starting again. This is intended to prevent excessive CPU usage in the event of repeated failures. (Bug #11761530, Bug #54035)

When you use `mysqld_safe` to start `mysqld`, `mysqld_safe` arranges for error (and notice) messages from itself and from `mysqld` to go to the same destination.

There are several `mysqld_safe` options for controlling the destination of these messages:

- `--syslog` [258]: Write error messages to `syslog` on systems that support the `logger` program.
- `--skip-syslog` [258]: Do not write error messages to `syslog`. Messages are written to the default error log file (`host_name.err` in the data directory), or to a named file if the `--log-error` [256] option is given.
- `--log-error=file_name` [256]: Write error messages to the named error file.

If none of these options is given, the default is `--skip-syslog` [258].

If `--syslog` [258] and `--log-error` [256] are both given, a warning is issued and `--log-error` [256] takes precedence.

When `mysqld_safe` writes a message, notices go to the logging destination (`syslog` or the error log file) and `stdout`. Errors go to the logging destination and `stderr`.

Normally, you should not edit the `mysqld_safe` script. Instead, configure `mysqld_safe` by using command-line options or options in the `[mysqld_safe]` section of a `my.cnf` option file. In rare cases, it might be necessary to edit `mysqld_safe` to get it to start the server properly. However, if you do this, your modified version of `mysqld_safe` might be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.

4.3.3. mysql.server — MySQL Server Startup Script

MySQL distributions on Unix include a script named `mysql.server`. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the Mac OS X Startup Item for MySQL.

`mysql.server` can be found in the `support-files` directory under your MySQL installation directory or in a MySQL source distribution.

If you use the Linux server RPM package (`MySQL-server-VERSION.rpm`), the `mysql.server` script will be installed in the `/etc/init.d` directory with the name `mysql`. You need not install it manually. See Section 2.5.1, “Installing MySQL from RPM Packages on Linux”, for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as `mysqld`.

If you install MySQL from a source distribution or using a binary distribution format that does not install `mysql.server` automatically, you can install it manually. Instructions are provided in Section 2.11.1.2, “Starting and Stopping MySQL Automatically”.

`mysql.server` reads options from the `[mysql.server]` and `[mysqld]` sections of option files. For backward compatibility, it also reads `[mysql_server]` sections, although you should rename such sections to `[mysql.server]` when using MySQL 5.5.

`mysql.server` supports the following options.

- `--basedir=path` [260]

The path to the MySQL installation directory.

- `--datadir=path` [260]

The path to the MySQL data directory.

- `--pid-file=file_name` [260]

The path name of the file in which the server should write its process ID.

- `--service-startup-timeout=file_name` [260]

How long in seconds to wait for confirmation of server startup. If the server does not start within this time, `mysql.server` exits with an error. The default value is 900. A value of 0 means not to wait at all for startup. Negative values mean to wait forever (no timeout).

- `--use-mysqld_safe` [260]

Use `mysqld_safe` to start the server. This is the default.

- `--user=user_name` [260]

The login user name to use for running `mysqld`.

4.3.4. `mysqld_multi` — Manage Multiple MySQL Servers

`mysqld_multi` is designed to manage several `mysqld` processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status.

`mysqld_multi` searches for groups named `[mysqld N]` in `my.cnf` (or in the file named by the `--config-file` [261] option). N can be any positive integer. This number is referred to in the following discussion as the option group number, or *GNR*. Group numbers distinguish option groups from one another and are used as arguments to `mysqld_multi` to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the `[mysqld]` group used for starting `mysqld`. (See, for example, [Section 2.11.1.2, “Starting and Stopping MySQL Automatically”](#).) However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#).

To invoke `mysqld_multi`, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]
```

`start`, `stop`, and `report` indicate which operation to perform. You can perform the designated operation for a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, `mysqld_multi` performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named

[`mysqld17`] is 17. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value 10-13 represents groups [`mysqld10`] through [`mysqld13`]. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group [`mysqld17`]:

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups [`mysqld8`] and [`mysqld10`] through [`mysqld13`]:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

`mysqld_multi` searches for option files as follows:

- With `--no-defaults` [261], no option files are read.
- With `--defaults-file=file_name` [261], only the named file is read.
- Otherwise, option files in the standard list of locations are read, including any file named by the `--defaults-extra-file=file_name` [261] option, if one is given. (If the option is given multiple times, the last value is used.)

Option files read are searched for [`mysqld_multi`] and [`mysqldN`] option groups. The [`mysqld_multi`] group can be used for options to `mysqld_multi` itself. [`mysqldN`] groups can be used for options passed to specific `mysqld` instances.

The [`mysqld`] or [`mysqld_safe`] groups can be used for common options read by all instances of `mysqld` or `mysqld_safe`. You can specify a `--defaults-file=file_name` [248] option to use a different configuration file for that instance, in which case the [`mysqld`] or [`mysqld_safe`] groups from that file will be used for that instance.

`mysqld_multi` supports the following options.

- `--help` [261]
Display a help message and exit.
- `--config-file=file_name` [261]
This option is deprecated. If given, it is treated the same way as `--defaults-extra-file` [261], described earlier. `--config-file` [261] was removed in MySQL 5.5.3.
- `--example` [261]
Display a sample option file.
- `--log=file_name` [261]
Specify the name of the log file. If the file exists, log output is appended to it.
- `--mysqladmin=prog_name` [261]
The `mysqladmin` binary to be used to stop servers.

- `--mysqld=prog_name` [262]

The `mysqld` binary to be used. Note that you can specify `mysqld_safe` as the value for this option also. If you use `mysqld_safe` to start the server, you can include the `mysqld` or `ledir` options in the corresponding `[mysqldN]` option group. These options indicate the name of the server that `mysqld_safe` should start and the path name of the directory where the server is located. (See the descriptions for these options in [Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”](#).) Example:

```
[mysqld38]
mysqld = mysqld-debug
ledir  = /opt/local/mysql/libexec
```

- `--no-log` [262]

Print log information to `stdout` rather than to the log file. By default, output goes to the log file.

- `--password=password` [262]

The password of the MySQL account to use when invoking `mysqladmin`. Note that the password value is not optional for this option, unlike for other MySQL programs.

- `--silent` [262]

Silent mode; disable warnings.

- `--tcp-ip` [262]

Connect to each MySQL server through the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only through the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects `stop` and `report` operations.

- `--user=user_name` [262]

The user name of the MySQL account to use when invoking `mysqladmin`.

- `--verbose` [262]

Be more verbose.

- `--version` [262]

Display version information and exit.

Some notes about `mysql_d_multi`:

- **Most important:** Before using `mysql_d_multi` be sure that you understand the meanings of the options that are passed to the `mysqld` servers and *why* you would want to have separate `mysqld` processes. Beware of the dangers of using multiple `mysqld` servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory does *not* give you extra performance in a threaded system. See [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#).



Important

Make sure that the data directory for each server is fully accessible to the Unix account that the specific `mysqld` process is started as. *Do not* use the Unix `root` account for this, unless you *know* what you are doing. See [Section 6.1.5, “How to Run MySQL as a Normal User”](#).

- Make sure that the MySQL account used for stopping the `mysqld` servers (with the `mysqladmin` program) has the same user name and password for each server. Also, make sure that the account has the `SHUTDOWN` [697] privilege. If the servers that you want to manage have different user names or passwords for the administrative accounts, you might want to create an account on each server that has the same user name and password. For example, you might set up a common `multi_admin` account by executing the following commands for each server:

```
shell> mysql -u root -S /tmp/mysql.sock -p
Enter password:
mysql> GRANT SHUTDOWN ON *.*
-> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
```

See [Section 6.2, “The MySQL Access Privilege System”](#). You have to do this for each `mysqld` server. Change the connection parameters appropriately when connecting to each one. Note that the host name part of the account name must permit you to connect as `multi_admin` from the host where you want to run `mysqld_multi`.

- The Unix socket file and the TCP/IP port number must be different for every `mysqld`. (Alternatively, if the host has multiple network addresses, you can use `--bind-address` [435] to cause different servers to listen to different interfaces.)
- The `--pid-file` [257] option is very important if you are using `mysqld_safe` to start `mysqld` (for example, `--mysqld=mysqld_safe` [257]). Every `mysqld` should have its own process ID file. The advantage of using `mysqld_safe` instead of `mysqld` is that `mysqld_safe` monitors its `mysqld` process and restarts it if the process terminates due to a signal sent using `kill -9` or for other reasons, such as a segmentation fault. Please note that the `mysqld_safe` script might require that you start it from a certain place. This means that you might have to change location to a certain directory before running `mysqld_multi`. If you have problems starting, please see the `mysqld_safe` script. Check especially the lines:

```
-----
MY_PWD=`pwd`
# Check if we are starting this relative (for the binary release)
if test -d $MY_PWD/data/mysql -a \
  -f ./share/mysql/english/errmsg.sys -a \
  -x ./bin/mysqld
-----
```

The test performed by these lines should be successful, or you might encounter problems. See [Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”](#).

- You might want to use the `--user` [466] option for `mysqld`, but to do this you need to run the `mysqld_multi` script as the Unix superuser (`root`). Having the option in the option file doesn't matter; you just get a warning if you are not the superuser and the `mysqld` processes are started under your own Unix account.

The following example shows how you might set up an option file for use with `mysqld_multi`. The order in which the `mysqld` programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth `[mysqldN]` groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen

[mysqld_multi]
mysqld      = /usr/local/bin/mysqld_safe
mysqladmin  = /usr/local/bin/mysqladmin
```

```

user      = multi_admin
password  = multipass

[mysqld2]
socket    = /tmp/mysql.sock2
port      = 3307
pid-file  = /usr/local/mysql/var2/hostname.pid2
datadir   = /usr/local/mysql/var2
language  = /usr/local/share/mysql/english
user      = john

[mysqld3]
socket    = /tmp/mysql.sock3
port      = 3308
pid-file  = /usr/local/mysql/var3/hostname.pid3
datadir   = /usr/local/mysql/var3
language  = /usr/local/share/mysql/swedish
user      = monty

[mysqld4]
socket    = /tmp/mysql.sock4
port      = 3309
pid-file  = /usr/local/mysql/var4/hostname.pid4
datadir   = /usr/local/mysql/var4
language  = /usr/local/share/mysql/estonia
user      = tonu

[mysqld6]
socket    = /tmp/mysql.sock6
port      = 3311
pid-file  = /usr/local/mysql/var6/hostname.pid6
datadir   = /usr/local/mysql/var6
language  = /usr/local/share/mysql/japanese
user      = jani

```

See [Section 4.2.3.3, “Using Option Files”](#).

4.4. MySQL Installation-Related Programs

The programs in this section are used when installing or upgrading MySQL.

4.4.1. `comp_err` — Compile MySQL Error Message File

`comp_err` creates the `errmsg.sys` file that is used by `mysqld` to determine the error messages to display for different error codes. `comp_err` normally is run automatically when MySQL is built. It compiles the `errmsg.sys` file from the plaintext file located at `sql/share/errmsg.txt` in MySQL source distributions.

`comp_err` also generates `mysqld_error.h`, `mysqld_ename.h`, and `sql_state.h` header files.

For more information about how error messages are defined, see the [MySQL Internals Manual](#).

Invoke `comp_err` like this:

```
shell> comp_err [options]
```

`comp_err` supports the following options.

- `--help [264], -?`

Display a help message and exit.

- `--charset=path [265], -C path`

The character set directory. The default is `../sql/share/charsets`.

- `--debug=debug_options [265], -# debug_options`

Write a debugging log. A typical `debug_options` string is `'d:t:0,file_name'`. The default is `'d:t:0,/tmp/comp_err.trace'`.

- `--debug-info [265], -T`

Print some debugging information when the program exits.

- `--header_file=file_name [265], -H file_name`

The name of the error header file. The default is `mysqld_error.h`.

- `--in_file=file_name [265], -F file_name`

The name of the input file. The default is `../sql/share/errmsg.txt`.

- `--name_file=file_name [265], -N file_name`

The name of the error name file. The default is `mysqld_ername.h`.

- `--out_dir=path [265], -D path`

The name of the output base directory. The default is `../sql/share/`.

- `--out_file=file_name [265], -O file_name`

The name of the output file. The default is `errmsg.sys`.

- `--statefile=file_name [265], -S file_name`

The name for the SQLSTATE header file. The default is `sql_state.h`.

- `--version [265], -V`

Display version information and exit.

4.4.2. mysqlbug — Generate Bug Report

This program is obsolete.

The normal way to report bugs is to visit <http://bugs.mysql.com/>, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports.

4.4.3. mysql_install_db — Initialize MySQL Data Directory

`mysql_install_db` initializes the MySQL data directory and creates the system tables that it contains, if they do not exist. `mysql_install_db` is a shell script and is available only on Unix platforms. (As of MySQL 5.6, `mysql_install_db` is a Perl script and can be used on any system with Perl installed.)

To invoke `mysql_install_db`, use the following syntax:

```
shell> mysql_install_db [options]
```

Because the MySQL server, `mysqld`, needs to access the data directory when it runs later, you should either run `mysql_install_db` from the same system account that will be used for running `mysqld` or run it as `root` and use the `--user` [267] option to indicate the user name that `mysqld` will run as. It might be necessary to specify other options such as `--basedir` [266] or `--datadir` [266] if `mysql_install_db` does not use the correct locations for the installation directory or data directory. For example:

```
shell> scripts/mysql_install_db --user=mysql \
      --basedir=/opt/mysql/mysql \
      --datadir=/opt/mysql/mysql/data
```

`mysql_install_db` needs to invoke `mysqld` with the `--bootstrap` [436] and `--skip-grant-tables` [461] options. If MySQL was configured with the `DISABLE_GRANT_OPTIONS` [156] compiler flag, `--bootstrap` [436] and `--skip-grant-tables` [461] will be disabled (see Section 2.10.4, “MySQL Source-Configuration Options”). To handle this, set the `MYSQLD_BOOTSTRAP` environment variable to the full path name of a server that has all options enabled. `mysql_install_db` will use that server.



Note

If you have set a custom `TMPDIR` environment variable when performing the installation, and the specified directory is not accessible, `mysql_install_db` may fail. If so, unset `TMPDIR` or set `TMPDIR` to point to the system temporary directory (usually `/tmp`).

`mysql_install_db` supports the following options, which can be specified on the command line or in the `[mysql_install_db]` group of an option file. (Options that are common to `mysqld` can also be specified in the `[mysqld]` group.) Other options are passed to `mysqld`. For information about option files, see Section 4.2.3.3, “Using Option Files”. `mysql_install_db` also supports the options for processing option files described at Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”.

- `--basedir=path` [266]

The path to the MySQL installation directory.

- `--datadir=path` [266], `--ldata=path` [266]

The path to the MySQL data directory.

- `--force` [266]

Cause `mysql_install_db` to run even if DNS does not work. Grant table entries that normally use host names will use IP addresses.

- `--rpm` [266]

For internal use. This option is used during the MySQL installation process for install operations performed using RPM packages.

- `--skip-name-resolve` [266]

Use IP addresses rather than host names when creating grant table entries. This option can be useful if your DNS does not work.

- `--srcdir=path` [266]

For internal use. This option specifies the directory under which `mysql_install_db` looks for support files such as the error message file and the file for populating the help tables.

- `--user=user_name` [267]

The system (login) user name to use for running `mysqld`. Files and directories created by `mysqld` will be owned by this user. You must be `root` to use this option. By default, `mysqld` runs using your current login name and files and directories that it creates will be owned by you.

- `--verbose` [267]

Verbose mode. Print more information about what the program does.

- `--windows` [267]

For internal use. This option is used for creating Windows distributions.

4.4.4. `mysql_plugin` — Configure MySQL Server Plugins

The `mysql_plugin` utility enables MySQL administrators to manage which plugins a MySQL server loads. It provides an alternative to manually specifying the `--plugin-load` [457] option at server startup or using the `INSTALL PLUGIN` and `UNINSTALL PLUGIN` statements at runtime. `mysql_plugin` is available as of MySQL 5.5.16.

Depending on whether `mysql_plugin` is invoked to enable or disable plugins, it inserts or deletes rows in the `mysql.plugin` table that serves as a plugin registry. (To perform this operation, `mysql_plugin` invokes the MySQL server in bootstrap mode. This means that the server must not already be running.) For normal server startups, the server loads and enables plugins listed in `mysql.plugin` automatically. For additional control over plugin activation, use `--plugin_name` options named for specific plugins, as described in [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

Each invocation of `mysql_plugin` reads a configuration file to determine how to configure the plugins contained in a single plugin library object file. To invoke `mysql_plugin`, use this syntax:

```
mysql_plugin [options] plugin {ENABLE|DISABLE}
```

`plugin` is the name of the plugin to configure. `ENABLE` or `DISABLE` (not case sensitive) specify whether to enable or disable components of the plugin library named in the configuration file. The order of the `plugin` and `ENABLE` or `DISABLE` arguments does not matter.

For example, to configure components of a plugin library file named `myplugins.so` on Linux or `myplugins.dll` on Windows, specify a `plugin` value of `myplugins`. Suppose that this plugin library contains three plugins, `plugin1`, `plugin2`, and `plugin3`, all of which should be configured under `mysql_plugin` control. By convention, configuration files have a suffix of `.ini` and the same basename as the plugin library, so the default configuration file name for this plugin library is `myplugins.ini`. The configuration file contents look like this:

```
myplugins
plugin1
plugin2
plugin3
```

The first line in the `myplugins.ini` file is the name of the library object file, without any extension such as `.so` or `.dll`. The remaining lines are the names of the components to be enabled or disabled. Each value in the file should be on a separate line. Lines on which the first character is `#` are taken as comments and ignored.

To enable the plugins listed in the configuration file, invoke `mysql_plugin` this way:

```
shell> mysql_plugin myplugins ENABLE
```

To disable the plugins, use `DISABLE` rather than `ENABLE`.

An error occurs if `mysql_plugin` cannot find the configuration file or plugin library file, or if `mysql_plugin` cannot start the MySQL server.

`mysql_plugin` supports the following options, which can be specified on the command line or in the `[mysqld]` group of any option file. For options specified in a `[mysqld]` group, `mysql_plugin` recognizes the `--basedir` [268], `--datadir` [268], and `--plugin-dir` [268] options and ignores others. For information about option files, see [Section 4.2.3.3, “Using Option Files”](#).

mysql_plugin Options

- `--help` [268], `-?`

Display a help message and exit.

- `--basedir=path` [268], `-b path`

The server base directory.

- `--datadir=path` [268], `-d path`

The server data directory.

- `--my-print-defaults=path` [268], `-b path`

The path to the `my_print_defaults` program.

- `--mysqld=path` [268], `-b path`

The path to the `mysqld` server.

- `--no-defaults` [268], `-p`

Do not read values from the configuration file. This option enables an administrator to skip reading defaults from the configuration file.

With `mysql_plugin`, this option need not be given first on the command line, unlike most other MySQL programs that support `--no-defaults` [268].

- `--plugin-dir=path` [268], `-p path`

The server plugin directory.

- `--plugin-ini=file_name` [268], `-i file_name`

The `mysql_plugin` configuration file. Relative path names are interpreted relative to the current directory. If this option is not given, the default is `plugin.ini` in the plugin directory, where `plugin` is the `plugin` argument on the command line.

- `--print-defaults` [268], `-P`

Display the default values from the configuration file. This option causes `mysql_plugin` to print the defaults for `--basedir` [268], `--datadir` [268], and `--plugin-dir` [268] if they are found in the configuration file. If no value for a variable is found, nothing is shown.

With `mysql_plugin`, this option need not be given first on the command line, unlike most other MySQL programs that support `--print-defaults` [268].

- `--verbose` [269], `-v`

Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version` [269], `-V`

Display version information and exit.

4.4.5. `mysql_secure_installation` — Improve MySQL Installation Security

This program enables you to improve the security of your MySQL installation in the following ways:

- You can set a password for `root` accounts.
- You can remove `root` accounts that are accessible from outside the local host.
- You can remove anonymous-user accounts.
- You can remove the `test` database (which by default can be accessed by all users, even anonymous users), and privileges that permit anyone to access databases with names that start with `test_`.

`mysql_secure_installation` helps you implement security recommendations similar to those described at [Section 2.11.2, “Securing the Initial MySQL Accounts”](#).

Invoke `mysql_secure_installation` without arguments:

```
shell> mysql_secure_installation
```

The script will prompt you to determine which actions to perform.

`mysql_secure_installation` is not available on Windows.

4.4.6. `mysql_tzinfo_to_sql` — Load the Time Zone Tables

The `mysql_tzinfo_to_sql` program loads the time zone tables in the `mysql` database. It is used on systems that have a `zoneinfo` database (the set of files describing time zones). Examples of such systems are Linux, FreeBSD, Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory (`/usr/share/lib/zoneinfo` on Solaris). If your system does not have a `zoneinfo` database, you can use the downloadable package described in [Section 10.6, “MySQL Server Time Zone Support”](#).

`mysql_tzinfo_to_sql` can be invoked several ways:

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

For the first invocation syntax, pass the `zoneinfo` directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

The second syntax causes `mysql_tzinfo_to_sql` to load a single time zone file `tz_file` that corresponds to a time zone name `tz_name`:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

If your time zone needs to account for leap seconds, invoke `mysql_tzinfo_to_sql` using the third syntax, which initializes the leap second information. `tz_file` is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

After running `mysql_tzinfo_to_sql`, it is best to restart the server so that it does not continue to use any previously cached time zone data.

4.4.7. mysql_upgrade — Check and Upgrade MySQL Tables

`mysql_upgrade` examines all tables in all databases for incompatibilities with the current version of MySQL Server. `mysql_upgrade` also upgrades the system tables so that you can take advantage of new privileges or capabilities that might have been added.

`mysql_upgrade` should be executed each time you upgrade MySQL.

If `mysql_upgrade` finds that a table has a possible incompatibility, it performs a table check and, if problems are found, attempts a table repair. If the table cannot be repaired, see [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#) for manual table repair strategies.



Note

On Windows Server 2008, Vista, and newer, you must run `mysql_upgrade` with administrator privileges. You can do this by running a Command Prompt as Administrator and running the command. Failure to do so may result in the upgrade failing to execute correctly.



Caution

You should always back up your current MySQL installation *before* performing an upgrade. See [Section 7.2, “Database Backup Methods”](#).

Some upgrade incompatibilities may require special handling before you upgrade your MySQL installation and run `mysql_upgrade`. See [Section 2.12.1, “Upgrading MySQL”](#), for instructions on determining whether any such incompatibilities apply to your installation and how to handle them.

To use `mysql_upgrade`, make sure that the server is running, and then invoke it like this:

```
shell> mysql_upgrade [options]
```

After running `mysql_upgrade`, stop the server and restart it so that any changes made to the system tables take effect.

`mysql_upgrade` executes the following commands to check and repair tables and to upgrade the system tables:

```
mysqlcheck --all-databases --check-upgrade --auto-repair
mysql < fix_priv_tables
mysqlcheck --all-databases --check-upgrade --fix-db-names --fix-table-names
```

Notes about the preceding commands:

- Because `mysql_upgrade` invokes `mysqlcheck` with the `--all-databases` [303] option, it processes all tables in all databases, which might take a long time to complete. Each table is locked and

therefore unavailable to other sessions while it is being processed. Check and repair operations can be time-consuming, particularly for large tables.

- For details about what checks the `--check-upgrade` [304] option entails, see the description of the `FOR UPGRADE` option of the `CHECK TABLE` statement (see [Section 13.7.2.2, “CHECK TABLE Syntax”](#)).
- `fix_priv_tables` represents a script generated internally by `mysql_upgrade` that contains SQL statements to upgrade the tables in the `mysql` database.

All checked and repaired tables are marked with the current MySQL version number. This ensures that next time you run `mysql_upgrade` with the same version of the server, it can tell whether there is any need to check or repair the table again.

`mysql_upgrade` also saves the MySQL version number in a file named `mysql_upgrade_info` in the data directory. This is used to quickly check whether all tables have been checked for this release so that table-checking can be skipped. To ignore this file and perform the check regardless, use the `--force` [272] option.

If you install MySQL from RPM packages on Linux, you must install the server and client RPMs. `mysql_upgrade` is included in the server RPM but requires the client RPM because the latter includes `mysqlcheck`. (See [Section 2.5.1, “Installing MySQL from RPM Packages on Linux”](#).)

`mysql_upgrade` does not upgrade the contents of the help tables. For upgrade instructions, see [Section 5.1.10, “Server-Side Help”](#).

`mysql_upgrade` supports the following options, which can be specified on the command line or in the `[mysql_upgrade]` and `[client]` groups of an option file. Other options are passed to `mysqlcheck`. For example, it might be necessary to specify the `--password[=password]` [305] option. `mysql_upgrade` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

- `--help` [271]

Display a short help message and exit.

- `--basedir=path` [271]

The path to the MySQL installation directory. This option is accepted for backward compatibility but ignored.

- `--datadir=path` [271]

The path to the data directory. This option is accepted for backward compatibility but ignored.

- `--debug-check` [271]

Print some debugging information when the program exits.

- `--debug-info` [271], `-T`

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin` [271]

The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.10.

- `--force` [272]

Ignore the `mysql_upgrade_info` file and force execution of `mysqlcheck` even if `mysql_upgrade` has already been executed for the current version of MySQL.

- `--plugin-dir=path` [272]

The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [271] option is used to specify an authentication plugin but `mysql_upgrade` does not find it. See Section 6.3.6, “Pluggable Authentication”.

This option was added in MySQL 5.5.10.

- `--tmpdir=path` [272], `-t path`

The path name of the directory to use for creating temporary files.

- `--upgrade-system-tables` [272], `-s`

Upgrade only the system tables, do not upgrade data.

- `--user=user_name` [272], `-u user_name`

The MySQL user name to use when connecting to the server. The default user name is `root`.

- `--verbose` [272]

Verbose mode. Print more information about what the program does.

- `--write-binlog` [272]

Cause binary logging to be enabled while `mysql_upgrade` runs. This is the default behavior; to disable binary logging during the upgrade, use the inverse of this option (that is, start the program with `--skip-write-binlog` [272]).

4.5. MySQL Client Programs

This section describes client programs that connect to the MySQL server.

4.5.1. `mysql` — The MySQL Command-Line Tool

`mysql` is a simple SQL shell with input line editing capabilities. It supports interactive and noninteractive use. When used interactively, query results are presented in an ASCII-table format. When used noninteractively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the `--quick` [280] option. This forces `mysql` to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the `mysql_use_result()` C API function in the client/server library rather than `mysql_store_result()`.

Using `mysql` is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password=your_password db_name
```

Then type an SQL statement, end it with “;”, \g, or \G and press Enter.

Typing Control+C causes `mysql` to attempt to kill the current statement. If this cannot be done, or Control +C is typed again before the statement is killed, `mysql` exits. Previously, Control+C caused `mysql` to exit in all cases.

You can execute SQL statements in a script file (batch file) like this:

```
shell> mysql db_name < script.sql > output.tab
```

On Unix, the `mysql` client writes a record of executed statements to a history file. See [Section 4.5.1.3, “mysql History File”](#).

4.5.1.1. mysql Options

`mysql` supports the following options, which can be specified on the command line or in the `[mysql]` and `[client]` groups of an option file. `mysql` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.2. mysql Options

Format	Option File	Description	Introduced	Removed
<code>--auto-rehash [276]</code>	<code>auto-rehash [276]</code>	Enable automatic rehashing		
<code>--auto-vertical-output [276]</code>	<code>auto-vertical-output [276]</code>	Enable automatic vertical result set display	5.5.3	
<code>--batch [276]</code>	<code>batch [276]</code>	Don't use history file		
<code>--character-sets-dir=path [277]</code>	<code>character-sets-dir [277]</code>	Set the default character set		
<code>--column-names [277]</code>	<code>column-names [277]</code>	Write column names in results		
<code>--column-type-info [277]</code>	<code>column-type-info [277]</code>	Display result set metadata		
<code>--comments [277]</code>	<code>comments [277]</code>	Whether to retain or strip comments in statements sent to the server		
<code>--compress [277]</code>	<code>compress [277]</code>	Compress all information sent between the client and the server		
<code>--connect_timeout=value [283]</code>	<code>connect_timeout [283]</code>	The number of seconds before connection timeout		
<code>--database=dbname [277]</code>	<code>database [277]</code>	The database to use		
<code>--debug[=debug_options] [277]</code>	<code>debug [277]</code>	Write a debugging log		
<code>--debug-check [277]</code>	<code>debug-check [277]</code>	Print debugging information when the program exits		
<code>--debug-info [277]</code>	<code>debug-info [277]</code>	Print debugging information, memory and CPU statistics when the program exits		
<code>--default-auth=plugin [277]</code>	<code>default-auth=plugin [277]</code>	The authentication plugin to use	5.5.7	

Format	Option File	Description	Introduced	Removed
<code>--default-character-set=charset_name</code> [277]	<code>default-character-set</code> [277]	Use <code>charset_name</code> as the default character set		
<code>--delimiter=str</code> [278]	<code>delimiter</code> [278]	Set the statement delimiter		
<code>--enable-cleartext-plugin</code> [278]	<code>enable-cleartext-plugin</code> [278]	Enable <code>cleartext</code> authentication plugin	5.5.27	
<code>--execute=statement</code> [278]	<code>execute</code> [278]	Execute the statement and quit		
<code>--force</code> [278]	<code>force</code> [278]	Continue even if an SQL error occurs		
<code>--help</code> [276]		Display help message and exit		
<code>--host=host_name</code> [278]	<code>host</code> [278]	Connect to the MySQL server on the given host		
<code>--html</code> [278]	<code>html</code> [278]	Produce HTML output		
<code>--ignore-spaces</code> [278]	<code>ignore-spaces</code> [278]	Ignore spaces after function names		
<code>--init-command=str</code> [278]	<code>init-command</code> [278]	SQL statement to execute after connecting		
<code>--line-numbers</code> [278]	<code>line-numbers</code> [278]	Write line numbers for errors		
<code>--local-infile[={0 1}]</code> [278]	<code>local-infile</code> [278]	Enable or disable for LOCAL capability for LOAD DATA INFILE		
<code>--max_allowed_packet=value</code> [283]	<code>max_allowed_packet</code> [283]	The maximum packet length to send to or receive from the server		
<code>--max_join_size=value</code> [283]	<code>max_join_size</code> [283]	The automatic limit for rows in a join when using <code>--safe-updates</code>		
<code>--named-commands</code> [279]	<code>named-commands</code> [279]	Enable named mysql commands		
<code>--net_buffer_length=value</code>	<code>net_buffer_length</code>	The buffer size for TCP/IP and socket communication		
<code>--no-auto-rehash</code> [279]		Disable automatic rehashing		
<code>--no-beep</code> [279]	<code>no-beep</code> [279]	Do not beep when errors occur		
<code>--no-named-commands</code> [279]	<code>no-named-commands</code> [279]	Disable named mysql commands		5.5.3
<code>--no-pager</code> [279]	<code>no-pager</code> [279]	Deprecated form of <code>--skip-pager</code>		5.5.3
<code>--no-tee</code> [279]	<code>no-tee</code> [279]	Do not copy output to a file		5.5.3
<code>--one-database</code> [279]	<code>one-database</code> [279]	Ignore statements except those for the default database named on the command line		
<code>--pager[=command]</code> [280]	<code>pager</code> [280]	Use the given command for paging query output		

Format	Option File	Description	Introduced	Removed
--password[=password]	password [280]	The password to use when connecting to the server		
--plugin-dir=path	plugin-dir=path [280]	The directory where plugins are located	5.5.7	
--port=port_num	port [280]	The TCP/IP port number to use for the connection		
--prompt=format_str	prompt [280]	Set the prompt to the specified format		
--protocol=type	protocol [280]	The connection protocol to use		
--quick	quick [280]	Do not cache each query result		
--raw	raw [280]	Write column values without escape conversion		
--reconnect	reconnect [281]	If the connection to the server is lost, automatically try to reconnect		
--safe-updates	safe-updates [281]	Allow only UPDATE and DELETE statements that specify key values		
--secure-auth	secure-auth [281]	Do not send passwords to the server in old (pre-4.1.1) format		
--select-limit=value	select_limit [283]	The automatic limit for SELECT statements when using --safe-updates		
--show-warnings	show-warnings [281]	Show warnings after each statement if there are any		
--sigint-ignore	sigint-ignore [281]	Ignore SIGINT signals (typically the result of typing Control+C)		
--silent	silent [281]	Silent mode		
--skip-auto-rehash	skip-auto-rehash [276]	Disable automatic rehashing		
--skip-column-names	skip-column-names [282]	Do not write column names in results		
--skip-line-numbers	skip-line-numbers [282]	Skip line numbers for errors		
--skip-named-commands	skip-named-commands [279]	Disable named mysql commands		
--skip-pager	skip-pager [280]	Disable paging		
--skip-reconnect	skip-reconnect [281]	Disable reconnecting		
--socket=path	socket [282]	For connections to localhost		
--ssl-ca=file_name	ssl-ca [282]	The path to a file that contains a list of trusted SSL CAs		
--ssl-capath=dir_name	ssl-capath [282]	The path to a directory that contains trusted SSL CA certificates in PEM format		

Format	Option File	Description	Introduced	Removed
<code>--ssl-cert=file_name</code> [282]	<code>ssl-cert</code> [282]	The name of the SSL certificate file to use for establishing a secure connection		
<code>--ssl-cipher=cipher_list</code> [282]	<code>ssl-cipher</code> [282]	A list of allowable ciphers to use for SSL encryption		
<code>--ssl-key=file_name</code> [282]	<code>ssl-key</code> [282]	The name of the SSL key file to use for establishing a secure connection		
<code>--ssl-verify-server-cert</code> [282]	<code>ssl-verify-server-cert</code> [282]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server		
<code>--table</code> [282]	<code>table</code> [282]	Display output in tabular format		
<code>--tee=file_name</code> [282]	<code>tee</code> [282]	Append a copy of output to the given file		
<code>--unbuffered</code> [282]	<code>unbuffered</code> [282]	Flush the buffer after each query		
<code>--user=user_name</code> [282]	<code>user</code> [282]	MySQL user name to use when connecting to server		
<code>--verbose</code> [282]		Verbose mode		
<code>--version</code> [282]		Display version information and exit		
<code>--vertical</code> [282]	<code>vertical</code> [282]	Print query output rows vertically (one line per column value)		
<code>--wait</code> [282]	<code>wait</code> [282]	If the connection cannot be established, wait and retry instead of aborting		
<code>--xml</code> [282]	<code>xml</code> [282]	Produce XML output		

- `--help` [276], `-?`

Display a help message and exit.

- `--auto-rehash` [276]

Enable automatic rehashing. This option is on by default, which enables database, table, and column name completion. Use `--disable-auto-rehash` [276] to disable rehashing. That causes `mysql` to start faster, but you must issue the `rehash` command if you want to use name completion.

To complete a name, enter the first part and press Tab. If the name is unambiguous, `mysql` completes it. Otherwise, you can press Tab again to see the possible names that begin with what you have typed so far. Completion does not occur if there is no default database.

- `--auto-vertical-output` [276]

Cause result sets to be displayed vertically if they are too wide for the current window, and using normal tabular format otherwise. (This applies to statements terminated by `;` or `\G`.) This option was added in MySQL 5.5.3.

- `--batch` [276], `-B`

Print results using tab as the column separator, with each row on a new line. With this option, `mysql` does not use the history file.

Batch mode results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` [280] option.

- `--bind-address=ip_address` [277]

On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

This option is supported only in the version of the `mysql` client that is supplied with MySQL Cluster. It is not available in standard MySQL Server 5.5 releases.

- `--character-sets-dir=path` [277]

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--column-names` [277]

Write column names in results.

- `--column-type-info` [277], `-m`

Display result set metadata.

- `--comments` [277], `-c`

Whether to preserve comments in statements sent to the server. The default is `--skip-comments` (discard comments), enable with `--comments` (preserve comments).

- `--compress` [277], `-C`

Compress all information sent between the client and the server if both support compression.

- `--database=db_name` [277], `-D db_name`

The database to use. This is useful primarily in an option file.

- `--debug[=debug_options]` [277], `-# [debug_options]`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysql.trace'`.

- `--debug-check` [277]

Print some debugging information when the program exits.

- `--debug-info` [277], `-T`

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin` [277]

The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.7.

- `--default-character-set=charset_name` [277]

Use `charset_name` as the default character set for the client and connection.

A common issue that can occur when the operating system uses `utf8` or another multi-byte character set is that output from the `mysql` client is formatted incorrectly, due to the fact that the MySQL client uses the `latin1` character set by default. You can usually fix such issues by using this option to force the client to use the system character set instead.

See [Section 10.5, “Character Set Configuration”](#), for more information.

- `--delimiter=str` [278]

Set the statement delimiter. The default is the semicolon character (“;”).

- `--disable-named-commands` [278]

Disable named commands. Use the `*` form only, or use named commands only at the beginning of a line ending with a semicolon (“;”). `mysql` starts with this option *enabled* by default. However, even with this option, long-format commands still work from the first line. See [Section 4.5.1.2, “mysql Commands”](#).

- `--enable-cleartext-plugin` [278]

Enable the `mysql_clear_password` cleartext authentication plugin. (See [Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”](#).) This option was added in MySQL 5.5.27.

- `--execute=statement` [278], `-e statement`

Execute the statement and quit. The default output format is like that produced with `--batch` [276]. See [Section 4.2.3.1, “Using Options on the Command Line”](#), for some examples. With this option, `mysql` does not use the history file.

- `--force` [278], `-f`

Continue even if an SQL error occurs.

- `--host=host_name` [278], `-h host_name`

Connect to the MySQL server on the given host.

- `--html` [278], `-H`

Produce HTML output.

- `--ignore-spaces` [278], `-i`

Ignore spaces after function names. The effect of this is described in the discussion for the `IGNORE_SPACE` [613] SQL mode (see [Section 5.1.7, “Server SQL Modes”](#)).

- `--init-command=str` [278]

SQL statement to execute after connecting to the server. If auto-reconnect is enabled, the statement is executed again after reconnection occurs.

- `--line-numbers` [278]

Write line numbers for errors. Disable this with `--skip-line-numbers` [282].

- `--local-infile[={0|1}]` [278]

Enable or disable `LOCAL` capability for `LOAD DATA INFILE`. With no value, the option enables `LOCAL`. The option may be given as `--local-infile=0` [278] or `--local-infile=1` [278] to explicitly disable or enable `LOCAL`. Enabling `LOCAL` has no effect if the server does not also support it.

- `--named-commands` [279], `-G`

Enable named `mysql` commands. Long-format commands are permitted, not just short-format commands. For example, `quit` and `\q` both are recognized. Use `--skip-named-commands` [279] to disable named commands. See Section 4.5.1.2, “`mysql` Commands”.

- `--no-auto-rehash` [276], `-A`

This has the same effect as `--skip-auto-rehash`. See the description for `--auto-rehash` [276].

- `--no-beep` [279], `-b`

Do not beep when errors occur.

- `--no-named-commands` [279], `-g`

Deprecated, use `--disable-named-commands` [278] instead. `--no-named-commands` [279] was removed in MySQL 5.5.3.

- `--no-pager` [279]

Deprecated form of `--skip-pager` [280]. See the `--pager` [280] option. `--no-pager` [279] was removed in MySQL 5.5.3.

- `--no-tee` [279]

Deprecated form of `--skip-tee` [282]. See the `--tee` [282] option. `--no-tee` [279] is removed in MySQL 5.5.3.

- `--one-database` [279], `-o`

Ignore statements except those that occur while the default database is the one named on the command line. This option is rudimentary and should be used with care. Statement filtering is based only on `USE` statements.

Initially, `mysql` executes statements in the input because specifying a database `db_name` on the command line is equivalent to inserting `USE db_name` at the beginning of the input. Then, for each `USE` statement encountered, `mysql` accepts or rejects following statements depending on whether the database named is the one on the command line. The content of the statements is immaterial.

Suppose that `mysql` is invoked to process this set of statements:

```
DELETE FROM db2.t2;
USE db2;
DROP TABLE db1.t1;
CREATE TABLE db1.t1 (i INT);
USE db1;
INSERT INTO t1 (i) VALUES(1);
CREATE TABLE db2.t1 (j INT);
```

If the command line is `mysql --force --one-database db1`, `mysql` handles the input as follows:

- The `DELETE` statement is executed because the default database is `db1`, even though the statement names a table in a different database.

- The `DROP TABLE` and `CREATE TABLE` statements are not executed because the default database is not `db1`, even though the statements name a table in `db1`.
- The `INSERT` and `CREATE TABLE` statements are executed because the default database is `db1`, even though the `CREATE TABLE` statement names a table in a different database.
- `--pager[=command]` [280]

Use the given command for paging query output. If the command is omitted, the default pager is the value of your `PAGER` environment variable. Valid pagers are `less`, `more`, `cat [> filename]`, and so forth. This option works only on Unix and only in interactive mode. To disable paging, use `--skip-pager` [280]. Section 4.5.1.2, “mysql Commands”, discusses output paging further.

- `--password[=password]` [280], `-p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` [280] or `-p` option on the command line, `mysql` prompts for one.

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe` [280], `-W`

On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path` [280]

The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [277] option is used to specify an authentication plugin but `mysql` does not find it. See Section 6.3.6, “Pluggable Authentication”.

This option was added in MySQL 5.5.7.

- `--port=port_num` [280], `-P port_num`

The TCP/IP port number to use for the connection.

- `--prompt=format_str` [280]

Set the prompt to the specified format. The default is `mysql>`. The special sequences that the prompt can contain are described in Section 4.5.1.2, “mysql Commands”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [280]

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--quick` [280], `-q`

Do not cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, `mysql` does not use the history file.

- `--raw` [280], `-r`

For tabular output, the “boxing” around columns enables one column value to be distinguished from another. For nontabular output (such as is produced in batch mode or when the `--batch` [276] or `--silent` [281] option is given), special characters are escaped in the output so they can be identified easily. Newline, tab, `NUL`, and backslash are written as `\n`, `\t`, `\0`, and `\\`. The `--raw` [280] option disables this character escaping.

The following example demonstrates tabular versus nontabular output and the use of raw mode to disable escaping:

```
% mysql
mysql> SELECT CHAR(92);
+-----+
| CHAR(92) |
+-----+
| \        |
+-----+

% mysql -s
mysql> SELECT CHAR(92);
CHAR(92)
\\

% mysql -s -r
mysql> SELECT CHAR(92);
CHAR(92)
\
```

- `--reconnect` [281]

If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use `--skip-reconnect` [281].

- `--safe-updates` [281], `--i-am-a-dummy` [281], `-U`

Permit only those `UPDATE` and `DELETE` statements that specify which rows to modify by using key values. If you have set this option in an option file, you can override it by using `--safe-updates` [281] on the command line. See [Section 4.5.1.6, “mysql Tips”](#), for more information about this option.

- `--secure-auth` [281]

Do not send passwords to the server in old (pre-4.1.1) format. This prevents connections except for servers that use the newer password format.

- `--show-warnings` [281]

Cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode.

- `--sigint-ignore` [281]

Ignore `SIGINT` signals (typically the result of typing Control+C).

- `--silent` [281], `-s`

Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

This option results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` [280] option.

- `--skip-column-names` [282], `-N`

Do not write column names in results.

- `--skip-line-numbers` [282], `-L`

Do not write line numbers for errors. Useful when you want to compare result files that include error messages.

- `--socket=path` [282], `-S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` [749] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See [Section 6.3.8.4, “SSL Command Options”](#).

- `--table` [282], `-t`

Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- `--tee=file_name` [282]

Append a copy of output to the given file. This option works only in interactive mode. [Section 4.5.1.2, “mysql Commands”](#), discusses tee files further.

- `--unbuffered` [282], `-n`

Flush the buffer after each query.

- `--user=user_name` [282], `-u user_name`

The MySQL user name to use when connecting to the server.

- `--verbose` [282], `-v`

Verbose mode. Produce more output about what the program does. This option can be given multiple times to produce more and more output. (For example, `-v -v -v` produces table output format even in batch mode.)

- `--version` [282], `-V`

Display version information and exit.

- `--vertical` [282], `-E`

Print query output rows vertically (one line per column value). Without this option, you can specify vertical output for individual statements by terminating them with `\G`.

- `--wait` [282], `-w`

If the connection cannot be established, wait and retry instead of aborting.

- `--xml` [282], `-X`

Produce XML output.

```
<field name="column_name">NULL</field>
```

The output when `--xml` [282] is used with `mysql` matches that of `mysqldump --xml` [322]. See Section 4.5.4, “`mysqldump` — A Database Backup Program” for details.

The XML output also uses an XML namespace, as shown here:

```
shell> mysql --xml -uroot -e "SHOW VARIABLES LIKE 'version%'"
<?xml version="1.0"?>

<resultset statement="SHOW VARIABLES LIKE 'version%'" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<row>
<field name="Variable_name">version</field>
<field name="Value">5.0.40-debug</field>
</row>

<row>
<field name="Variable_name">version_comment</field>
<field name="Value">Source distribution</field>
</row>

<row>
<field name="Variable_name">version_compile_machine</field>
<field name="Value">i686</field>
</row>

<row>
<field name="Variable_name">version_compile_os</field>
<field name="Value">suse-linux-gnu</field>
</row>
</resultset>
```

(See Bug #25946.)

You can also set the following variables by using `--var_name=value`. The `--set-variable` format is deprecated and was removed in MySQL 5.5.3.

- `connect_timeout`

The number of seconds before connection timeout. (Default value is 0.)

- `max_allowed_packet`

The maximum packet length to send to or receive from the server. (Default value is 16MB.)

- `max_join_size` [522]

The automatic limit for rows in a join when using `--safe-updates` [281]. (Default value is 1,000,000.)

- `net_buffer_length` [531]

The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `select_limit`

The automatic limit for `SELECT` statements when using `--safe-updates` [281]. (Default value is 1,000.)

4.5.1.2. `mysql` Commands

`mysql` sends each SQL statement that you issue to the server to be executed. There is also a set of commands that `mysql` itself interprets. For a list of these commands, type `help` or `\h` at the `mysql>` prompt:

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?          (\?)  Synonym for `help`.
clear      (\c)  Clear command.
connect    (\r)  Reconnect to the server. Optional arguments are db and host.
delimiter (\d)  Set statement delimiter.
edit       (\e)  Edit command with $EDITOR.
ego        (\G)  Send command to mysql server, display result vertically.
exit       (\q)  Exit mysql. Same as quit.
go         (\g)  Send command to mysql server.
help       (\h)  Display this help.
nopager    (\n)  Disable pager, print to stdout.
notee      (\t)  Don't write into outfile.
pager      (\P)  Set PAGER [to_pager]. Print the query results via PAGER.
print      (\p)  Print current command.
prompt     (\R)  Change your mysql prompt.
quit       (\q)  Quit mysql.
rehash     (\#)  Rebuild completion hash.
source     (\.)  Execute an SQL script file. Takes a file name as an argument.
status     (\s)  Get status information from the server.
system     (\!)  Execute a system shell command.
tee        (\T)  Set outfile [to_outfile]. Append everything into given
              outfile.
use        (\u)  Use another database. Takes database name as argument.
charset    (\C)  Switch to another charset. Might be needed for processing
              binlog with multi-byte charsets.
warnings   (\W)  Show warnings after every statement.
nowarning  (\w)  Don't show warnings after every statement.

For server side help, type 'help contents'
```

Each command has both a long and short form. The long form is not case sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

The use of short-form commands within multi-line `/* ... */` comments is not supported.

- `help [arg], \h [arg], \? [arg], ? [arg]`

Display a help message listing the available `mysql` commands.

If you provide an argument to the `help` command, `mysql` uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. For more information, see [Section 4.5.1.4, “mysql Server-Side Help”](#).

- `charset charset_name, \C charset_name`

Change the default character set and issue a `SET NAMES` statement. This enables the character set to remain synchronized on the client and server if `mysql` is run with auto-reconnect enabled (which is not recommended), because the specified character set is used for reconnects.

- `clear, \c`

Clear the current input. Use this if you change your mind about executing the statement that you are entering.

- `connect [db_name host_name]], \r [db_name host_name]`

Reconnect to the server. The optional database name and host name arguments may be given to specify the default database or the host where the server is running. If omitted, the current values are used.

- `delimiter str, \d str`

Change the string that `mysql` interprets as the separator between SQL statements. The default is the semicolon character (“;”).

The delimiter string can be specified as an unquoted or quoted argument on the `delimiter` command line. Quoting can be done with either single quote (‘), double quote (“), or backtick (`) characters. To include a quote within a quoted string, either quote the string with a different quote character or escape the quote with a backslash (“\”) character. Backslash should be avoided outside of quoted strings because it is the escape character for MySQL. For an unquoted argument, the delimiter is read up to the first space or end of line. For a quoted argument, the delimiter is read up to the matching quote on the line.

`mysql` interprets instances of the delimiter string as a statement delimiter anywhere it occurs, except within quoted strings. Be careful about defining a delimiter that might occur within other words. For example, if you define the delimiter as `x`, you will be unable to use the word `INDEX` in statements. `mysql` interprets this as `INDE` followed by the delimiter `x`.

When the delimiter recognized by `mysql` is set to something other than the default of “;”, instances of that character are sent to the server without interpretation. However, the server itself still interprets “;” as a statement delimiter and processes statements accordingly. This behavior on the server side comes into play for multiple-statement execution (see [Section 22.8.13, “C API Support for Multiple Statement Execution”](#)), and for parsing the body of stored procedures and functions, triggers, and events (see [Section 19.1, “Defining Stored Programs”](#)).

- `edit, \e`

Edit the current input statement. `mysql` checks the values of the `EDITOR` and `VISUAL` environment variables to determine which editor to use. The default editor is `vi` if neither variable is set.

The `edit` command works only in Unix.

- `ego, \G`

Send the current statement to the server to be executed and display the result using vertical format.

- `exit, \q`

Exit `mysql`.

- `go, \g`

Send the current statement to the server to be executed.

- `nopager, \n`

Disable output paging. See the description for `pager`.

The `nopager` command works only in Unix.

- `notee, \t`

Disable output copying to the tee file. See the description for `tee`.

- `nowarning, \w`

Enable display of warnings after each statement.

- `pager [command], \P [command]`

Enable output paging. By using the `--pager [280]` option when you invoke `mysql`, it is possible to browse or search query results in interactive mode with Unix programs such as `less`, `more`, or any other similar program. If you specify no value for the option, `mysql` checks the value of the `PAGER` environment variable and sets the pager to that. Pager functionality works only in interactive mode.

Output paging can be enabled interactively with the `pager` command and disabled with `nopager`. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or `stdout` if no pager was specified.

Output paging works only in Unix because it uses the `popen()` function, which does not exist on Windows. For Windows, the `tee` option can be used instead to save query output, although it is not as convenient as `pager` for browsing output in some situations.

- `print, \p`

Print the current input statement without executing it.

- `prompt [str], \R [str]`

Reconfigure the `mysql` prompt to the given string. The special character sequences that can be used in the prompt are described later in this section.

If you specify the `prompt` command with no argument, `mysql` resets the prompt to the default of `mysql>`.

- `quit, \q`

Exit `mysql`.

- `rehash, \#`

Rebuild the completion hash that enables database, table, and column name completion while you are entering statements. (See the description for the `--auto-rehash [276]` option.)

- `source file_name, \. file_name`

Read the named file and executes the statements contained therein. On Windows, you can specify path name separators as `/` or `\\`.

- `status, \s`

Provide status information about the connection and the server you are using. If you are running in `--safe-updates [281]` mode, `status` also prints the values for the `mysql` variables that affect your queries.

- `system command, \! command`

Execute the given command using your default command interpreter.

The `system` command works only in Unix.

- `tee [file_name],\T [file_name]`

By using the `--tee` [282] option when you invoke `mysql`, you can log statements and their output. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. `mysql` flushes results to the file after each statement, just before it prints its next prompt. Tee functionality works only in interactive mode.

You can enable this feature interactively with the `tee` command. Without a parameter, the previous file is used. The `tee` file can be disabled with the `notee` command. Executing `tee` again re-enables logging.

- `use db_name,\u db_name`

Use `db_name` as the default database.

- `warnings,\W`

Enable display of warnings after each statement (if there are any).

Here are a few tips about the `pager` command:

- You can use it to write to a file and the results go only to the file:

```
mysql> pager cat > /tmp/log.txt
```

You can also pass any options for the program that you want to use as your pager:

```
mysql> pager less -n -i -S
```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to `less` can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within `less` to switch the horizontal-browse mode on and off. For more information, read the `less` manual page:

```
shell> man less
```

- The `-F` and `-X` options may be used with `less` to cause it to exit if output fits on one screen, which is convenient when no scrolling is necessary:

```
mysql> pager less -n -i -S -F -X
```

- You can specify very complex pager commands for handling query output:

```
mysql> pager cat | tee /dr1/tmp/res.txt \  
      | tee /dr2/tmp/res2.txt | less -n -i -S
```

In this example, the command would send query results to two files in two different directories on two different file systems mounted on `/dr1` and `/dr2`, yet still display the results onscreen using `less`.

You can also combine the `tee` and `pager` functions. Have a `tee` file enabled and `pager` set to `less`, and you are able to browse the results using the `less` program and still have everything appended into a file the same time. The difference between the Unix `tee` used with the `pager` command and the `mysql` built-in `tee` command is that the built-in `tee` works even if you do not have the Unix `tee` available. The built-in `tee` also logs everything that is printed on the screen, whereas the Unix `tee` used with `pager` does not log quite that much. Additionally, `tee` file logging can be turned on and off interactively from within `mysql`. This is useful when you want to log some queries to a file, but not others.

The `prompt` command reconfigures the default `mysql>` prompt. The string for defining the prompt can contain the following special sequences.

Option	Description
\c	A counter that increments for each statement you issue
\D	The full current date
\d	The default database
\h	The server host
\l	The current delimiter
\m	Minutes of the current time
\n	A newline character
\O	The current month in three-letter format (Jan, Feb, ...)
\o	The current month in numeric format
\P	am/pm
\p	The current TCP/IP port or socket file
\R	The current time, in 24-hour military time (0–23)
\r	The current time, standard 12-hour time (1–12)
\S	Semicolon
\s	Seconds of the current time
\t	A tab character
\U	Your full <code>user_name@host_name</code> account name
\u	Your user name
\v	The server version
\w	The current day of the week in three-letter format (Mon, Tue, ...)
\Y	The current year, four digits
\y	The current year, two digits
_	A space
\	A space (a space follows the backslash)
\'	Single quote
\"	Double quote
\\	A literal “\” backslash character
\x	<code>x</code> , for any “ <code>x</code> ” not listed above

You can set the prompt in several ways:

- *Use an environment variable.* You can set the `MYSQL_PS1` environment variable to a prompt string. For example:

```
shell> export MYSQL_PS1="(\\u@\\h) [\\d]> "
```

- *Use a command-line option.* You can set the `--prompt` [280] option on the command line to `mysql`. For example:

```
shell> mysql --prompt="(\\u@\\h) [\\d]> "
(user@host) [database]>
```

- *Use an option file.* You can set the `prompt` option in the `[mysql]` group of any MySQL option file, such as `/etc/my.cnf` or the `.my.cnf` file in your home directory. For example:

```
[mysql]
prompt=(\u@\h) [\d]>\_
```

In this example, note that the backslashes are doubled. If you set the prompt using the `prompt` option in an option file, it is advisable to double the backslashes when using the special prompt options. There is some overlap in the set of permissible prompt options and the set of special escape sequences that are recognized in option files. (The rules for escape sequences in option files are listed in [Section 4.2.3.3, “Using Option Files”](#).) The overlap may cause you problems if you use single backslashes. For example, `\s` is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in `HH:MM:SS>` format:

```
[mysql]
prompt="\r:\m:\s> "
```

- *Set the prompt interactively.* You can change your prompt interactively by using the `prompt` (or `\R`) command. For example:

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

4.5.1.3. mysql History File

On Unix, the `mysql` client writes a record of executed statements to a history file. By default, this file is named `.mysql_history` and is created in your home directory. To specify a different file, set the value of the `MYSQL_HISTFILE` environment variable.

The `.mysql_history` should be protected with a restrictive access mode because sensitive information might be written to it, such as the text of SQL statements that contain passwords. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#).

`mysql` does not write statements to the history file when used noninteractively (for example, when reading input from a file or a pipe). It is also possible to explicitly suppress logging of statements to the history file by using the `--batch` [276] or `--execute` [278] option.

If you do not want to maintain a history file, first remove `.mysql_history` if it exists, and then use either of the following techniques:

- Set the `MYSQL_HISTFILE` variable to `/dev/null`. To cause this setting to take effect each time you log in, put the setting in one of your shell's startup files.
- Create `.mysql_history` as a symbolic link to `/dev/null`:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

You need do this only once.

4.5.1.4. mysql Server-Side Help

```
mysql> help search_string
```

If you provide an argument to the `help` command, `mysql` uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. The proper operation of this command requires that the help tables in the `mysql` database be initialized with help topic information (see [Section 5.1.10, “Server-Side Help”](#)).

If there is no match for the search string, the search fails:

```
mysql> help me

Nothing found
Please try to run 'help contents' for a list of all accessible topics
```

Use `help contents` to see a list of the help categories:

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following categories:
  Account Management
  Administration
  Data Definition
  Data Manipulation
  Data Types
  Functions
  Functions and Modifiers for Use with GROUP BY
  Geographic Features
  Language Structure
  Plugins
  Storage Engines
  Stored Routines
  Table Maintenance
  Transactions
  Triggers
```

If the search string matches multiple items, `mysql` shows a list of matching topics:

```
mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
  SHOW
  SHOW BINARY LOGS
  SHOW ENGINE
  SHOW LOGS
```

Use a topic as the search string to see the help entry for that topic:

```
mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as
part of the procedure described in [purge-binary-logs], that shows how
to determine which logs can be purged.

mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name      | File_size |
+-----+-----+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+-----+-----+
```

4.5.1.5. Executing SQL Statements from a Text File

The `mysql` client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell `mysql` to read its input from that file. To do so, create a text file `text_file` that contains the statements you wish to execute. Then invoke `mysql` as shown here:

```
shell> mysql db_name < text_file
```

If you place a `USE db_name` statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are already running `mysql`, you can execute an SQL script file using the `source` command or `\.` command:

```
mysql> source file_name
mysql> \. file_name
```

Sometimes you may want your script to display progress information to the user. For this you can insert statements like this:

```
SELECT '<info_to_display>' AS ' ';
```

The statement shown outputs `<info_to_display>`.

You can also invoke `mysql` with the `--verbose` [282] option, which causes each statement to be displayed before the result that it produces.

`mysql` ignores Unicode byte order mark (BOM) characters at the beginning of input files. Previously, it read them and sent them to the server, resulting in a syntax error. Presence of a BOM does not cause `mysql` to change its default character set. To do that, invoke `mysql` with an option such as `--default-character-set=utf8` [277].

For more information about batch mode, see [Section 3.5, “Using `mysql` in Batch Mode”](#).

4.5.1.6. `mysql` Tips

This section describes some techniques that can help you use `mysql` more effectively.

4.5.1.6.1. Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with `\G` instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,\G
***** 1. row *****
  msg_nro: 3068
    date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Monty
      reply: monty@no.spam.com
mail_to: "Thimble Smith" <tim@no.spam.com>
      sbj: UTF-8
      txt: >>>> "Thimble" == Thimble Smith writes:

Thimble> Hi. I think this is a good idea. Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.

Yes, please do that.

Regards,
Monty
```

```
file: inbox-jani-1
hash: 190402944
1 row in set (0.09 sec)
```

4.5.1.6.2. Using the `--safe-updates` [281] Option

For beginners, a useful startup option is `--safe-updates` [281] (or `--i-am-a-dummy` [281], which has the same effect). It is helpful for cases when you might have issued a `DELETE FROM tbl_name` statement but forgotten the `WHERE` clause. Normally, such a statement deletes all rows from the table. With `--safe-updates` [281], you can delete rows only by specifying the key values that identify them. This helps prevent accidents.

When you use the `--safe-updates` [281] option, `mysql` issues the following statement when it connects to the MySQL server:

```
SET sql_safe_updates=1, sql_select_limit=1000, sql_max_join_size=1000000;
```

See [Section 5.1.4, “Server System Variables”](#).

The `SET` statement has the following effects:

- You are not permitted to execute an `UPDATE` or `DELETE` statement unless you specify a key constraint in the `WHERE` clause or provide a `LIMIT` clause (or both). For example:

```
UPDATE tbl_name SET not_key_column=val WHERE key_column=val;
UPDATE tbl_name SET not_key_column=val LIMIT 1;
```

- The server limits all large `SELECT` results to 1,000 rows unless the statement includes a `LIMIT` clause.
- The server aborts multiple-table `SELECT` statements that probably need to examine more than 1,000,000 row combinations.

To specify limits different from 1,000 and 1,000,000, you can override the defaults by using the `--select_limit` and `--max_join_size` options:

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

4.5.1.6.3. Disabling `mysql` Auto-Reconnect

If the `mysql` client loses its connection to the server while sending a statement, it immediately and automatically tries to reconnect once to the server and send the statement again. However, even if `mysql` succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables. Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted between the first and second statements without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+-----+
| a     |
```

```
+-----+
| NULL |
+-----+
1 row in set (0.05 sec)
```

The `@a` user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have `mysql` terminate with an error if the connection has been lost, you can start the `mysql` client with the `--skip-reconnect` [281] option.

For more information about auto-reconnect and its effect on state information when a reconnection occurs, see [Section 22.8.12, “Controlling Automatic Reconnection Behavior”](#).

4.5.2. `mysqladmin` — Client for Administering a MySQL Server

`mysqladmin` is a client for performing administrative operations. You can use it to check the server's configuration and current status, to create and drop databases, and more.

Invoke `mysqladmin` like this:

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

`mysqladmin` supports the following commands. Some of the commands take an argument following the command name.

- `create db_name`
Create a new database named `db_name`.
- `debug`
Tell the server to write debug information to the error log.
This includes information about the Event Scheduler. See [Section 19.4.5, “Event Scheduler Status”](#).
- `drop db_name`
Delete the database named `db_name` and all its tables.
- `extended-status`
Display the server status variables and their values.
- `flush-hosts`
Flush all information in the host cache.
- `flush-logs`
Flush all logs.
- `flush-privileges`
Reload the grant tables (same as `reload`).
- `flush-status`
Clear status variables.
- `flush-tables`
Flush all tables.

- `flush-threads`

Flush the thread cache.

- `kill id,id,...`

Kill server threads. If multiple thread ID values are given, there must be no spaces in the list.

- `old-password new-password`

This is like the `password` command but stores the password using the old (pre-4.1) password-hashing format. (See [Section 6.1.2.4, “Password Hashing in MySQL”](#).)

- `password new-password`

Set a new password. This changes the password to `new-password` for the account that you use with `mysqladmin` for connecting to the server. Thus, the next time you invoke `mysqladmin` (or any other client program) using the same account, you will need to specify the new password.

If the `new-password` value contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotation marks. On Windows, be sure to use double quotation marks rather than single quotation marks; single quotation marks are not stripped from the password, but rather are interpreted as part of the password. For example:

```
shell> mysqladmin password "my new password"
```

As of MySQL 5.5.3, the new password can be omitted following the `password` command. In this case, `mysqladmin` prompts for the password value, which enables you to avoid specifying the password on the command line. Omitting the password value should be done only if `password` is the final command on the `mysqladmin` command line. Otherwise, the next argument is taken as the password.



Caution

Do not use this command used if the server was started with the `--skip-grant-tables` [461] option. No password change will be applied. This is true even if you precede the `password` command with `flush-privileges` on the same command line to re-enable the grant tables because the flush operation occurs after you connect. However, you can use `mysqladmin flush-privileges` to re-enable the grant table and then use a separate `mysqladmin password` command to change the password.

- `ping`

Check whether the server is available. The return status from `mysqladmin` is 0 if the server is running, 1 if it is not. This is 0 even in case of an error such as `Access denied`, because this means that the server is running but refused the connection, which is different from the server not running.

- `processlist`

Show a list of active server threads. This is like the output of the `SHOW PROCESSLIST` statement. If the `--verbose` [300] option is given, the output is like that of `SHOW FULL PROCESSLIST`. (See [Section 13.7.5.30, “SHOW PROCESSLIST Syntax”](#).)

- `reload`

Reload the grant tables.

- `refresh`

Flush all tables and close and open log files.

- `shutdown`

Stop the server.

- `start-slave`

Start replication on a slave server.

- `status`

Display a short server status message.

- `stop-slave`

Stop replication on a slave server.

- `variables`

Display the server system variables and their values.

- `version`

Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host      | db | Command | Time | State | Info |
+-----+-----+-----+-----+-----+-----+-----+
| 51 | monty | localhost |    | Query   | 0    |      | show processlist |
+-----+-----+-----+-----+-----+-----+-----+
Uptime: 1473624  Threads: 1  Questions: 39487
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

The `mysqladmin status` command result displays the following values:

- `Uptime` [611]

The number of seconds the MySQL server has been running.

- `Threads`

The number of active threads (clients).

- `Questions` [605]

The number of questions (queries) from clients since the server was started.

- `Slow queries`

The number of queries that have taken more than `long_query_time` [517] seconds. See [Section 5.2.5, “The Slow Query Log”](#).

- `Opens`

The number of tables the server has opened.

- `Flush tables`

The number of `flush-*`, `refresh`, and `reload` commands the server has executed.

- `Open tables`

The number of tables that currently are open.

- `Memory in use`

The amount of memory allocated directly by `mysqld`. This value is displayed only when MySQL has been compiled with `safemalloc`, which is available only before MySQL 5.5.6.

- `Maximum memory used`

The maximum amount of memory allocated directly by `mysqld`. This value is displayed only when MySQL has been compiled with `safemalloc`, which is available only before MySQL 5.5.6.

If you execute `mysqladmin shutdown` when connecting to a local server using a Unix socket file, `mysqladmin` waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

`mysqladmin` supports the following options, which can be specified on the command line or in the `[mysqladmin]` and `[client]` groups of an option file. `mysqladmin` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.3. `mysqladmin` Options

Format	Option File	Description	Introduced
<code>--compress [298]</code>	<code>compress [298]</code>	Compress all information sent between the client and the server	
<code>--connect_timeout=seconds [300]</code>	<code>connect_timeout [300]</code>	The number of seconds before connection timeout	
<code>--count=# [298]</code>	<code>count [298]</code>	The number of iterations to make for repeated command execution	
<code>--debug[=debug_options] [298]</code>	<code>debug [298]</code>	Write a debugging log	
<code>--debug-check [298]</code>	<code>debug-check [298]</code>	Print debugging information when the program exits	
<code>--debug-info [298]</code>	<code>debug-info [298]</code>	Print debugging information, memory and CPU statistics when the program exits	
<code>--default-auth=plugin [298]</code>	<code>default-auth=plugin [298]</code>	The authentication plugin to use	5.5.9
<code>--default-character-set=charset_name [298]</code>	<code>default-character-set [298]</code>	Use <code>charset_name</code> as the default character set	
<code>--enable-cleartext-plugin [298]</code>	<code>enable-cleartext-plugin [298]</code>	Enable cleartext authentication plugin	5.5.27
<code>--force [298]</code>	<code>force [298]</code>	Continue even if an SQL error occurs	
<code>--help [297]</code>		Display help message and exit	

Format	Option File	Description	Introduced
-- host=host_name [298]	host [298]	Connect to the MySQL server on the given host	
--no-beep [299]	no-beep [299]	Do not beep when errors occur	
-- password[=password] [299]	password [299]	The password to use when connecting to the server	
--pipe [299]		On Windows, connect to server using a named pipe	
--plugin- dir=path [299]	plugin- dir=path [299]	The directory where plugins are located	5.5.9
-- port=port_num [299]	port [299]	The TCP/IP port number to use for the connection	
-- protocol=type [299]	protocol [299]	The connection protocol to use	
--relative [299]	relative [299]	Show the difference between the current and previous values when used with the --sleep option	
-- shutdown_timeout=seconds [300]	shutdown_timeout [300]	Maximum number of seconds to wait for server shutdown	
--silent [299]	silent [299]	Silent mode	
-- sleep=delay [299]	sleep [299]	Execute commands repeatedly, sleeping for delay seconds in between	
-- socket=path [299]	socket [299]	For connections to localhost	
--ssl- ca=file_name [299]	ssl-ca [299]	The path to a file that contains a list of trusted SSL CAs	
--ssl- capath=dir_name [299]	ssl-capath [299]	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl- cert=file_name [299]	ssl-cert [299]	The name of the SSL certificate file to use for establishing a secure connection	
--ssl- cipher=cipher_list [299]	ssl-cipher [299]	A list of allowable ciphers to use for SSL encryption	
--ssl- key=file_name [299]	ssl-key [299]	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server- cert [299]	ssl-verify-server- cert [299]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
-- user=user_name, [300]	user [300]	MySQL user name to use when connecting to server	
--verbose [300]		Verbose mode	
--version [300]		Display version information and exit	
--vertical [300]	vertical [300]	Print query output rows vertically (one line per column value)	
--wait [300]	wait [300]	If the connection cannot be established, wait and retry instead of aborting	

- --help [297], -?

Display a help message and exit.

- `--bind-address=ip_address` [277]

On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

This option is supported only in the version of `mysqladmin` that is supplied with MySQL Cluster. It is not available in standard MySQL Server 5.5 releases.

- `--character-sets-dir=path` [298]

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--compress` [298], `-C`

Compress all information sent between the client and the server if both support compression.

- `--count=N` [298], `-c N`

The number of iterations to make for repeated command execution if the `--sleep` [299] option is given.

- `--debug[=debug_options]` [298], `-# [debug_options]`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysqladmin.trace'`.

- `--debug-check` [298]

Print some debugging information when the program exits.

- `--debug-info` [298]

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin` [298]

The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.9.

- `--default-character-set=charset_name` [298]

Use `charset_name` as the default character set. See [Section 10.5, “Character Set Configuration”](#).

- `--enable-cleartext-plugin` [298]

Enable the `mysql_clear_password` cleartext authentication plugin. (See [Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”](#).) This option was added in MySQL 5.5.27.

- `--force` [298], `-f`

Do not ask for confirmation for the `drop db_name` command. With multiple commands, continue even if an error occurs.

- `--host=host_name` [298], `-h host_name`

Connect to the MySQL server on the given host.

- `--no-beep` [299], `-b`

Suppress the warning beep that is emitted by default for errors such as a failure to connect to the server.

- `--password[=password]` [299], `-p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` [299] or `-p` option on the command line, `mysqladmin` prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--pipe` [299], `-W`

On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path` [299]

The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [298] option is used to specify an authentication plugin but `mysqladmin` does not find it. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.9.

- `--port=port_num` [299], `-P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [299]

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see [Section 4.2.2, “Connecting to the MySQL Server”](#).

- `--relative` [299], `-r`

Show the difference between the current and previous values when used with the `--sleep` [299] option. This option works only with the `extended-status` command.

- `--silent` [299], `-s`

Exit silently if a connection to the server cannot be established.

- `--sleep=delay` [299], `-i delay`

Execute commands repeatedly, sleeping for `delay` seconds in between. The `--count` [298] option determines the number of iterations. If `--count` [298] is not given, `mysqladmin` executes commands indefinitely until interrupted.

- `--socket=path` [299], `-S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` [749] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See [Section 6.3.8.4, “SSL Command Options”](#).

- `--user=user_name` [300], `-u user_name`

The MySQL user name to use when connecting to the server.

- `--verbose` [300], `-v`

Verbose mode. Print more information about what the program does.

- `--version` [300], `-V`

Display version information and exit.

- `--vertical` [300], `-E`

Print output vertically. This is similar to `--relative` [299], but prints output vertically.

- `--wait[=count]` [300], `-w[count]`

If the connection cannot be established, wait and retry instead of aborting. If a *count* value is given, it indicates the number of times to retry. The default is one time.

You can also set the following variables by using `--var_name=value`. The `--set-variable` format is deprecated and was removed in MySQL 5.5.3. syntax:

- `connect_timeout`

The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- `shutdown_timeout`

The maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).

4.5.3. mysqlcheck — A Table Maintenance Program

The `mysqlcheck` client performs table maintenance: It checks, repairs, optimizes, or analyzes tables.

Each table is locked and therefore unavailable to other sessions while it is being processed, although for check operations, the table is locked with a `READ` lock only (see [Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”](#), for more information about `READ` and `WRITE` locks). Table maintenance operations can be time-consuming, particularly for large tables. If you use the `--databases` [304] or `--all-databases` [303] option to process all tables in one or more databases, an invocation of `mysqlcheck` might take a long time. (This is also true for `mysql_upgrade` because that program invokes `mysqlcheck` to check all tables and repair them if necessary.)

`mysqlcheck` is similar in function to `myisamchk`, but works differently. The main operational difference is that `mysqlcheck` must be used when the `mysqld` server is running, whereas `myisamchk` should be used when it is not. The benefit of using `mysqlcheck` is that you do not have to stop the server to perform table maintenance.

`mysqlcheck` uses the SQL statements `CHECK TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, and `OPTIMIZE TABLE` in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statements to the server to be executed. For details about which storage engines each statement works with, see the descriptions for those statements in [Section 13.7.2, “Table Maintenance Statements”](#).

The `MyISAM` storage engine supports all four maintenance operations, so `mysqlcheck` can be used to perform any of them on `MyISAM` tables. Other storage engines do not necessarily support all operations. In such cases, an error message is displayed. For example, if `test.t` is a `MEMORY` table, an attempt to check it produces this result:

```
shell> mysqlcheck test t
test.t
note      : The storage engine for the table doesn't support check
```

If `mysqlcheck` is unable to repair a table, see [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#) for manual table repair strategies. This will be the case, for example, for `InnoDB` tables, which can be checked with `CHECK TABLE`, but not repaired with `REPAIR TABLE`.



Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

There are three general ways to invoke `mysqlcheck`:

```
shell> mysqlcheck [options] db_name [tbl_name ...]
shell> mysqlcheck [options] --databases db_name ...
shell> mysqlcheck [options] --all-databases
```

If you do not name any tables following `db_name` or if you use the `--databases` [\[304\]](#) or `--all-databases` [\[303\]](#) option, entire databases are checked.

`mysqlcheck` has a special feature compared to other client programs. The default behavior of checking tables (`--check` [\[304\]](#)) can be changed by renaming the binary. If you want to have a tool that repairs tables by default, you should just make a copy of `mysqlcheck` named `mysqlrepair`, or make a symbolic link to `mysqlcheck` named `mysqlrepair`. If you invoke `mysqlrepair`, it repairs tables.

The names shown in the following table can be used to change `mysqlcheck` default behavior.

Command	Meaning
<code>mysqlrepair</code>	The default option is <code>--repair</code> [306]
<code>mysqlanalyze</code>	The default option is <code>--analyze</code> [303]
<code>mysqloptimize</code>	The default option is <code>--optimize</code> [305]

`mysqlcheck` supports the following options, which can be specified on the command line or in the `[mysqlcheck]` and `[client]` groups of an option file. `mysqlcheck` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.4. `mysqlcheck` Options

Format	Option File	Description	Introduced
<code>--all-databases</code> [303]	<code>all-databases</code> [303]	Check all tables in all databases	
<code>--all-in-1</code> [303]	<code>all-in-1</code> [303]	Execute a single statement for each database that names all the tables from that database	
<code>--analyze</code> [303]	<code>analyze</code> [303]	Analyze the tables	
<code>--auto-repair</code> [304]	<code>auto-repair</code> [304]	If a checked table is corrupted, automatically fix it	

Format	Option File	Description	Introduced
<code>--character-sets-dir=path</code> [304]	<code>character-sets-dir</code> [304]	The directory where character sets are installed	
<code>--check</code> [304]	<code>check</code> [304]	Check the tables for errors	
<code>--check-only-changed</code> [304]	<code>check-only-changed</code> [304]	Check only tables that have changed since the last check	
<code>--check-upgrade</code> [304]	<code>check-upgrade</code> [304]	Invoke CHECK TABLE with the FOR UPGRADE option	
<code>--compress</code> [304]	<code>compress</code> [304]	Compress all information sent between the client and the server	
<code>--databases</code> [304]	<code>databases</code> [304]	Process all tables in the named databases	
<code>--debug[=debug_options]</code> [304]	<code>debug</code> [304]	Write a debugging log	
<code>--debug-check</code> [304]	<code>debug-check</code> [304]	Print debugging information when the program exits	
<code>--debug-info</code> [304]	<code>debug-info</code> [304]	Print debugging information, memory and CPU statistics when the program exits	
<code>--default-auth=plugin</code> [305]	<code>default-auth=plugin</code> [305]	The authentication plugin to use	5.5.10
<code>--default-character-set=charset_name</code> [304]	<code>default-character-set</code> [304]	Use charset_name as the default character set	
<code>--extended</code> [304]	<code>extended</code> [304]	Check and repair tables	
<code>--fast</code> [305]	<code>fast</code> [305]	Check only tables that have not been closed properly	
<code>--fix-db-names</code> [305]	<code>fix-db-names</code> [305]	Convert database names to 5.1 format	
<code>--fix-table-names</code> [305]	<code>fix-table-names</code> [305]	Convert table names to 5.1 format	
<code>--force</code> [305]	<code>force</code> [305]	Continue even if an SQL error occurs	
<code>--help</code> [303]		Display help message and exit	
<code>--host=host_name</code> [305]	<code>host</code> [305]	Connect to the MySQL server on the given host	
<code>--medium-check</code> [305]	<code>medium-check</code> [305]	Do a check that is faster than an --extended operation	
<code>--optimize</code> [305]	<code>optimize</code> [305]	Optimize the tables	
<code>--password[=password]</code> [305]	<code>password</code> [305]	The password to use when connecting to the server	
<code>--pipe</code> [305]		On Windows, connect to server using a named pipe	
<code>--plugin-dir=path</code> [306]	<code>plugin-dir=path</code> [306]	The directory where plugins are located	5.5.10
<code>--port=port_num</code> [306]	<code>port</code> [306]	The TCP/IP port number to use for the connection	

Format	Option File	Description	Introduced
<code>--protocol=type [306]</code>	<code>protocol [306]</code>	The connection protocol to use	
<code>--quick [306]</code>	<code>quick [306]</code>	The fastest method of checking	
<code>--repair [306]</code>	<code>repair [306]</code>	Perform a repair that can fix almost anything except unique keys that are not unique	
<code>--silent [306]</code>	<code>silent [306]</code>	Silent mode	
<code>--socket=path [306]</code>	<code>socket [306]</code>	For connections to localhost	
<code>--ssl-ca=file_name [306]</code>	<code>ssl-ca [306]</code>	The path to a file that contains a list of trusted SSL CAs	
<code>--ssl-capath=dir_name [306]</code>	<code>ssl-capath [306]</code>	The path to a directory that contains trusted SSL CA certificates in PEM format	
<code>--ssl-cert=file_name [306]</code>	<code>ssl-cert [306]</code>	The name of the SSL certificate file to use for establishing a secure connection	
<code>--ssl-cipher=cipher_list [306]</code>	<code>ssl-cipher [306]</code>	A list of allowable ciphers to use for SSL encryption	
<code>--ssl-key=file_name [306]</code>	<code>ssl-key [306]</code>	The name of the SSL key file to use for establishing a secure connection	
<code>--ssl-verify-server-cert [306]</code>	<code>ssl-verify-server-cert [306]</code>	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
<code>--tables [306]</code>	<code>tables [306]</code>	Overrides the <code>--databases</code> or <code>-B</code> option	
<code>--use-frm [306]</code>	<code>use-frm [306]</code>	For repair operations on MyISAM tables	
<code>--user=user_name, [306]</code>	<code>user [306]</code>	MySQL user name to use when connecting to server	
<code>--verbose [306]</code>		Verbose mode	
<code>--version [307]</code>		Display version information and exit	
<code>--write-binlog [307]</code>	<code>write-binlog [307]</code>	Log ANALYZE, OPTIMIZE, REPAIR statements to binary log. <code>--skip-write-binlog</code> adds <code>NO_WRITE_TO_BINLOG</code> to these statements.	

- `--help [303]`, `-?`

Display a help message and exit.

- `--all-databases [303]`, `-A`

Check all tables in all databases. This is the same as using the `--databases [304]` option and naming all the databases on the command line.

- `--all-in-1 [303]`, `-1`

Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze [303]`, `-a`

Analyze the tables.

- `--auto-repair` [304]

If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--bind-address=ip_address` [277]

On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

This option is supported only in the version of `mysqlcheck` that is supplied with MySQL Cluster. It is not available in standard MySQL Server 5.5 releases.

- `--character-sets-dir=path` [304]

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--check` [304], `-c`

Check the tables for errors. This is the default operation.

- `--check-only-changed` [304], `-C`

Check only tables that have changed since the last check or that have not been closed properly.

- `--check-upgrade` [304], `-g`

Invoke `CHECK TABLE` with the `FOR UPGRADE` option to check tables for incompatibilities with the current version of the server. This option automatically enables the `--fix-db-names` [305] and `--fix-table-names` [305] options.

- `--compress` [304]

Compress all information sent between the client and the server if both support compression.

- `--databases` [304], `-B`

Process all tables in the named databases. Normally, `mysqlcheck` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names.

- `--debug[=debug_options]` [304], `-# [debug_options]`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o'`.

- `--debug-check` [304]

Print some debugging information when the program exits.

- `--debug-info` [304]

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-character-set=charset_name` [304]

Use `charset_name` as the default character set. See [Section 10.5, “Character Set Configuration”](#).

- `--extended` [304], `-e`

If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- `--default-auth=plugin` [305]

The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.10.

- `--fast` [305], `-F`

Check only tables that have not been closed properly.

- `--fix-db-names` [305]

Convert database names to 5.1 format. Only database names that contain special characters are affected.

- `--fix-table-names` [305]

Convert table names to 5.1 format. Only table names that contain special characters are affected. This option also applies to views.

- `--force` [305], `-f`

Continue even if an SQL error occurs.

- `--host=host_name` [305], `-h host_name`

Connect to the MySQL server on the given host.

- `--medium-check` [305], `-m`

Do a check that is faster than an `--extended` [304] operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--optimize` [305], `-o`

Optimize the tables.

- `--password[=password]` [305], `-p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` [305] or `-p` option on the command line, `mysqlcheck` prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--pipe` [305], `-W`

On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path` [306]

The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [305] option is used to specify an authentication plugin but `mysqlcheck` does not find it. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.10.

- `--port=port_num` [306], `-P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [306]

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see [Section 4.2.2, “Connecting to the MySQL Server”](#).

- `--quick` [306], `-q`

If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- `--repair` [306], `-r`

Perform a repair that can fix almost anything except unique keys that are not unique.

- `--silent` [306], `-s`

Silent mode. Print only error messages.

- `--socket=path` [306], `-S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` [749] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See [Section 6.3.8.4, “SSL Command Options”](#).

- `--tables` [306]

Override the `--databases` [304] or `-B` option. All name arguments following the option are regarded as table names.

- `--use-frm` [306]

For repair operations on `MyISAM` tables, get the table structure from the `.frm` file so that the table can be repaired even if the `.MYI` header is corrupted.

- `--user=user_name` [306], `-u user_name`

The MySQL user name to use when connecting to the server.

- `--verbose` [306], `-v`

Verbose mode. Print information about the various stages of program operation.

- `--version` [307], `-V`

Display version information and exit.

- `--write-binlog` [307]

This option is enabled by default, so that `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements generated by `mysqlcheck` are written to the binary log. Use `--skip-write-binlog` [307] to cause `NO_WRITE_TO_BINLOG` to be added to the statements so that they are not logged. Use the `--skip-write-binlog` [307] when these statements should not be sent to replication slaves or run when using the binary logs for recovery from backup.

4.5.4. `mysqldump` — A Database Backup Program

The `mysqldump` client is a backup program originally written by Igor Romanenko. It can be used to dump a database or a collection of databases for backup or transfer to another SQL server (not necessarily a MySQL server). The dump typically contains SQL statements to create the table, populate it, or both. However, `mysqldump` can also be used to generate files in CSV, other delimited text, or XML format.

`mysqldump` requires at least the `SELECT` [697] privilege for dumped tables, `SHOW VIEW` [697] for dumped views, and `LOCK TABLES` [696] if the `--single-transaction` [321] option is not used. Certain options might require other privileges as noted in the option descriptions.

If you are doing a backup on the server and your tables all are `MyISAM` tables, consider using the `mysqlhotcopy` instead because it can accomplish faster backups and faster restores. See [Section 4.6.9, “mysqlhotcopy — A Database Backup Program”](#).

There are three general ways to invoke `mysqldump`:

```
shell> mysqldump [options] db_name [tbl_name ...]
shell> mysqldump [options] --databases db_name ...
shell> mysqldump [options] --all-databases
```

If you do not name any tables following `db_name` or if you use the `--databases` [314] or `--all-databases` [313] option, entire databases are dumped.

`mysqldump` does not dump the `INFORMATION_SCHEMA` database by default. `mysqldump` dumps `INFORMATION_SCHEMA` only if you name it explicitly on the command line, although currently you must also use the `--skip-lock-tables` [317] option. Before MySQL 5.5 `mysqldump` silently ignores `INFORMATION_SCHEMA` even if you name it explicitly on the command line.

`mysqldump` does not dump the `performance_schema` database.

Before MySQL 5.5.25, `mysqldump` does not dump the `general_log` or `slow_query_log` tables for dumps of the `mysql` database. As of 5.5.25, the dump includes statements to recreate those tables so that they are not missing after reloading the dump file. Log table contents are not dumped.

`mysqldump` also does not dump the MySQL Cluster `ndbinfo` information database.

To see a list of the options your version of `mysqldump` supports, execute `mysqldump --help`.

Some `mysqldump` options are shorthand for groups of other options:

- Use of `--opt` [319] is the same as specifying `--add-drop-table` [313], `--add-locks` [313], `--create-options` [314], `--disable-keys` [315], `--extended-insert` [316], `--lock-tables` [317], `--quick` [320], and `--set-charset` [320]. All of the options that `--opt` [319] stands for also are on by default because `--opt` [319] is on by default.

- Use of `--compact` [314] is the same as specifying `--skip-add-drop-table` [313], `--skip-add-locks` [313], `--skip-comments` [321], `--skip-disable-keys` [315], and `--skip-set-charset` [320] options.

To reverse the effect of a group option, uses its `--skip-xxx` form (`--skip-opt` [321] or `--skip-compact` [314]). It is also possible to select only part of the effect of a group option by following it with options that enable or disable specific features. Here are some examples:

- To select the effect of `--opt` [319] except for some features, use the `--skip` option for each feature. To disable extended inserts and memory buffering, use `--opt` [319] `--skip-extended-insert` [316] `--skip-quick` [320]. (Actually, `--skip-extended-insert` [316] `--skip-quick` [320] is sufficient because `--opt` [319] is on by default.)
- To reverse `--opt` [319] for all features except index disabling and table locking, use `--skip-opt` [321] `--disable-keys` [315] `--lock-tables` [317].

When you selectively enable or disable the effect of a group option, order is important because options are processed first to last. For example, `--disable-keys` [315] `--lock-tables` [317] `--skip-opt` [321] would not have the intended effect; it is the same as `--skip-opt` [321] by itself.

`mysqldump` can retrieve and dump table contents row by row, or it can retrieve the entire content from a table and buffer it in memory before dumping it. Buffering in memory can be a problem if you are dumping large tables. To dump tables row by row, use the `--quick` [320] option (or `--opt` [319], which enables `--quick` [320]). The `--opt` [319] option (and hence `--quick` [320]) is enabled by default, so to enable memory buffering, use `--skip-quick` [320].

If you are using a recent version of `mysqldump` to generate a dump to be reloaded into a very old MySQL server, you should not use the `--opt` [319] or `--extended-insert` [316] option. Use `--skip-opt` [321] instead.

For additional information about `mysqldump`, see [Section 7.4, “Using `mysqldump` for Backups”](#).

`mysqldump` supports the following options, which can be specified on the command line or in the `[mysqldump]` and `[client]` groups of an option file. `mysqldump` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.5. `mysqldump` Options

Format	Option File	Description	Introduced	Removed
<code>--add-drop-database</code> [313]	<code>add-drop-database</code> [313]	Add a DROP DATABASE statement before each CREATE DATABASE statement		
<code>--add-drop-table</code> [313]	<code>add-drop-table</code> [313]	Add a DROP TABLE statement before each CREATE TABLE statement		
<code>--add-locks</code> [313]	<code>add-locks</code> [313]	Surround each table dump with LOCK TABLES and UNLOCK TABLES statements		
<code>--all-databases</code> [313]	<code>all-databases</code> [313]	Dump all tables in all databases		
<code>--allow-keywords</code> [313]	<code>allow-keywords</code> [313]	Allow creation of column names that are keywords		
<code>--apply-slave-statements</code> [313]	<code>apply-slave-statements</code> [313]	Include STOP SLAVE prior to CHANGE MASTER statement and START SLAVE at end of output	5.5.3	

Format	Option File	Description	Introduced	Removed
<code>--bind-address=ip_address</code> [313]	<code>bind-address</code> [313]	Use the specified network interface to connect to the MySQL Server	5.5.8	
<code>--comments</code> [314]	<code>comments</code> [314]	Add comments to the dump file		
<code>--compact</code> [314]	<code>compact</code> [314]	Produce more compact output		
<code>--compatible=name</code> [314]	<code>compatible</code> [314] <code>[,name,...]</code> [314]	Produce output that is more compatible with other database systems or with older MySQL servers		
<code>--complete-insert</code> [314]	<code>complete-insert</code> [314]	Use complete INSERT statements that include column names		
<code>--create-options</code> [314]	<code>create-options</code> [314]	Include all MySQL-specific table options in CREATE TABLE statements		
<code>--databases</code> [314]	<code>databases</code> [314]	Dump several databases		
<code>--debug[=debug_options]</code> [314]	<code>debug</code> [314] <code>[,debug_options]</code> [314]	Write a debugging log		
<code>--debug-check</code> [315]	<code>debug-check</code> [315]	Print debugging information when the program exits		
<code>--debug-info</code> [315]	<code>debug-info</code> [315]	Print debugging information, memory and CPU statistics when the program exits		
<code>--default-auth=plugin</code> [315]	<code>default-auth=plugin</code> [315]	The authentication plugin to use	5.5.9	
<code>--default-character-set=charset_name</code> [315]	<code>default-character-set</code> [315] <code>[,charset_name]</code> [315]	Use <code>charset_name</code> as the default character set		
<code>--delayed-insert</code> [315]	<code>delayed-insert</code> [315]	Write INSERT DELAYED statements rather than INSERT statements		
<code>--delete-master-logs</code> [315]	<code>delete-master-logs</code> [315]	On a master replication server, delete the binary logs after performing the dump operation		
<code>--disable-keys</code> [315]	<code>disable-keys</code> [315]	For each table, surround the INSERT statements with statements to disable and enable keys		
<code>--dump-date</code> [315]	<code>dump-date</code> [315]	Include dump date as "Dump completed on" comment if <code>--comments</code> is given		
<code>--dump-slave[=value]</code> [315]	<code>dump-slave</code> [315] <code>[,value]</code> [315]	Include CHANGE MASTER statement that lists binary log coordinates of slave's master	5.5.3	
<code>--events</code> [316]	<code>events</code> [316]	Dump events from the dumped databases		
<code>--extended-insert</code> [316]	<code>extended-insert</code> [316]	Use multiple-row INSERT syntax that include several VALUES lists		
<code>--fields-enclosed-by=string</code> [316]	<code>fields-enclosed-by</code> [316] <code>[,string]</code> [316]	This option is used with the <code>--tab</code> option and has the same meaning as the corresponding clause for LOAD DATA INFILE		

Format	Option File	Description	Introduced	Removed
--fields-escaped-by [316]	fields-escaped-by [316]	This option is used with the <code>--tab</code> option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--fields-optionally-enclosed-by=string [316]	fields-optionally-enclosed-by [316]	This option is used with the <code>--tab</code> option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--fields-terminated-by=string [316]	fields-terminated-by [316]	This option is used with the <code>--tab</code> option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--first-slave [316]	first-slave [316]	Deprecated; use <code>--lock-all-tables</code> instead		5.5.3
--flush-logs [316]	flush-logs [316]	Flush the MySQL server log files before starting the dump		
--flush-privileges [316]	flush-privileges [316]	Emit a FLUSH PRIVILEGES statement after dumping the mysql database		
--help [313]		Display help message and exit		
--hex-blob [316]	hex-blob [316]	Dump binary columns using hexadecimal notation (for example, 'abc' becomes 0x616263)		
--host [316]	host [316]	Host to connect to (IP address or hostname)		
--ignore-table=db_name.table [317]	ignore-table [317]	Do not dump the given table		
--include-master-host-port [317]	include-master-host-port [317]	Include MASTER_HOST/MASTER_PORT options in CHANGE MASTER statement produced with <code>--dump-slave</code>	5.5.3	
--insert-ignore [317]	insert-ignore [317]	Write INSERT IGNORE statements rather than INSERT statements		
--lines-terminated-by=string [317]	lines-terminated-by [317]	This option is used with the <code>--tab</code> option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--lock-all-tables [317]	lock-all-tables [317]	Lock all tables across all databases		
--lock-tables [317]	lock-tables [317]	Lock all tables before dumping them		
--log-error=file_name [317]	log-error [317]	Append warnings and errors to the named file		
--master-data[=value] [317]	master-data [317]	Write the binary log file name and position to the output		
--max_allowed_packet=value	max_allowed_packet [324]	Set the maximum packet length to send to or receive from the server		
--net_buffer_length=value [324]	net_buffer_length [324]	Set the buffer size for TCP/IP and socket communication		

Format	Option File	Description	Introduced	Removed
--no-autocommit [318]	no-autocommit [318]	Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements		
--no-create-db [318]	no-create-db [318]	This option suppresses the CREATE DATABASE statements		
--no-create-info [318]	no-create-info [318]	Do not write CREATE TABLE statements that re-create each dumped table		
--no-data [319]	no-data [319]	Do not dump table contents		
--no-set-names [319]	no-set-names [319]	Same as --skip-set-charset		
--no-tablespaces [319]	no-tablespaces [319]	Do not write any CREATE LOGFILE GROUP or CREATE TABLESPACE statements in output		
--opt [319]	opt [319]	Shorthand for --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset.		
--order-by-primary [319]	order-by-primary [319]	Dump each table's rows sorted by its primary key, or by its first unique index		
--password[=password] [319]	password [319]	The password to use when connecting to the server		
--pipe [319]		On Windows, connect to server using a named pipe		
--plugin-dir=path [319]	plugin-dir=path [319]	The directory where plugins are located	5.5.9	
--port=port_num [320]	port [320]	The TCP/IP port number to use for the connection		
--quick [320]	quick [320]	Retrieve rows for a table from the server a row at a time		
--quote-names [320]	quote-names [320]	Quote identifiers within backtick characters		
--replace [320]	replace [320]	Write REPLACE statements rather than INSERT statements		
--result-file=file [320]	result-file [320]	Direct output to a given file		
--routines [320]	routines [320]	Dump stored routines (procedures and functions) from the dumped databases		
--set-charset [320]	set-charset [320]	Add SET NAMES default_character_set to output		
--single-transaction [321]	single-transaction [321]	This option issues a BEGIN SQL statement before dumping data from the server		
--skip-add-drop-table [313]	skip-add-drop-table [313]	Do not add a DROP TABLE statement before each CREATE TABLE statement		
--skip-add-locks [313]	skip-add-locks [313]	Do not add locks		

Format	Option File	Description	Introduced	Removed
<code>--skip-comments</code> [321]	<code>skip-comments</code> [321]	Do not add comments to the dump file		
<code>--skip-compact</code> [314]	<code>skip-compact</code> [314]	Do not produce more compact output		
<code>--skip-disable-keys</code> [315]	<code>skip-disable-keys</code> [315]	Do not disable keys		
<code>--skip-extended-insert</code> [316]	<code>skip-extended-insert</code> [316]	Turn off extended-insert		
<code>--skip-opt</code> [321]	<code>skip-opt</code> [321]	Turn off the options set by <code>--opt</code>		
<code>--skip-quick</code> [320]	<code>skip-quick</code> [320]	Do not retrieve rows for a table from the server a row at a time		
<code>--skip-quote-names</code> [320]	<code>skip-quote-names</code> [320]	Do not quote identifiers		
<code>--skip-set-charset</code> [320]	<code>skip-set-charset</code> [320]	Suppress the SET NAMES statement		
<code>--skip-triggers</code> [322]	<code>skip-triggers</code> [322]	Do not dump triggers		
<code>--skip-tz-utc</code> [322]	<code>skip-tz-utc</code> [322]	Turn off tz-utc		
<code>--ssl-ca=file_name</code> [321]	<code>ssl-ca</code> [321]	The path to a file that contains a list of trusted SSL CAs		
<code>--ssl-capath=dir_name</code> [321]	<code>ssl-capath</code> [321]	The path to a directory that contains trusted SSL CA certificates in PEM format		
<code>--ssl-cert=file_name</code> [321]	<code>ssl-cert</code> [321]	The name of the SSL certificate file to use for establishing a secure connection		
<code>--ssl-cipher=cipher_list</code> [321]	<code>ssl-cipher</code> [321]	A list of allowable ciphers to use for SSL encryption		
<code>--ssl-key=file_name</code> [321]	<code>ssl-key</code> [321]	The name of the SSL key file to use for establishing a secure connection		
<code>--ssl-verify-server-cert</code> [321]	<code>ssl-verify-server-cert</code> [321]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server		
<code>--tab=path</code> [321]	<code>tab</code> [321]	Produce tab-separated data files		
<code>--tables</code> [322]	<code>tables</code> [322]	Override the <code>--databases</code> or <code>-B</code> option		
<code>--triggers</code> [322]	<code>triggers</code> [322]	Dump triggers for each dumped table		
<code>--tz-utc</code> [322]	<code>tz-utc</code> [322]	Add SET TIME_ZONE='+00:00' to the dump file		
<code>--user=user_name</code>	<code>user</code> [322]	MySQL user name to use when connecting to server		
<code>--verbose</code> [322]		Verbose mode		
<code>--version</code> [322]		Display version information and exit		

Format	Option File	Description	Introduced	Removed
-- where='where_condition' [322]	where [322] xml [322]	Dump only rows selected by the given WHERE condition		
--xml [322]	xml [322]	Produce XML output		

- `--help` [313], `-?`

Display a help message and exit.

- `--add-drop-database` [313]

Add a `DROP DATABASE` statement before each `CREATE DATABASE` statement. This option is typically used in conjunction with the `--all-databases` [313] or `--databases` [314] option because no `CREATE DATABASE` statements are written unless one of those options is specified.

- `--add-drop-table` [313]

Add a `DROP TABLE` statement before each `CREATE TABLE` statement.

- `--add-drop-trigger` [313]

Add a `DROP TRIGGER` statement before each `CREATE TRIGGER` statement.



Note

This option is supported only by `mysqldump` as supplied with MySQL Cluster. It is not available when using MySQL Server 5.5.

- `--add-locks` [313]

Surround each table dump with `LOCK TABLES` and `UNLOCK TABLES` statements. This results in faster inserts when the dump file is reloaded. See [Section 8.2.2.1, “Speed of INSERT Statements”](#).

- `--all-databases` [313], `-A`

Dump all tables in all databases. This is the same as using the `--databases` [314] option and naming all the databases on the command line.

- `--all-tablespaces` [313], `-Y`

Adds to a table dump all SQL statements needed to create any tablespaces used by an `NDBCLUSTER` table. This information is not otherwise included in the output from `mysqldump`. This option is currently relevant only to MySQL Cluster tables.

- `--allow-keywords` [313]

Permit creation of column names that are keywords. This works by prefixing each column name with the table name.

- `--apply-slave-statements` [313]

For a slave dump produced with the `--dump-slave` [315] option, add a `STOP SLAVE` statement before the `CHANGE MASTER TO` statement and a `START SLAVE` statement at the end of the output. This option was added in MySQL 5.5.3.

- `--bind-address=ip_address` [277]

On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

This option is supported only in the version of `mysqldump` that is supplied with MySQL Cluster. It is not available in standard MySQL Server 5.5 releases.

- `--character-sets-dir=path` [314]

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--comments` [314], `-i`

Write additional information in the dump file such as program version, server version, and host. This option is enabled by default. To suppress this additional information, use `--skip-comments` [321].

- `--compact` [314]

Produce more compact output. This option enables the `--skip-add-drop-table` [313], `--skip-add-locks` [313], `--skip-comments` [321], `--skip-disable-keys` [315], and `--skip-set-charset` [320] options.

- `--compatible=name` [314]

Produce output that is more compatible with other database systems or with older MySQL servers. The value of *name* can be `ansi`, `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`, `db2`, `maxdb`, `no_key_options`, `no_table_options`, or `no_field_options`. To use several values, separate them by commas. These values have the same meaning as the corresponding options for setting the server SQL mode. See [Section 5.1.7, “Server SQL Modes”](#).

This option does not guarantee compatibility with other servers. It only enables those SQL mode values that are currently available for making dump output more compatible. For example, `--compatible=oracle` [314] does not map data types to Oracle types or use Oracle comment syntax.

This option requires a server version of 4.1.0 or higher. With older servers, it does nothing.

- `--complete-insert` [314], `-c`

Use complete `INSERT` statements that include column names.

- `--compress` [314], `-C`

Compress all information sent between the client and the server if both support compression.

- `--create-options` [314]

Include all MySQL-specific table options in the `CREATE TABLE` statements.

- `--databases` [314], `-B`

Dump several databases. Normally, `mysqldump` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE` and `USE` statements are included in the output before each new database.

- `--debug[=debug_options]` [314], `-# [debug_options]`

Write a debugging log. A typical *debug_options* string is `'d:t:o,file_name'`. The default value is `'d:t:o,/tmp/mysqldump.trace'`.

- `--debug-check` [315]

Print some debugging information when the program exits.

- `--debug-info` [315]

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin` [315]

The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.9.

- `--default-character-set=charset_name` [315]

Use *charset_name* as the default character set. See [Section 10.5, “Character Set Configuration”](#). If no character set is specified, `mysqldump` uses `utf8`, and earlier versions use `latin1`.

- `--delayed-insert` [315]

Write `INSERT DELAYED` statements rather than `INSERT` statements.

- `--delete-master-logs` [315]

On a master replication server, delete the binary logs by sending a `PURGE BINARY LOGS` statement to the server after performing the dump operation. This option automatically enables `--master-data` [317].

- `--disable-keys` [315], `-K`

For each table, surround the `INSERT` statements with `/*!40000 ALTER TABLE tbl_name DISABLE KEYS */;` and `/*!40000 ALTER TABLE tbl_name ENABLE KEYS */;` statements. This makes loading the dump file faster because the indexes are created after all rows are inserted. This option is effective only for nonunique indexes of `MyISAM` tables. It has no effect for other tables.

- `--dump-date` [315]

If the `--comments` [314] option is given, `mysqldump` produces a comment at the end of the dump of the following form:

```
-- Dump completed on DATE
```

However, the date causes dump files taken at different times to appear to be different, even if the data are otherwise identical. `--dump-date` [315] and `--skip-dump-date` [315] control whether the date is added to the comment. The default is `--dump-date` [315] (include the date in the comment). `--skip-dump-date` [315] suppresses date printing.

- `--dump-slave[=value]` [315]

This option is similar to `--master-data` [317] except that it is used to dump a replication slave server to produce a dump file that can be used to set up another server as a slave that has the same master as the dumped server. It causes the dump output to include a `CHANGE MASTER TO` statement that indicates the binary log coordinates (file name and position) of the dumped slave's master (rather than the coordinates of the dumped server, as is done by the `--master-data` [317] option). These are the master server coordinates from which the slave should start replicating. This option was added in MySQL 5.5.3.

The option value is handled the same way as for `--master-data` [317] and has the same effect as `--master-data` [317] in terms of enabling or disabling other options and in how locking is handled.

In conjunction with `--dump-slave` [315], the `--apply-slave-statements` [313] and `--include-master-host-port` [317] options can also be used.

- `--events` [316], `-E`

Include Event Scheduler events for the dumped databases in the output.

- `--extended-insert` [316], `-e`

Use multiple-row `INSERT` syntax that include several `VALUES` lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

- `--fields-terminated-by=...` [316], `--fields-enclosed-by=...` [316], `--fields-optionally-enclosed-by=...` [316], `--fields-escaped-by=...` [316]

These options are used with the `--tab` [321] option and have the same meaning as the corresponding `FIELDS` clauses for `LOAD DATA INFILE`. See Section 13.2.6, “`LOAD DATA INFILE` Syntax”.

- `--first-slave` [316]

Deprecated. Use `--lock-all-tables` [317] instead. `--first-slave` [316] was removed in MySQL 5.5.3.

- `--flush-logs` [316], `-F`

Flush the MySQL server log files before starting the dump. This option requires the `RELOAD` [696] privilege. If you use this option in combination with the `--all-databases` [313] option, the logs are flushed *for each database dumped*. The exception is when using `--lock-all-tables` [317] or `--master-data` [317]: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked. If you want your dump and the log flush to happen at exactly the same moment, you should use `--flush-logs` [316] together with either `--lock-all-tables` [317] or `--master-data` [317].

- `--flush-privileges` [316]

Send a `FLUSH PRIVILEGES` statement to the server after dumping the `mysql` database. This option should be used any time the dump contains the `mysql` database and any other database that depends on the data in the `mysql` database for proper restoration.

- `--force` [316], `-f`

Continue even if an SQL error occurs during a table dump.

One use for this option is to cause `mysqldump` to continue executing even when it encounters a view that has become invalid because the definition refers to a table that has been dropped. Without `--force` [316], `mysqldump` exits with an error message. With `--force` [316], `mysqldump` prints the error message, but it also writes an SQL comment containing the view definition to the dump output and continues executing.

- `--host=host_name` [316], `-h host_name`

Dump data from the MySQL server on the given host. The default host is `localhost`.

- `--hex-blob` [316]

Dump binary columns using hexadecimal notation (for example, 'abc' becomes 0x616263). The affected data types are `BINARY`, `VARBINARY`, the `BLOB` types, and `BIT`.

- `--include-master-host-port` [317]

For the `CHANGE MASTER TO` statement in a slave dump produced with the `--dump-slave` [315] option, add `MASTER_PORT` and `MASTER_HOST` options for the host name and TCP/IP port number of the slave's master. This option was added in MySQL 5.5.3.

- `--ignore-table=db_name.tbl_name` [317]

Do not dump the given table, which must be specified using both the database and table names. To ignore multiple tables, use this option multiple times. This option also can be used to ignore views.

- `--insert-ignore` [317]

Write `INSERT IGNORE` statements rather than `INSERT` statements.

- `--lines-terminated-by=...` [317]

This option is used with the `--tab` [321] option and has the same meaning as the corresponding `LINES` clause for `LOAD DATA INFILE`. See [Section 13.2.6, “LOAD DATA INFILE Syntax”](#).

- `--lock-all-tables` [317], `-x`

Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off `--single-transaction` [321] and `--lock-tables` [317].

- `--lock-tables` [317], `-l`

For each dumped database, lock all tables to be dumped before dumping them. The tables are locked with `READ LOCAL` to permit concurrent inserts in the case of `MyISAM` tables. For transactional tables such as `InnoDB`, `--single-transaction` [321] is a much better option than `--lock-tables` [317] because it does not need to lock the tables at all.

Because `--lock-tables` [317] locks tables for each database separately, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

- `--log-error=file_name` [317]

Log warnings and errors by appending them to the named file. The default is to do no logging.

- `--master-data[=value]` [317]

Use this option to dump a master replication server to produce a dump file that can be used to set up another server as a slave of the master. It causes the dump output to include a `CHANGE MASTER TO` statement that indicates the binary log coordinates (file name and position) of the dumped server. These are the master server coordinates from which the slave should start replicating after you load the dump file into the slave.

If the option value is 2, the `CHANGE MASTER TO` statement is written as an SQL comment, and thus is informative only; it has no effect when the dump file is reloaded. If the option value is 1, the statement is not written as a comment and takes effect when the dump file is reloaded. If no option value is specified, the default value is 1.

This option requires the `RELOAD` [696] privilege and the binary log must be enabled.

The `--master-data` [317] option automatically turns off `--lock-tables` [317]. It also turns on `--lock-all-tables` [317], unless `--single-transaction` [321] also is specified, in which case, a global read lock is acquired only for a short time at the beginning of the dump (see the description for `--single-transaction` [321]). In all cases, any action on logs happens at the exact moment of the dump.

It is also possible to set up a slave by dumping an existing slave of the master. To do this, use the following procedure on the existing slave:

1. Stop the slave's SQL thread and get its current status:

```
mysql> STOP SLAVE SQL_THREAD;
mysql> SHOW SLAVE STATUS;
```

2. From the output of the `SHOW SLAVE STATUS` statement, the binary log coordinates of the master server from which the new slave should start replicating are the values of the `Relay_Master_Log_File` and `Exec_Master_Log_Pos` fields. Denote those values as `file_name` and `file_pos`.

3. Dump the slave server:

```
shell> mysqldump --master-data=2 --all-databases > dumpfile
```

Using `--master-data=2` works only if binary logging has been enabled on the slave. Otherwise, `mysqldump` fails with the error `Binlogging on server not active`. In this case you must handle any locking issues in another manner, using one or more of `--add-locks` [313], `--lock-tables` [317], `--lock-all-tables` [317], or `--single-transaction` [321], as required by your application and environment.

4. Restart the slave:

```
mysql> START SLAVE;
```

5. On the new slave, load the dump file:

```
shell> mysql < dumpfile
```

6. On the new slave, set the replication coordinates to those of the master server obtained earlier:

```
mysql> CHANGE MASTER TO
-> MASTER_LOG_FILE = 'file_name', MASTER_LOG_POS = file_pos;
```

The `CHANGE MASTER TO` statement might also need other parameters, such as `MASTER_HOST` to point the slave to the correct master server host. Add any such parameters as necessary.

- `--no-autocommit` [318]

Enclose the `INSERT` statements for each dumped table within `SET autocommit = 0` and `COMMIT` statements.

- `--no-create-db` [318], `-n`

This option suppresses the `CREATE DATABASE` statements that are otherwise included in the output if the `--databases` [314] or `--all-databases` [313] option is given.

- `--no-create-info` [318], `-t`

Do not write `CREATE TABLE` statements that re-create each dumped table.



Note

This option does *not* exclude statements creating log file groups or tablespaces from `mysqldump` output; however, you can use the `--no-tablespaces` [319] option for this purpose.

- `--no-data` [319], `-d`

Do not write any table row information (that is, do not dump table contents). This is useful if you want to dump only the `CREATE TABLE` statement for the table (for example, to create an empty copy of the table by loading the dump file).

- `--no-set-names` [319], `-N`

This has the same effect as `--skip-set-charset` [320].

- `--no-tablespaces` [319], `-y`

This option suppresses all `CREATE LOGFILE GROUP` and `CREATE TABLESPACE` statements in the output of `mysqldump`.

- `--opt` [319]

This option is shorthand. It is the same as specifying `--add-drop-table` [313] `--add-locks` [313] `--create-options` [314] `--disable-keys` [315] `--extended-insert` [316] `--lock-tables` [317] `--quick` [320] `--set-charset` [320]. It should give you a fast dump operation and produce a dump file that can be reloaded into a MySQL server quickly.

The `--opt` [319] option is enabled by default. Use `--skip-opt` [321] to disable it. See the discussion at the beginning of this section for information about selectively enabling or disabling a subset of the options affected by `--opt` [319].

- `--order-by-primary` [319]

Dump each table's rows sorted by its primary key, or by its first unique index, if such an index exists. This is useful when dumping a `MyISAM` table to be loaded into an `InnoDB` table, but will make the dump operation take considerably longer.

- `--password[=password]` [319], `-p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` [319] or `-p` option on the command line, `mysqldump` prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--pipe` [319], `-W`

On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path` [319]

The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [315] option is used to specify an authentication plugin but `mysqldump` does not find it. See Section 6.3.6, “Pluggable Authentication”.

This option was added in MySQL 5.5.9.

- `--port=port_num` [320], `-P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [320]

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--quick` [320], `-q`

This option is useful for dumping large tables. It forces `mysqldump` to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- `--quote-names` [320], `-Q`

Quote identifiers (such as database, table, and column names) within “`” characters. If the `ANSI_QUOTES` [612] SQL mode is enabled, identifiers are quoted within “” characters. This option is enabled by default. It can be disabled with `--skip-quote-names` [320], but this option should be given after any option such as `--compatible` [314] that may enable `--quote-names` [320].

- `--replace` [320]

Write `REPLACE` statements rather than `INSERT` statements.

- `--result-file=file_name` [320], `-r file_name`

Direct output to a given file. This option should be used on Windows to prevent newline “\n” characters from being converted to “\r\n” carriage return/newline sequences. The result file is created and its previous contents overwritten, even if an error occurs while generating the dump.

- `--routines` [320], `-R`

Include stored routines (procedures and functions) for the dumped databases in the output. Use of this option requires the `SELECT` [697] privilege for the `mysql.proc` table. The output generated by using `--routines` [320] contains `CREATE PROCEDURE` and `CREATE FUNCTION` statements to re-create the routines. However, these statements do not include attributes such as the routine creation and modification timestamps. This means that when the routines are reloaded, they will be created with the timestamps equal to the reload time.

If you require routines to be re-created with their original timestamp attributes, do not use `--routines` [320]. Instead, dump and reload the contents of the `mysql.proc` table directly, using a MySQL account that has appropriate privileges for the `mysql` database.

Prior to MySQL 5.5.21, this option had no effect when used together with the `--xml` [322] option. (Bug #11760384, Bug #52792)

- `--set-charset` [320]

Add `SET NAMES default_character_set` to the output. This option is enabled by default. To suppress the `SET NAMES` statement, use `--skip-set-charset` [320].

- `--single-transaction` [321]

This option sends a `START TRANSACTION` SQL statement to the server before dumping data. It is useful only with transactional tables such as `InnoDB`, because then it dumps the consistent state of the database at the time when `BEGIN` was issued without blocking any applications.

When using this option, you should keep in mind that only `InnoDB` tables are dumped in a consistent state. For example, any `MyISAM` or `MEMORY` tables dumped while using this option may still change state.

While a `--single-transaction` [321] dump is in process, to ensure a valid dump file (correct table contents and binary log coordinates), no other connection should use the following statements: `ALTER TABLE`, `CREATE TABLE`, `DROP TABLE`, `RENAME TABLE`, `TRUNCATE TABLE`. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the `SELECT` that is performed by `mysqldump` to retrieve the table contents to obtain incorrect contents or fail.

The `--single-transaction` [321] option and the `--lock-tables` [317] option are mutually exclusive because `LOCK TABLES` causes any pending transactions to be committed implicitly.

This option is not supported for MySQL Cluster tables; the results cannot be guaranteed to be consistent due to the fact that the `NDBCLUSTER` storage engine supports only the `READ_COMMITTED` transaction isolation level. You should always use `NDB` backup and restore instead.

To dump large tables, you should combine the `--single-transaction` [321] option with `--quick` [320].

- `--skip-comments` [321]

See the description for the `--comments` [314] option.

- `--skip-opt` [321]

See the description for the `--opt` [319] option.

- `--socket=path` [321], `-S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` [749] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See [Section 6.3.8.4, “SSL Command Options”](#).

- `--tab=path` [321], `-T path`

Produce tab-separated text-format data files. For each dumped table, `mysqldump` creates a `tbl_name.sql` file that contains the `CREATE TABLE` statement that creates the table, and the server writes a `tbl_name.txt` file that contains its data. The option value is the directory in which to write the files.

**Note**

This option should be used only when `mysqldump` is run on the same machine as the `mysqld` server. You must have the `FILE [696]` privilege, and the server must have permission to write files in the directory that you specify.

By default, the `.txt` data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the `--fields-xxx` and `--lines-terminated-by [317]` options.

Column values are converted to the character set specified by the `--default-character-set [315]` option.

- `--tables [322]`

Override the `--databases [314]` or `-B` option. `mysqldump` regards all name arguments following the option as table names.

- `--triggers [322]`

Include triggers for each dumped table in the output. This option is enabled by default; disable it with `--skip-triggers [322]`.

- `--tz-utc [322]`

This option enables `TIMESTAMP` columns to be dumped and reloaded between servers in different time zones. `mysqldump` sets its connection time zone to UTC and adds `SET TIME_ZONE='+00:00'` to the dump file. Without this option, `TIMESTAMP` columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change if the servers are in different time zones. `--tz-utc [322]` also protects against changes due to daylight saving time. `--tz-utc [322]` is enabled by default. To disable it, use `--skip-tz-utc [322]`.

- `--user=user_name [322]`, `-u user_name`

The MySQL user name to use when connecting to the server.

- `--verbose [322]`, `-v`

Verbose mode. Print more information about what the program does.

- `--version [322]`, `-V`

Display version information and exit.

- `--where='where_condition' [322]`, `-w 'where_condition'`

Dump only rows selected by the given `WHERE` condition. Quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter.

Examples:

```
--where="user='jimf' "  
-w"userid>1"  
-w"userid<1"
```

- `--xml [322]`, `-X`

Write dump output as well-formed XML.

NULL, 'NULL', and Empty Values: For a column named *column_name*, the NULL value, an empty string, and the string value 'NULL' are distinguished from one another in the output generated by this option as follows.

Value:	XML Representation:
NULL (<i>unknown value</i>)	<field name="column_name" xsi:nil="true" />
' ' (<i>empty string</i>)	<field name="column_name"></field>
'NULL' (<i>string value</i>)	<field name="column_name">NULL</field>

The output from the `mysql` client when run using the `--xml` [282] option also follows the preceding rules. (See Section 4.5.1.1, “`mysql` Options”).

XML output from `mysqldump` includes the XML namespace, as shown here:

```
shell> mysqldump --xml -u root world City
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="world">
<table_structure name="City">
<field Field="ID" Type="int(11)" Null="NO" Key="PRI" Extra="auto_increment" />
<field Field="Name" Type="char(35)" Null="NO" Key="" Default="" Extra="" />
<field Field="CountryCode" Type="char(3)" Null="NO" Key="" Default="" Extra="" />
<field Field="District" Type="char(20)" Null="NO" Key="" Default="" Extra="" />
<field Field="Population" Type="int(11)" Null="NO" Key="" Default="0" Extra="" />
<key Table="City" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="ID"
Collation="A" Cardinality="4079" Null="" Index_type="BTREE" Comment="" />
<options Name="City" Engine="MyISAM" Version="10" Row_format="Fixed" Rows="4079"
Avg_row_length="67" Data_length="273293" Max_data_length="18858823439613951"
Index_length="43008" Data_free="0" Auto_increment="4080"
Create_time="2007-03-31 01:47:01" Update_time="2007-03-31 01:47:02"
Collation="latin1_swedish_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="City">
<row>
<field name="ID">1</field>
<field name="Name">Kabul</field>
<field name="CountryCode">AFG</field>
<field name="District">Kabul</field>
<field name="Population">1780000</field>
</row>
...
<row>
<field name="ID">4079</field>
<field name="Name">Rafah</field>
<field name="CountryCode">PSE</field>
<field name="District">Rafah</field>
<field name="Population">92020</field>
</row>
</table_data>
</database>
</mysqldump>
```

Prior to MySQL 5.5.21, this option prevented the `--routines` [320] option from working correctly—that is, no stored routines, triggers, or events could be dumped in XML format. (Bug #11760384, Bug #52792)

You can also set the following variables by using `--var_name=value` syntax:

- `max_allowed_packet`

The maximum size of the buffer for client/server communication. The maximum is 1GB.

- `net_buffer_length` [531]

The initial size of the buffer for client/server communication. When creating multiple-row `INSERT` statements (as with the `--extended-insert` [316] or `--opt` [319] option), `mysqldump` creates rows up to `net_buffer_length` [531] length. If you increase this variable, you should also ensure that the `net_buffer_length` [531] variable in the MySQL server is at least this large.

A common use of `mysqldump` is for making a backup of an entire database:

```
shell> mysqldump db_name > backup-file.sql
```

You can load the dump file back into the server like this:

```
shell> mysql db_name < backup-file.sql
```

Or like this:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

`mysqldump` is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

It is possible to dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

To dump all databases, use the `--all-databases` [313] option:

```
shell> mysqldump --all-databases > all_databases.sql
```

For InnoDB tables, `mysqldump` provides a way of making an online backup:

```
shell> mysqldump --all-databases --single-transaction > all_databases.sql
```

This backup acquires a global read lock on all tables (using `FLUSH TABLES WITH READ LOCK`) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the `FLUSH` statement is issued, the MySQL server may get stalled until those statements finish. After that, the dump becomes lock free and does not disturb reads and writes on the tables. If the update statements that the MySQL server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as “roll-forward,” when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log (see [Section 5.2.4, “The Binary Log”](#)) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mysqldump --all-databases --flush-logs --master-data=2
> all_databases.sql
```

The `--master-data` [317] and `--single-transaction` [321] options can be used simultaneously, which provides a convenient way to make an online backup suitable for use prior to point-in-time recovery if tables are stored using the InnoDB storage engine.

For more information on making backups, see [Section 7.2, “Database Backup Methods”](#), and [Section 7.3, “Example Backup and Recovery Strategy”](#).

If you encounter problems backing up views, please read the section that covers restrictions on views which describes a workaround for backing up views when this fails due to insufficient privileges. See [Section E.5, “Restrictions on Views”](#).

4.5.5. mysqlimport — A Data Import Program

The `mysqlimport` client provides a command-line interface to the `LOAD DATA INFILE` SQL statement. Most options to `mysqlimport` correspond directly to clauses of `LOAD DATA INFILE` syntax. See [Section 13.2.6, “LOAD DATA INFILE Syntax”](#).

Invoke `mysqlimport` like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, `mysqlimport` strips any extension from the file name and uses the result to determine the name of the table into which to import the file's contents. For example, files named `patient.txt`, `patient.text`, and `patient` all would be imported into a table named `patient`.

For additional information about `mysqldump`, see [Section 7.4, “Using mysqldump for Backups”](#).

`mysqlimport` supports the following options, which can be specified on the command line or in the `[mysqlimport]` and `[client]` groups of an option file. `mysqlimport` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.6. mysqlimport Options

Format	Option File	Description	Introduced
<code>--columns=column_list</code> [327]	<code>columns</code> [327]	This option takes a comma-separated list of column names as its value	
<code>--compress</code> [327]	<code>compress</code> [327]	Compress all information sent between the client and the server	
<code>--debug[=debug_options]</code> [327]	<code>debug</code> [327]	Write a debugging log	
<code>--debug-check</code> [327]	<code>debug-check</code> [327]	Print debugging information when the program exits	
<code>--debug-info</code> [327]	<code>debug-info</code> [327]	Print debugging information, memory and CPU statistics when the program exits	
<code>--default-auth=plugin</code> [328]	<code>default-auth=plugin</code> [328]	The authentication plugin to use	5.5.10
<code>--default-character-set=charset_name</code> [327]	<code>default-character-set</code> [327]	Use <code>charset_name</code> as the default character set	
<code>--delete</code> [328]	<code>delete</code> [328]	Empty the table before importing the text file	
<code>--fields-enclosed-by=string</code> [328]	<code>fields-enclosed-by</code> [328]	This option has the same meaning as the corresponding clause for <code>LOAD DATA INFILE</code>	

Format	Option File	Description	Introduced
--fields-escaped-by [328]	fields-escaped-by [328]	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-optionally-enclosed-by=string [328]	fields-optionally-enclosed-by [328]	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-terminated-by=string [328]	fields-terminated-by [328]	-- This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--force [328]	force [328]	Continue even if an SQL error occurs	
--help [327]		Display help message and exit	
--host=host_name [328]	host [328]	Connect to the MySQL server on the given host	
--ignore [328]	ignore [328]	See the description for the --replace option	
--ignore-lines=# [328]	ignore-lines [328]	Ignore the first N lines of the data file	
--lines-terminated-by=string [328]	lines-terminated-by [328]	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--local [328]	local [328]	Read input files locally from the client host	
--lock-tables [328]	lock-tables [328]	Lock all tables for writing before processing any text files	
--low-priority [328]	low-priority [328]	Use LOW_PRIORITY when loading the table.	
--password[=password] [328]	password [328]	The password to use when connecting to the server	
--pipe [329]		On Windows, connect to server using a named pipe	
--plugin-dir=path [329]	plugin-dir=path [329]	The directory where plugins are located	5.5.10
--port=port_num [329]	port [329]	The TCP/IP port number to use for the connection	
--protocol=type [329]	protocol [329]	The connection protocol to use	
--replace [329]	replace [329]	The --replace and --ignore options control handling of input rows that duplicate existing rows on unique key values	
--silent [329]	silent [329]	Produce output only when errors occur	
--socket=path [329]	socket [329]	For connections to localhost	
--ssl-ca=file_name [329]	ssl-ca [329]	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name [329]	ssl-capath [329]	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name [329]	ssl-cert [329]	The name of the SSL certificate file to use for establishing a secure connection	

Format	Option File	Description	Introduced
<code>--ssl-cipher=cipher_list</code> [329]	<code>ssl-cipher</code> [329]	A list of allowable ciphers to use for SSL encryption	
<code>--ssl-key=file_name</code> [329]	<code>ssl-key</code> [329]	The name of the SSL key file to use for establishing a secure connection	
<code>--ssl-verify-server-cert</code> [329]	<code>ssl-verify-server-cert</code> [329]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
<code>--use-threads=#</code> [329]	<code>use-threads</code> [329]	The number of threads for parallel file-loading	
<code>--user=user_name,</code> [329]	<code>user</code> [329]	MySQL user name to use when connecting to server	
<code>--verbose</code> [330]		Verbose mode	
<code>--version</code> [330]		Display version information and exit	

- `--help` [327], `-?`

Display a help message and exit.

- `--bind-address=ip_address` [277]

On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

This option is supported only in the version of `mysqlimport` that is supplied with MySQL Cluster. It is not available in standard MySQL Server 5.5 releases.

- `--character-sets-dir=path` [327]

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--columns=column_list` [327], `-c column_list`

This option takes a comma-separated list of column names as its value. The order of the column names indicates how to match data file columns with table columns.

- `--compress` [327], `-C`

Compress all information sent between the client and the server if both support compression.

- `--debug[=debug_options]` [327], `-# [debug_options]`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o'`.

- `--debug-check` [327]

Print some debugging information when the program exits.

- `--debug-info` [327]

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-character-set=charset_name` [327]

Use `charset_name` as the default character set. See [Section 10.5, “Character Set Configuration”](#).

- `--default-auth=plugin` [328]

The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.10.

- `--delete` [328], `-D`

Empty the table before importing the text file.

- `--fields-terminated-by=...` [328], `--fields-enclosed-by=...` [328], `--fields-optionally-enclosed-by=...` [328], `--fields-escaped-by=...` [328]

These options have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See [Section 13.2.6, “LOAD DATA INFILE Syntax”](#).

- `--force` [328], `-f`

Ignore errors. For example, if a table for a text file does not exist, continue processing any remaining files. Without `--force` [328], `mysqlimport` exits if a table does not exist.

- `--host=host_name` [328], `-h host_name`

Import data to the MySQL server on the given host. The default host is `localhost`.

- `--ignore` [328], `-i`

See the description for the `--replace` [329] option.

- `--ignore-lines=N` [328]

Ignore the first *N* lines of the data file.

- `--lines-terminated-by=...` [328]

This option has the same meaning as the corresponding clause for `LOAD DATA INFILE`. For example, to import Windows files that have lines terminated with carriage return/linefeed pairs, use `--lines-terminated-by="\r\n"` [328]. (You might have to double the backslashes, depending on the escaping conventions of your command interpreter.) See [Section 13.2.6, “LOAD DATA INFILE Syntax”](#).

- `--local` [328], `-L`

Read input files locally from the client host.

- `--lock-tables` [317], `-l`

Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- `--low-priority` [328]

Use `LOW_PRIORITY` when loading the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- `--password[=password]` [328], `-p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` [328] or `-p` option on the command line, `mysqlimport` prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--pipe` [329], `-W`

On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path` [329]

The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [328] option is used to specify an authentication plugin but `mysqlimport` does not find it. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.10.

- `--port=port_num` [329], `-P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [329]

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see [Section 4.2.2, “Connecting to the MySQL Server”](#).

- `--replace` [329], `-r`

The `--replace` [329] and `--ignore` [328] options control handling of input rows that duplicate existing rows on unique key values. If you specify `--replace` [329], new rows replace existing rows that have the same unique key value. If you specify `--ignore` [328], input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- `--silent` [329], `-s`

Silent mode. Produce output only when errors occur.

- `--socket=path` [329], `-S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` [749] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See [Section 6.3.8.4, “SSL Command Options”](#).

- `--user=user_name` [329], `-u user_name`

The MySQL user name to use when connecting to the server.

- `--use-threads=N` [329]

Load files in parallel using N threads.

- `--verbose` [330], `-v`

Verbose mode. Print more information about what the program does.

- `--version` [330], `-V`

Display version information and exit.

Here is a sample session that demonstrates use of `mysqlimport`:

```
shell> mysql -e 'CREATE TABLE impptest(id INT, n VARCHAR(30))' test
shell> ed
a
100      Max Sydow
101      Count Dracula
.
w impptest.txt
32
q
shell> od -c impptest.txt
0000000  1  0  0  \t  M  a  x          S  y  d  o  w  \n  1  0
0000020  1  \t  C  o  u  n  t          D  r  a  c  u  l  a  \n
0000040
shell> mysqlimport --local test impptest.txt
test.impptest: Records: 2 Deleted: 0 Skipped: 0 Warnings: 0
shell> mysql -e 'SELECT * FROM impptest' test
+-----+-----+
| id  | n          |
+-----+-----+
| 100 | Max Sydow  |
| 101 | Count Dracula |
+-----+-----+
```

4.5.6. mysqlshow — Display Database, Table, and Column Information

The `mysqlshow` client can be used to quickly see which databases exist, their tables, or a table's columns or indexes.

`mysqlshow` provides a command-line interface to several SQL `SHOW` statements. See [Section 13.7.5, “SHOW Syntax”](#). The same information can be obtained by using those statements directly. For example, you can issue them from the `mysql` client program.

Invoke `mysqlshow` like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, a list of database names is shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters (“*”, “?”, “%”, or “_”), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. “*” and “?” characters are converted into SQL “%” and “_” wildcard characters. This might cause some

confusion when you try to display the columns for a table with a “_” in the name, because in this case, `mysqlshow` shows you only the table names that match the pattern. This is easily fixed by adding an extra “%” last on the command line as a separate argument.

`mysqlshow` supports the following options, which can be specified on the command line or in the `[mysqlshow]` and `[client]` groups of an option file. `mysqlshow` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.7. `mysqlshow` Options

Format	Option File	Description	Introduced
<code>--bind-address=ip_address [332]</code>	<code>bind-address [332]</code>	Use the specified network interface to connect to the MySQL Server	5.5.8
<code>--compress [332]</code>	<code>compress [332]</code>	Compress all information sent between the client and the server	
<code>--count [332]</code>	<code>count [332]</code>	Show the number of rows per table	
<code>--debug[=debug_options] [332]</code>	<code>debug [332]</code>	Write a debugging log	
<code>--debug-check [332]</code>	<code>debug-check [332]</code>	Print debugging information when the program exits	
<code>--debug-info [332]</code>	<code>debug-info [332]</code>	Print debugging information, memory and CPU statistics when the program exits	
<code>--default-auth=plugin [333]</code>	<code>default-auth=plugin [333]</code>	The authentication plugin to use	5.5.10
<code>--default-character-set=charset_name [333]</code>	<code>default-character-set [333]</code>	Use <code>charset_name</code> as the default character set	
<code>--help [332]</code>		Display help message and exit	
<code>--host=host_name [333]</code>	<code>host [333]</code>	Connect to the MySQL server on the given host	
<code>--keys [333]</code>	<code>keys [333]</code>	Show table indexes	
<code>--password[=password] [333]</code>	<code>password [333]</code>	The password to use when connecting to the server	
<code>--pipe [333]</code>		On Windows, connect to server using a named pipe	
<code>--plugin-dir=path [333]</code>	<code>plugin-dir=path [333]</code>	The directory where plugins are located	5.5.10
<code>--port=port_num [333]</code>	<code>port [333]</code>	The TCP/IP port number to use for the connection	
<code>--protocol=type [333]</code>	<code>protocol [333]</code>	The connection protocol to use	
<code>--show-table-type [333]</code>		Show a column indicating the table type	
<code>--socket=path [333]</code>	<code>socket [333]</code>	For connections to localhost	
<code>--ssl-ca=file_name [334]</code>	<code>ssl-ca [334]</code>	The path to a file that contains a list of trusted SSL CAs	

Format	Option File	Description	Introduced
<code>--ssl-capath=dir_name</code> [334]	<code>ssl-capath</code> [334]	The path to a directory that contains trusted SSL CA certificates in PEM format	
<code>--ssl-cert=file_name</code> [334]	<code>ssl-cert</code> [334]	The name of the SSL certificate file to use for establishing a secure connection	
<code>--ssl-cipher=cipher_list</code> [334]	<code>ssl-cipher</code> [334]	A list of allowable ciphers to use for SSL encryption	
<code>--ssl-key=file_name</code> [334]	<code>ssl-key</code> [334]	The name of the SSL key file to use for establishing a secure connection	
<code>--ssl-verify-server-cert</code> [334]	<code>ssl-verify-server-cert</code> [334]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
<code>--status</code> [334]	<code>status</code> [334]	Display extra information about each table	
<code>--user=user_name,</code> [334]	<code>user</code> [334]	MySQL user name to use when connecting to server	
<code>--verbose</code> [334]		Verbose mode	
<code>--version</code> [334]		Display version information and exit	

- `--help` [332], `-?`

Display a help message and exit.

- `--bind-address=ip_address` [277]

On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

This option is supported only in the version of `mysqlshow` that is supplied with MySQL Cluster. It is not available in standard MySQL Server 5.5 releases.

- `--character-sets-dir=path` [332]

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--compress` [332], `-C`

Compress all information sent between the client and the server if both support compression.

- `--count` [332]

Show the number of rows per table. This can be slow for non-MyISAM tables.

- `--debug [=debug_options]` [332], `-# [debug_options]`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o'`.

- `--debug-check` [332]

Print some debugging information when the program exits.

- `--debug-info` [332]

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-character-set=charset_name` [333]
Use *charset_name* as the default character set. See [Section 10.5, “Character Set Configuration”](#).
- `--default-auth=plugin` [333]
The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).
This option was added in MySQL 5.5.10.
- `--host=host_name` [333], `-h host_name`
Connect to the MySQL server on the given host.
- `--keys` [333], `-k`
Show table indexes.
- `--password[=password]` [333], `-p[password]`
The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` [333] or `-p` option on the command line, `mysqlshow` prompts for one.
Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.
- `--pipe` [333], `-W`
On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.
- `--plugin-dir=path` [333]
The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [333] option is used to specify an authentication plugin but `mysqlshow` does not find it. See [Section 6.3.6, “Pluggable Authentication”](#).
This option was added in MySQL 5.5.10.
- `--port=port_num` [333], `-P port_num`
The TCP/IP port number to use for the connection.
- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [333]
The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see [Section 4.2.2, “Connecting to the MySQL Server”](#).
- `--show-table-type` [333], `-t`
Show a column indicating the table type, as in `SHOW FULL TABLES`. The type is `BASE TABLE` or `VIEW`.
- `--socket=path` [333], `-S path`
For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` [749] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See [Section 6.3.8.4, “SSL Command Options”](#).

- `--status` [334], `-i`

Display extra information about each table.

- `--user=user_name` [334], `-u user_name`

The MySQL user name to use when connecting to the server.

- `--verbose` [334], `-v`

Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version` [334], `-V`

Display version information and exit.

4.5.7. mysqlslap — Load Emulation Client

`mysqlslap` is a diagnostic program designed to emulate client load for a MySQL server and to report the timing of each stage. It works as if multiple clients are accessing the server.

Invoke `mysqlslap` like this:

```
shell> mysqlslap [options]
```

Some options such as `--create` [338] or `--query` [341] enable you to specify a string containing an SQL statement or a file containing statements. If you specify a file, by default it must contain one statement per line. (That is, the implicit statement delimiter is the newline character.) Use the `--delimiter` [339] option to specify a different delimiter, which enables you to specify statements that span multiple lines or place multiple statements on a single line. You cannot include comments in a file; `mysqlslap` does not understand them.

`mysqlslap` runs in three stages:

1. Create schema, table, and optionally any stored programs or data to use for the test. This stage uses a single client connection.
2. Run the load test. This stage can use many client connections.
3. Clean up (disconnect, drop table if specified). This stage uses a single client connection.

Examples:

Supply your own create and query SQL statements, with 50 clients querying and 200 selects for each (enter the command on a single line):

```
mysqlslap --delimiter=";"
--create="CREATE TABLE a (b int);INSERT INTO a VALUES (23)"
--query="SELECT * FROM a" --concurrency=50 --iterations=200
```

Let `mysqlslap` build the query SQL statement with a table of two `INT` columns and three `VARCHAR` columns. Use five clients querying 20 times each. Do not create the table or insert the data (that is, use the previous test's schema and data):

```
mysqlslap --concurrency=5 --iterations=20
```

```
--number-int-cols=2 --number-char-cols=3
--auto-generate-sql
```

Tell the program to load the create, insert, and query SQL statements from the specified files, where the `create.sql` file has multiple table creation statements delimited by `' ; '` and multiple insert statements delimited by `' ; '`. The `--query` file will have multiple queries delimited by `' ; '`. Run all the load statements, then run all the queries in the query file with five clients (five times each):

```
mysqlslap --concurrency=5
--iterations=5 --query=query.sql --create=create.sql
--delimiter=";"
```

`mysqlslap` supports the following options, which can be specified on the command line or in the `[mysqlslap]` and `[client]` groups of an option file. `mysqlslap` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.8. `mysqlslap` Options

Format	Option File	Description	Introduced
<code>--auto-generate-sql [337]</code>	<code>auto-generate-sql [337]</code>	Generate SQL statements automatically when they are not supplied in files or using command options	
<code>--auto-generate-sql-add-autoincrement [337]</code>	<code>auto-generate-sql-add-autoincrement [337]</code>	Add <code>AUTO_INCREMENT</code> column to automatically generated tables	
<code>--auto-generate-sql-execute-number=# [337]</code>	<code>auto-generate-sql-execute-number [337]</code>	Specify how many queries to generate automatically	
<code>--auto-generate-sql-guid-primary [338]</code>	<code>auto-generate-sql-guid-primary [338]</code>	Add a GUID-based primary key to automatically generated tables	
<code>--auto-generate-sql-load-type=type [338]</code>	<code>auto-generate-sql-load-type [338]</code>	Specify how many queries to generate automatically	
<code>--auto-generate-sql-secondary-indexes=# [338]</code>	<code>auto-generate-sql-secondary-indexes [338]</code>	Specify how many secondary indexes to add to automatically generated tables	
<code>--auto-generate-sql-unique-query-number=# [338]</code>	<code>auto-generate-sql-unique-query-number [338]</code>	How many different queries to generate for automatic tests.	
<code>--auto-generate-sql-unique-write-number=# [338]</code>	<code>auto-generate-sql-unique-write-number [338]</code>	How many different queries to generate for <code>--auto-generate-sql-write-number</code>	
<code>--auto-generate-sql-write-number=# [338]</code>	<code>auto-generate-sql-write-number [338]</code>	How many row inserts to perform on each thread	
<code>--commit=# [338]</code>	<code>commit [338]</code>	How many statements to execute before committing.	
<code>--compress [338]</code>	<code>compress [338]</code>	Compress all information sent between the client and the server	
<code>--concurrency=# [338]</code>	<code>concurrency [338]</code>	The number of clients to simulate when issuing the <code>SELECT</code> statement	

Format	Option File	Description	Introduced
--create=value [338]	create [338]	The file or string containing the statement to use for creating the table	
--create-and-drop-schema=value [338]	create-and-drop-schema [338]	The schema in which to run the tests; dropped at the end of the test run	5.5.12
--create-schema=value [338]	create-schema [338]	The schema in which to run the tests	
--csv=[file] [339]	csv [339]	Generate output in comma-separated values format	
--debug[=debug_options] [339]	debug [339]	Write a debugging log	
--debug-check [339]	debug-check [339]	Print debugging information when the program exits	
--debug-info [339]	debug-info [339]	Print debugging information, memory and CPU statistics when the program exits	
--default-auth=plugin [339]	default-auth=plugin [339]	The authentication plugin to use	5.5.10
--delimiter=str [339]	delimiter [339]	The delimiter to use in SQL statements	
--detach=# [339]	detach [339]	Detach (close and reopen) each connection after each N statements	
--enable-cleartext-plugin [339]	enable-cleartext-plugin [339]	Enable cleartext authentication plugin	5.5.27
--engine=engine_name [339]	engine [339]	The storage engine to use for creating the table	
--help [337]		Display help message and exit	
--host=host_name [339]	host [339]	Connect to the MySQL server on the given host	
--iterations=# [339]	iterations [339]	The number of times to run the tests	
--number-char-cols=# [339]	number-char-cols [339]	The number of VARCHAR columns to use if --auto-generate-sql is specified	
--number-int-cols=# [339]	number-int-cols [339]	The number of INT columns to use if --auto-generate-sql is specified	
--number-of-queries=# [339]	number-of-queries [339]	Limit each client to approximately this number of queries	
--only-print [340]	only-print [340]	Do not connect to databases. mysqlslap only prints what it would have done	
--password[=password] [340]	password [340]	The password to use when connecting to the server	
--pipe [340]		On Windows, connect to server using a named pipe	
--plugin-dir=path [340]	plugin-dir=path [340]	The directory where plugins are located	5.5.10
--port=port_num [340]	port [340]	The TCP/IP port number to use for the connection	

Format	Option File	Description	Introduced
<code>--post-query=value</code> [340]	<code>post-query</code> [340]	The file or string containing the statement to execute after the tests have completed	
<code>--post-system=str</code> [340]	<code>post-system</code> [340]	The string to execute using <code>system()</code> after the tests have completed	
<code>--pre-query=value</code> [340]	<code>pre-query</code> [340]	The file or string containing the statement to execute before running the tests	
<code>--pre-system=str</code> [341]	<code>pre-system</code> [341]	The string to execute using <code>system()</code> before running the tests	
<code>--protocol=type</code> [341]	<code>protocol</code> [341]	The connection protocol to use	
<code>--query=value</code> [341]	<code>query</code> [341]	The file or string containing the SELECT statement to use for retrieving data	
<code>--silent</code> [341]	<code>silent</code> [341]	Silent mode	
<code>--socket=path</code> [341]	<code>socket</code> [341]	For connections to localhost	
<code>--ssl-ca=file_name</code> [341]	<code>ssl-ca</code> [341]	The path to a file that contains a list of trusted SSL CAs	
<code>--ssl-capath=dir_name</code> [341]	<code>ssl-capath</code> [341]	The path to a directory that contains trusted SSL CA certificates in PEM format	
<code>--ssl-cert=file_name</code> [341]	<code>ssl-cert</code> [341]	The name of the SSL certificate file to use for establishing a secure connection	
<code>--ssl-cipher=cipher_list</code> [341]	<code>ssl-cipher</code> [341]	A list of allowable ciphers to use for SSL encryption	
<code>--ssl-key=file_name</code> [341]	<code>ssl-key</code> [341]	The name of the SSL key file to use for establishing a secure connection	
<code>--ssl-verify-server-cert</code> [341]	<code>ssl-verify-server-cert</code> [341]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
<code>--user=user_name,</code> [341]	<code>user</code> [341]	MySQL user name to use when connecting to server	
<code>--verbose</code> [341]		Verbose mode	
<code>--version</code> [341]		Display version information and exit	

- `--help` [337], `-?`
Display a help message and exit.
- `--auto-generate-sql` [337], `-a`
Generate SQL statements automatically when they are not supplied in files or using command options.
- `--auto-generate-sql-add-autoincrement` [337]
Add an `AUTO_INCREMENT` column to automatically generated tables.
- `--auto-generate-sql-execute-number=N` [337]
Specify how many queries to generate automatically.

- `--auto-generate-sql-guid-primary` [338]
Add a GUID-based primary key to automatically generated tables.
- `--auto-generate-sql-load-type=type` [338]
Specify the test load type. The permissible values are `read` (scan tables), `write` (insert into tables), `key` (read primary keys), `update` (update primary keys), or `mixed` (half inserts, half scanning selects). The default is `mixed`.
- `--auto-generate-sql-secondary-indexes=N` [338]
Specify how many secondary indexes to add to automatically generated tables. By default, none are added.
- `--auto-generate-sql-unique-query-number=N` [338]
How many different queries to generate for automatic tests. For example, if you run a `key` test that performs 1000 selects, you can use this option with a value of 1000 to run 1000 unique queries, or with a value of 50 to perform 50 different selects. The default is 10.
- `--auto-generate-sql-unique-write-number=N` [338]
How many different queries to generate for `--auto-generate-sql-write-number` [338]. The default is 10.
- `--auto-generate-sql-write-number=N` [338]
How many row inserts to perform on each thread. The default is 100.
- `--commit=N` [338]
How many statements to execute before committing. The default is 0 (no commits are done).
- `--compress` [338], `-C`
Compress all information sent between the client and the server if both support compression.
- `--concurrency=N` [338], `-c N`
The number of clients to simulate when issuing the `SELECT` statement.
- `--create=value` [338]
The file or string containing the statement to use for creating the table.
- `--create-and-drop-schema=value` [338]
The schema in which to run the tests. `mysqlslap` drops the schema at the end of the test run. This option was added in MySQL 5.5.12.
- `--create-schema=value` [338]
The schema in which to run the tests.

**Note**

If the `--auto-generate-sql` [337] option is also given, `mysqlslap` drops the schema at the end of the test run. To avoid this, use the `--create-and-drop-schema` [338] option instead.

- `--csv[=file_name]` [339]

Generate output in comma-separated values format. The output goes to the named file, or to the standard output if no file is given.

- `--debug[=debug_options]` [339], `-# [debug_options]`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysqlslap.trace'`.

- `--debug-check` [339]

Print some debugging information when the program exits.

- `--debug-info` [339], `-T`

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin` [339]

The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.10.

- `--delimiter=str` [339], `-F str`

The delimiter to use in SQL statements supplied in files or using command options.

- `--detach=N` [339]

Detach (close and reopen) each connection after each `N` statements. The default is 0 (connections are not detached).

- `--enable-cleartext-plugin` [339]

Enable the `mysql_clear_password` cleartext authentication plugin. (See [Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”](#).) This option was added in MySQL 5.5.27.

- `--engine=engine_name` [339], `-e engine_name`

The storage engine to use for creating tables.

- `--host=host_name` [339], `-h host_name`

Connect to the MySQL server on the given host.

- `--iterations=N` [339], `-i N`

The number of times to run the tests.

- `--number-char-cols=N` [339], `-x N`

The number of `VARCHAR` columns to use if `--auto-generate-sql` [337] is specified.

- `--number-int-cols=N` [339], `-y N`

The number of `INT` columns to use if `--auto-generate-sql` [337] is specified.

- `--number-of-queries=N` [339]

Limit each client to approximately this many queries. Query counting takes into account the statement delimiter. For example, if you invoke `mysqlslap` as follows, the `;` delimiter is recognized so that each instance of the query string counts as two queries. As a result, 5 rows (not 10) are inserted.

```
shell> mysqlslap --delimiter=";" --number-of-queries=10
      --query="use test;insert into t values(null)"
```

- `--only-print` [340]

Do not connect to databases. `mysqlslap` only prints what it would have done.

- `--password[=password]` [340], `-p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` [340] or `-p` option on the command line, `mysqlslap` prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--pipe` [340], `-W`

On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path` [340]

The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [339] option is used to specify an authentication plugin but `mysqlslap` does not find it. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.10.

- `--port=port_num` [340], `-P port_num`

The TCP/IP port number to use for the connection.

- `--post-query=value` [340]

The file or string containing the statement to execute after the tests have completed. This execution is not counted for timing purposes.

- `--shared-memory-base-name=name` [340]

On Windows, the shared-memory name to use, for connections made using shared memory to a local server. This option applies only if the server supports shared-memory connections.

- `--post-system=str` [340]

The string to execute using `system()` after the tests have completed. This execution is not counted for timing purposes.

- `--pre-query=value` [340]

The file or string containing the statement to execute before running the tests. This execution is not counted for timing purposes.

- `--pre-system=str` [341]

The string to execute using `system()` before running the tests. This execution is not counted for timing purposes.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [341]

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see [Section 4.2.2, “Connecting to the MySQL Server”](#).

- `--query=value` [341], `-q value`

The file or string containing the `SELECT` statement to use for retrieving data.

- `--silent` [341], `-s`

Silent mode. No output.

- `--socket=path` [341], `-S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` [749] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See [Section 6.3.8.4, “SSL Command Options”](#).

- `--user=user_name` [341], `-u user_name`

The MySQL user name to use when connecting to the server.

- `--verbose` [341], `-v`

Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version` [341], `-V`

Display version information and exit.

4.6. MySQL Administrative and Utility Programs

This section describes administrative programs and programs that perform miscellaneous utility operations.

4.6.1. `innochecksum` — Offline InnoDB File Checksum Utility

`innochecksum` prints checksums for InnoDB files. This tool reads an InnoDB tablespace file, calculates the checksum for each page, compares the calculated checksum to the stored checksum, and reports mismatches, which indicate damaged pages. It was originally developed to speed up verifying the integrity of tablespace files after power outages but can also be used after file copies. Because checksum mismatches will cause InnoDB to deliberately shut down a running server, it can be preferable to use this tool rather than waiting for a server in production usage to encounter the damaged pages.

`innochecksum` cannot be used on tablespace files that the server already has open. For such files, you should use `CHECK TABLE` to check tables within the tablespace.

If checksum mismatches are found, you would normally restore the tablespace from backup or start the server and attempt to use `mysqldump` to make a backup of the tables within the tablespace.

Invoke `innochecksum` like this:

```
shell> innochecksum [options] file_name
```

`innochecksum` supports the following options. For options that refer to page numbers, the numbers are zero-based.

- `-c`
Print a count of the number of pages in the file.
- `-d`
Debug mode; prints checksums for each page.
- `-e num`
End at this page number.
- `-p num`
Check only this page number.
- `-s num`
Start at this page number.
- `-v`
Verbose mode; print a progress indicator every five seconds.

4.6.2. `myisam_ftdump` — Display Full-Text Index information

`myisam_ftdump` displays information about `FULLTEXT` indexes in `MyISAM` tables. It reads the `MyISAM` index file directly, so it must be run on the server host where the table is located. Before using `myisam_ftdump`, be sure to issue a `FLUSH TABLES` statement first if the server is running.

`myisam_ftdump` scans and dumps the entire index, which is not particularly fast. On the other hand, the distribution of words changes infrequently, so it need not be run often.

Invoke `myisam_ftdump` like this:

```
shell> myisam_ftdump [options] tbl_name index_num
```

The `tbl_name` argument should be the name of a `MyISAM` table. You can also specify a table by naming its index file (the file with the `.MYI` suffix). If you do not invoke `myisam_ftdump` in the directory where the table files are located, the table or index file name must be preceded by the path name to the table's database directory. Index numbers begin with 0.

Example: Suppose that the `test` database contains a table named `mytexttable1` that has the following definition:

```
CREATE TABLE mytexttable
(
  id   INT NOT NULL,
  txt  TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
```

```
) ENGINE=MyISAM;
```

The index on `id` is index 0 and the `FULLTEXT` index on `txt` is index 1. If your working directory is the `test` database directory, invoke `myisam_ftdump` as follows:

```
shell> myisam_ftdump mytexttable 1
```

If the path name to the `test` database directory is `/usr/local/mysql/data/test`, you can also specify the table name argument using that path name. This is useful if you do not invoke `myisam_ftdump` in the database directory:

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

You can use `myisam_ftdump` to generate a list of index entries in order of frequency of occurrence like this:

```
shell> myisam_ftdump -c mytexttable 1 | sort -r
```

`myisam_ftdump` supports the following options:

- `--help [343]`, `-h -?`
Display a help message and exit.
- `--count [343]`, `-c`
Calculate per-word statistics (counts and global weights).
- `--dump [343]`, `-d`
Dump the index, including data offsets and word weights.
- `--length [343]`, `-l`
Report the length distribution.
- `--stats [343]`, `-s`
Report global index statistics. This is the default operation if no other operation is specified.
- `--verbose [343]`, `-v`
Verbose mode. Print more output about what the program does.

4.6.3. myisamchk — MyISAM Table-Maintenance Utility

The `myisamchk` utility gets information about your database tables or checks, repairs, or optimizes them. `myisamchk` works with `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See [Section 13.7.2.2, “CHECK TABLE Syntax”](#), and [Section 13.7.2.5, “REPAIR TABLE Syntax”](#).

The use of `myisamchk` with partitioned tables is not supported.



Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

Invoke `myisamchk` like this:

```
shell> myisamchk [options] tbl_name ...
```

The *options* specify what you want `myisamchk` to do. They are described in the following sections. You can also get a list of options by invoking `myisamchk --help`.

With no options, `myisamchk` simply checks your table as the default operation. To get more information or to tell `myisamchk` to take corrective action, specify options as described in the following discussion.

tbl_name is the database table you want to check or repair. If you run `myisamchk` somewhere other than in the database directory, you must specify the path to the database directory, because `myisamchk` has no idea where the database is located. In fact, `myisamchk` does not actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the `myisamchk` command line if you wish. You can also specify a table by naming its index file (the file with the `.MYI` suffix). This enables you to specify all tables in a directory by using the pattern `*.MYI`. For example, if you are in a database directory, you can check all the MyISAM tables in that directory like this:

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all MyISAM tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

If you want to check all MyISAM tables and repair any that are corrupted, you can use the following command:

```
shell> myisamchk --silent --force --fast --update-state \
  --key_buffer_size=64M --myisam_sort_buffer_size=64M \
  --read_buffer_size=1M --write_buffer_size=1M \
  /path/to/datadir/*/*.MYI
```

This command assumes that you have more than 64MB free. For more information about memory allocation with `myisamchk`, see [Section 4.6.3.6, “myisamchk Memory Usage”](#).

For additional information about using `myisamchk`, see [Section 7.6, “MyISAM Table Maintenance and Crash Recovery”](#).



Important

You must ensure that no other program is using the tables while you are running `myisamchk`. The most effective means of doing so is to shut down the MySQL server while running `myisamchk`, or to lock all tables that `myisamchk` is being used on.

Otherwise, when you run `myisamchk`, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This means that you are trying to check a table that has been updated by another program (such as the `mysqld` server) that hasn't yet closed the file or that has died without closing the file properly, which can sometimes lead to the corruption of one or more MyISAM tables.

If `mysqld` is running, you must force it to flush any table modifications that are still buffered in memory by using `FLUSH TABLES`. You should then ensure that no one is using the tables while you are running `myisamchk`.

However, the easiest way to avoid this problem is to use `CHECK TABLE` instead of `myisamchk` to check tables. See [Section 13.7.2.2, “CHECK TABLE Syntax”](#).

`myisamchk` supports the following options, which can be specified on the command line or in the `[myisamchk]` group of an option file. `myisamchk` also supports the options for processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.9. `myisamchk` Options

Format	Option File	Description	Introduced
<code>--analyze</code> [352]	<code>analyze</code> [352]	Analyze the distribution of key values	
<code>--backup</code> [350]	<code>backup</code> [350]	Make a backup of the .MYD file as file_name-time.BAK	
<code>--block-search=offset</code> [352]	<code>block-search</code> [352]	Find the record that a block at the given offset belongs to	
<code>--check</code> [349]	<code>check</code> [349]	Check the table for errors	
<code>--check-only-changed</code> [349]	<code>check-only-changed</code> [349]	Check only tables that have changed since the last check	
<code>--correct-checksum</code> [350]	<code>correct-checksum</code> [350]	Correct the checksum information for the table	
<code>--data-file-length=len</code> [350]	<code>data-file-length</code> [350]	Maximum length of the data file (when re-creating data file when it is full)	
<code>--debug[=debug_options]</code> [347]	<code>debug</code> [347]	Write a debugging log	
<code>decode_bits=#</code>	<code>decode_bits</code>	<code>Decode_bits</code>	
<code>--description</code> [352]	<code>description</code> [352]	Print some descriptive information about the table	
<code>--extend-check</code> [349]	<code>extend-check</code> [349]	Do very thorough table check or repair that tries to recover every possible row from the data file	
<code>--fast</code> [349]	<code>fast</code> [349]	Check only tables that haven't been closed properly	
<code>--force</code> [349]	<code>force</code> [349]	Do a repair operation automatically if <code>myisamchk</code> finds any errors in the table	
<code>--force</code>	<code>force-recover</code>	Overwrite old temporary files. For use with the <code>-r</code> or <code>-o</code> option	
<code>ft_max_word_len=#</code>	<code>ft_max_word_len</code>	Maximum word length for FULLTEXT indexes	
<code>ft_min_word_len=#</code>	<code>ft_min_word_len</code>	Minimum word length for FULLTEXT indexes	
<code>ft_stopword_file=val</code>	<code>ft_stopword_file</code>	Use stopwords from this file instead of built-in list	
<code>--HELP</code> [347]		Display help message and exit	

Format	Option File	Description	Introduced
<code>--help</code> [347]		Display help message and exit	
<code>--information</code> [349]	<code>information</code> [349]	Print informational statistics about the table that is checked	
<code>key_buffer_size=#</code>	<code>key_buffer_size</code>	The size of the buffer used for index blocks for MyISAM tables	
<code>--keys-used=val</code> [350]	<code>keys-used</code> [350]	A bit-value that indicates which indexes to update	
<code>--max-record-length=len</code> [351]	<code>max-record-length</code> [351]	Skip rows larger than the given length if myisamchk cannot allocate memory to hold them	
<code>--medium-check</code> [350]	<code>medium-check</code> [350]	Do a check that is faster than an <code>--extend-check</code> operation	
<code>myisam_block_size=#</code>	<code>myisam_block_size</code>	Block size to be used for MyISAM index pages	
<code>myisam_sort_buffer_size=#</code>	<code>myisam_sort_buffer_size</code>	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE	5.5.29
<code>--parallel-recover</code> [351]	<code>parallel-recover</code> [351]	Uses the same technique as <code>-r</code> and <code>-n</code> , but creates all the keys in parallel, using different threads (beta)	
<code>--quick</code> [351]	<code>quick</code> [351]	Achieve a faster repair by not modifying the data file.	
<code>read_buffer_size=#</code>	<code>read_buffer_size</code>	Each thread that does a sequential scan allocates a buffer of this size for each table it scans	
<code>--read-only</code> [350]	<code>read-only</code> [350]	Don't mark the table as checked	
<code>--recover</code> [351]	<code>recover</code> [351]	Do a repair that can fix almost any problem except unique keys that aren't unique	
<code>--safe-recover</code> [351]	<code>safe-recover</code> [351]	Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found	
<code>--set-auto-increment[=value]</code> [352]	<code>set-auto-increment</code> [352]	Force AUTO_INCREMENT numbering for new records to start at the given value	
<code>--set-collation=name</code> [351]	<code>set-collation</code> [351]	Specify the collation to use for sorting table indexes	
<code>--silent</code> [347]	<code>silent</code> [347]	Silent mode	
<code>sort_buffer_size=#</code>	<code>sort_buffer_size</code>	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE	
<code>--sort-index</code> [352]	<code>sort-index</code> [352]	Sort the index tree blocks in high-low order	
<code>sort_key_blocks=#</code>	<code>sort_key_blocks</code>	<code>sort_key_blocks</code>	
<code>--sort-records=#</code> [352]	<code>sort-records</code> [352]	Sort records according to a particular index	
<code>--sort-recover</code> [351]	<code>sort-recover</code> [351]	Force myisamchk to use sorting to resolve the keys even if the temporary files would be very large	
<code>stats_method=value</code>	<code>stats_method</code>	Specifies how MyISAM index statistics collection code should treat NULLs	

Format	Option File	Description	Introduced
<code>--tmpdir=path [351]</code>	<code>tmpdir [351]</code>	Path of the directory to be used for storing temporary files	
<code>--unpack [352]</code>	<code>unpack [352]</code>	Unpack a table that was packed with <code>myisampack</code>	
<code>--update-state [350]</code>	<code>update-state [350]</code>	Store information in the <code>.MYI</code> file to indicate when the table was checked and whether the table crashed	
<code>--verbose [347]</code>		Verbose mode	
<code>--version [347]</code>		Display version information and exit	
<code>write_buffer_size=#</code>	<code>write_buffer_size</code>	Write buffer size	

4.6.3.1. myisamchk General Options

The options described in this section can be used for any type of table maintenance operation performed by `myisamchk`. The sections following this one describe options that pertain only to specific operations, such as table checking or repairing.

- `--help [347]`, `-?`
Display a help message and exit. Options are grouped by type of operation.
- `--HELP [347]`, `-H`
Display a help message and exit. Options are presented in a single list.
- `--debug=debug_options [347]`, `-# debug_options`
Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/myisamchk.trace'`.
- `--silent [347]`, `-s`
Silent mode. Write output only when errors occur. You can use `-s` twice (`-ss`) to make `myisamchk` very silent.
- `--verbose [347]`, `-v`
Verbose mode. Print more information about what the program does. This can be used with `-d` and `-e`. Use `-v` multiple times (`-vv`, `-vvv`) for even more output.
- `--version [347]`, `-V`
Display version information and exit.
- `--wait [347]`, `-w`
Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. If you are running `mysqld` with external locking disabled, the table can be locked only by another `myisamchk` command.

You can also set the following variables by using `--var_name=value` syntax:

Variable	Default Value
<code>decode_bits</code>	9

Variable	Default Value
<code>ft_max_word_len</code>	version-dependent
<code>ft_min_word_len</code>	4
<code>ft_stopword_file</code>	built-in list
<code>key_buffer_size</code>	523264
<code>myisam_block_size</code>	1024
<code>myisam_sort_key_blocks</code>	16
<code>read_buffer_size</code>	262136
<code>sort_buffer_size</code>	2097144
<code>sort_key_blocks</code>	16
<code>stats_method</code>	nulls_unequal
<code>write_buffer_size</code>	262136

The possible `myisamchk` variables and their default values can be examined with `myisamchk --help`:

`sort_buffer_size` is used when the keys are repaired by sorting keys, which is the normal case when you use `--recover` [351]. As of MySQL 5.5.29, `myisam_sort_buffer_size` is available as an alternative name to `sort_buffer_size`. `myisam_sort_buffer_size` is preferable to `sort_buffer_size` because its name corresponds to the `myisam_sort_buffer_size` [530] server system variable that has a similar meaning. `sort_buffer_size` should be considered deprecated.

`key_buffer_size` is used when you are checking the table with `--extend-check` [349] or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use `--safe-recover` [351].
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for `CHAR`, `VARCHAR`, or `TEXT` columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force `myisamchk` to repair by sorting, you can use the `--sort-recover` [351] option.

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the `key_buffer_size` and `myisam_sort_buffer_size` variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

`myisam_block_size` is the size used for index blocks.

`stats_method` influences how `NULL` values are treated for index statistics collection when the `--analyze` [352] option is given. It acts like the `myisam_stats_method` system variable. For more information, see the description of `myisam_stats_method` in [Section 5.1.4, “Server System Variables”](#), and [Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”](#).

`ft_min_word_len` and `ft_max_word_len` indicate the minimum and maximum word length for `FULLTEXT` indexes. `ft_stopword_file` names the stopword file. These need to be set under the following circumstances.

If you use `myisamchk` to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values to `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, you can place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE`. These statements are performed by the server, which knows the proper full-text parameter values to use.

4.6.3.2. myisamchk Check Options

`myisamchk` supports the following options for table checking operations:

- `--check` [349], `-c`

Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- `--check-only-changed` [349], `-C`

Check only tables that have changed since the last check.

- `--extend-check` [349], `-e`

Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, `myisamchk` or `myisamchk --medium-check` should be able to determine whether there are any errors in the table.

If you are using `--extend-check` [349] and have plenty of memory, setting the `key_buffer_size` variable to a large value helps the repair operation run faster.

See also the description of this option under table repair options.

For a description of the output format, see [Section 4.6.3.5, “Obtaining Table Information with myisamchk”](#).

- `--fast` [349], `-F`

Check only tables that haven't been closed properly.

- `--force` [349], `-f`

Do a repair operation automatically if `myisamchk` finds any errors in the table. The repair type is the same as that specified with the `--recover` [351] or `-r` option.

- `--information` [349], `-i`

Print informational statistics about the table that is checked.

- `--medium-check [350], -m`

Do a check that is faster than an `--extend-check [349]` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--read-only [350], -T`

Do not mark the table as checked. This is useful if you use `myisamchk` to check a table that is in use by some other application that does not use locking, such as `mysqld` when run with external locking disabled.

- `--update-state [350], -U`

Store information in the `.MYI` file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the `--check-only-changed [349]` option, but you shouldn't use this option if the `mysqld` server is using the table and you are running it with external locking disabled.

4.6.3.3. `myisamchk` Repair Options

`myisamchk` supports the following options for table repair operations (operations performed when an option such as `--recover [351]` or `--safe-recover [351]` is given):

- `--backup [350], -B`

Make a backup of the `.MYD` file as `file_name-time.BAK`

- `--character-sets-dir=path [350]`

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--correct-checksum [350]`

Correct the checksum information for the table.

- `--data-file-length=len [350], -D len`

The maximum length of the data file (when re-creating data file when it is “full”).

- `--extend-check [349], -e`

Do a repair that tries to recover every possible row from the data file. Normally, this also finds a lot of garbage rows. Do not use this option unless you are desperate.

See also the description of this option under table checking options.

For a description of the output format, see [Section 4.6.3.5, “Obtaining Table Information with `myisamchk`”](#).

- `--force [349], -f`

Overwrite old intermediate files (files with names like `tbl_name.TMD`) instead of aborting.

- `--keys-used=val [350], -k val`

For `myisamchk`, the option value is a bit-value that indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. An option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using `myisamchk -r`.

- `--no-symlinks` [351], `-l`

Do not follow symbolic links. Normally `myisamchk` repairs the table that a symlink points to. This option does not exist as of MySQL 4.0 because versions from 4.0 on do not remove symlinks during repair operations.

- `--max-record-length=len` [351]

Skip rows larger than the given length if `myisamchk` cannot allocate memory to hold them.

- `--parallel-recover` [351], `-p`

Use the same technique as `-r` and `-n`, but create all the keys in parallel, using different threads. *This is beta-quality code. Use at your own risk!*

- `--quick` [351], `-q`

Achieve a faster repair by modifying only the index file, not the data file. You can specify this option twice to force `myisamchk` to modify the original data file in case of duplicate keys.

- `--recover` [351], `-r`

Do a repair that can fix almost any problem except unique keys that are not unique (which is an extremely unlikely error with MyISAM tables). If you want to recover a table, this is the option to try first. You should try `--safe-recover` [351] only if `myisamchk` reports that the table cannot be recovered using `--recover` [351]. (In the unlikely case that `--recover` [351] fails, the data file remains intact.)

If you have lots of memory, you should increase the value of `myisam_sort_buffer_size`.

- `--safe-recover` [351], `-o`

Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than `--recover` [351], but can handle a couple of very unlikely cases that `--recover` [351] cannot. This recovery method also uses much less disk space than `--recover` [351]. Normally, you should repair first using `--recover` [351], and then with `--safe-recover` [351] only if `--recover` [351] fails.

If you have lots of memory, you should increase the value of `key_buffer_size`.

- `--set-character-set=name` [351]

Change the character set used by the table indexes. This option was replaced by `--set-collation` [351] in MySQL 5.0.3.

- `--set-collation=name` [351]

Specify the collation to use for sorting table indexes. The character set name is implied by the first part of the collation name.

- `--sort-recover` [351], `-n`

Force `myisamchk` to use sorting to resolve the keys even if the temporary files would be very large.

- `--tmpdir=path` [351], `-t path`

The path of the directory to be used for storing temporary files. If this is not set, `myisamchk` uses the value of the `TMPDIR` environment variable. `--tmpdir` [351] can be set to a list of directory paths

that are used successively in round-robin fashion for creating temporary files. The separator character between directory names is the colon (":") on Unix and the semicolon (";") on Windows.

- `--unpack [352], -u`

Unpack a table that was packed with `myisampack`.

4.6.3.4. Other `myisamchk` Options

`myisamchk` supports the following options for actions other than table checks and repairs:

- `--analyze [352], -a`

Analyze the distribution of key values. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use. To obtain information about the key distribution, use a `myisamchk --description --verbose tbl_name` command or the `SHOW INDEX FROM tbl_name` statement.

- `--block-search=offset [352], -b offset`

Find the record that a block at the given offset belongs to.

- `--description [352], -d`

Print some descriptive information about the table. Specifying the `--verbose [347]` option once or twice produces additional information. See [Section 4.6.3.5, “Obtaining Table Information with `myisamchk`”](#).

- `--set-auto-increment [=value] [352], -A[value]`

Force `AUTO_INCREMENT` numbering for new records to start at the given value (or higher, if there are existing records with `AUTO_INCREMENT` values this large). If `value` is not specified, `AUTO_INCREMENT` numbers for new records begin with the largest value currently in the table, plus one.

- `--sort-index [352], -S`

Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.

- `--sort-records=N [352], -R N`

Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW INDEX`, which displays a table's indexes in the same order that `myisamchk` sees them. Indexes are numbered beginning with 1.

If keys are not packed (`PACK_KEYS=0`), they have the same length, so when `myisamchk` sorts and moves records, it just overwrites record offsets in the index. If keys are packed (`PACK_KEYS=1`), `myisamchk` must unpack key blocks first, then re-create indexes and pack the key blocks again. (In this case, re-creating indexes is faster than updating offsets for each index.)

4.6.3.5. Obtaining Table Information with `myisamchk`

To obtain a description of a `MyISAM` table or statistics about it, use the commands shown here. The output from these commands is explained later in this section.

- `myisamchk -d tbl_name`

Runs `myisamchk` in “describe mode” to produce a description of your table. If you start the MySQL server with external locking disabled, `myisamchk` may report an error for a table that is updated while it runs. However, because `myisamchk` does not change the table in describe mode, there is no risk of destroying data.

- `myisamchk -dv tbl_name`

Adding `-v` runs `myisamchk` in verbose mode so that it produces more information about the table. Adding `-v` a second time produces even more information.

- `myisamchk -eis tbl_name`

Shows only the most important information from a table. This operation is slow because it must read the entire table.

- `myisamchk -eiv tbl_name`

This is like `-eis`, but tells you what is being done.

The `tbl_name` argument can be either the name of a MyISAM table or the name of its index file, as described in [Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”](#). Multiple `tbl_name` arguments can be given.

Suppose that a table named `person` has the following structure. (The `MAX_ROWS` table option is included so that in the example output from `myisamchk` shown later, some values are smaller and fit the output format more easily.)

```
CREATE TABLE person
(
  id          INT NOT NULL AUTO_INCREMENT,
  last_name   VARCHAR(20) NOT NULL,
  first_name  VARCHAR(20) NOT NULL,
  birth       DATE,
  death       DATE,
  PRIMARY KEY (id),
  INDEX (last_name, first_name),
  INDEX (birth)
) MAX_ROWS = 1000000;
```

Suppose also that the table has these data and index file sizes:

```
-rw-rw---- 1 mysql mysql 9347072 Aug 19 11:47 person.MYD
-rw-rw---- 1 mysql mysql 6066176 Aug 19 11:47 person.MYI
```

Example of `myisamchk -dvv` output:

```
MyISAM file:      person
Record format:    Packed
Character set:    latin1_swedish_ci (8)
File-version:     1
Creation time:    2009-08-19 16:47:41
Recover time:     2009-08-19 16:47:56
Status:           checked,analyzed,optimized keys
Auto increment key: 1  Last value:      306688
Data records:     306688  Deleted blocks: 0
Datafile parts:   306688  Deleted data:   0
Datafile pointer (bytes): 4  Keyfile pointer (bytes): 3
Datafile length: 9347072  Keyfile length: 6066176
Max datafile length: 4294967294  Max keyfile length: 17179868159
Recordlength:    54
```

```

table description:
Key Start Len Index Type Rec/key Root Blocksize
1 2 4 unique long 1 99328 1024
2 6 20 multip. varchar prefix 512 3563520 1024
 27 20 varchar 512
3 48 3 multip. uint24 NULL 306688 6065152 1024

Field Start Length Nullpos Nullbit Type
1 1 1
2 2 4 no zeros
3 6 21 varchar
4 27 21 varchar
5 48 3 1 1 no zeros
6 51 3 1 2 no zeros
    
```

Explanations for the types of information `myisamchk` produces are given here. “Keyfile” refers to the index file. “Record” and “row” are synonymous, as are “field” and “column.”

The initial part of the table description contains these values:

- `MyISAM file`

Name of the `MyISAM` (index) file.

- `Record format`

The format used to store table rows. The preceding examples use `Fixed length`. Other possible values are `Compressed` and `Packed`. (`Packed` corresponds to what `SHOW TABLE STATUS` reports as `Dynamic`.)

- `Character set`

The table default character set.

- `File-version`

Version of `MyISAM` format. Currently always 1.

- `Creation time`

When the data file was created.

- `Recover time`

When the index/data file was last reconstructed.

- `Status`

Table status flags. Possible values are `crashed`, `open`, `changed`, `analyzed`, `optimized keys`, and `sorted index pages`.

- `Auto increment key, Last value`

The key number associated the table's `AUTO_INCREMENT` column, and the most recently generated value for this column. These fields do not appear if there is no such column.

- `Data records`

The number of rows in the table.

- `Deleted blocks`

How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See [Section 7.6.4, “MyISAM Table Optimization”](#).

- [Datafile parts](#)

For dynamic-row format, this indicates how many data blocks there are. For an optimized table without fragmented rows, this is the same as [Data records](#).

- [Deleted data](#)

How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See [Section 7.6.4, “MyISAM Table Optimization”](#).

- [Datafile pointer](#)

The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a row address. For dynamic tables, this is a byte address.

- [Keyfile pointer](#)

The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- [Max datafile length](#)

How long the table data file can become, in bytes.

- [Max keyfile length](#)

How long the table index file can become, in bytes.

- [Recordlength](#)

How much space each row takes, in bytes.

The [table description](#) part of the output includes a list of all keys in the table. For each key, [myisamchk](#) displays some low-level information:

- [Key](#)

This key's number. This value is shown only for the first column of the key. If this value is missing, the line corresponds to the second or later column of a multiple-column key. For the table shown in the example, there are two [table description](#) lines for the second index. This indicates that it is a multiple-part index with two parts.

- [Start](#)

Where in the row this portion of the index starts.

- [Len](#)

How long this portion of the index is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column. The total length of a multiple-part key is the sum of the [Len](#) values for all key parts.

- [Index](#)

Whether a key value can exist multiple times in the index. Possible values are `unique` or `multip.` (multiple).

- `Type`

What data type this portion of the index has. This is a `MyISAM` data type with the possible values `packed`, `stripped`, or `empty`.

- `Root`

Address of the root index block.

- `Blocksize`

The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

- `Rec/key`

This is a statistical value used by the optimizer. It tells how many rows there are per value for this index. A unique index always has a value of 1. This may be updated after a table is loaded (or greatly changed) with `myisamchk -a`. If this is not updated at all, a default value of 30 is given.

The last part of the output provides information about each column:

- `Field`

The column number.

- `Start`

The byte position of the column within table rows.

- `Length`

The length of the column in bytes.

- `Nullpos, Nullbit`

For columns that can be `NULL`, `MyISAM` stores `NULL` values as a flag in a byte. Depending on how many nullable columns there are, there can be one or more bytes used for this purpose. The `Nullpos` and `Nullbit` values, if nonempty, indicate which byte and bit contains that flag indicating whether the column is `NULL`.

The position and number of bytes used to store `NULL` flags is shown in the line for field 1. This is why there are six `Field` lines for the `person` table even though it has only five columns.

- `Type`

The data type. The value may contain any of the following descriptors:

- `constant`

All rows have the same value.

- `no endspace`

Do not store endspace.

- `no endspace, not_always`

Do not store endspace and do not do endspace compression for all values.

- `no endspace, no empty`

Do not store endspace. Do not store empty values.

- `table-lookup`

The column was converted to an `ENUM`.

- `zerofill(N)`

The most significant `N` bytes in the value are always 0 and are not stored.

- `no zeros`

Do not store zeros.

- `always zero`

Zero values are stored using one bit.

- `Huff tree`

The number of the Huffman tree associated with the column.

- `Bits`

The number of bits used in the Huffman tree.

The `Huff tree` and `Bits` fields are displayed if the table has been compressed with `myisampack`. See [Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#), for an example of this information.

Example of `myisamchk -eiv` output:

```
Checking MyISAM file: person
Data records: 306688 Deleted blocks: 0
- check file-size
- check record delete-chain
No recordlinks
- check key delete-chain
block_size 1024:
- check index reference
- check data record references index: 1
Key: 1: Keyblocks used: 98% Packed: 0% Max levels: 3
- check data record references index: 2
Key: 2: Keyblocks used: 99% Packed: 97% Max levels: 3
- check data record references index: 3
Key: 3: Keyblocks used: 98% Packed: -14% Max levels: 3
Total: Keyblocks used: 98% Packed: 89%

- check records and index references*** LOTS OF ROW NUMBERS DELETED ***

Records:          306688 M.recordlength:      25 Packed:          83%
Recordspace used: 97% Empty space:          2% Blocks/Record: 1.00
Record blocks:    306688 Delete blocks:      0
Record data:      7934464 Deleted data:      0
Lost space:       256512 Linkdata:          1156096
```

```
User time 43.08, System time 1.68
Maximum resident set size 0, Integral resident set size 0
Non-physical pagefaults 0, Physical pagefaults 0, Swaps 0
Blocks in 0 out 7, Messages in 0 out 0, Signals 0
Voluntary context switches 0, Involuntary context switches 0
Maximum memory usage: 1046926 bytes (1023k)
```

`myisamchk -eiv` output includes the following information:

- `Data records`

The number of rows in the table.

- `Deleted blocks`

How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See [Section 7.6.4, “MyISAM Table Optimization”](#).

- `Key`

The key number.

- `Keyblocks used`

What percentage of the keyblocks are used. When a table has just been reorganized with `myisamchk`, the values are very high (very near theoretical maximum).

- `Packed`

MySQL tries to pack key values that have a common suffix. This can only be used for indexes on `CHAR` and `VARCHAR` columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the preceding example, the second key is 40 bytes long and a 97% reduction in space is achieved.

- `Max levels`

How deep the B-tree for this key is. Large tables with long key values get high values.

- `Records`

How many rows are in the table.

- `M.recordlength`

The average row length. This is the exact row length for tables with fixed-length rows, because all rows have the same length.

- `Packed`

MySQL strips spaces from the end of strings. The `Packed` value indicates the percentage of savings achieved by doing this.

- `Recordspace used`

What percentage of the data file is used.

- `Empty space`

What percentage of the data file is unused.

- [Blocks/Record](#)

Average number of blocks per row (that is, how many links a fragmented row is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too large, you can reorganize the table. See [Section 7.6.4, “MyISAM Table Optimization”](#).

- [Recordblocks](#)

How many blocks (links) are used. For fixed-format tables, this is the same as the number of rows.

- [Deleteblocks](#)

How many blocks (links) are deleted.

- [Recorddata](#)

How many bytes in the data file are used.

- [Deleted data](#)

How many bytes in the data file are deleted (unused).

- [Lost space](#)

If a row is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- [Linkdata](#)

When the dynamic table format is used, row fragments are linked with pointers (4 to 7 bytes each). [Linkdata](#) is the sum of the amount of storage used by all such pointers.

4.6.3.6. myisamchk Memory Usage

Memory allocation is important when you run [myisamchk](#). [myisamchk](#) uses no more memory than its memory-related variables are set to. If you are going to use [myisamchk](#) on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get [myisamchk](#) to operate faster. For example, if you have more than 512MB RAM available, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk --myisam_sort_buffer_size=256M \
  --key_buffer_size=512M \
  --read_buffer_size=64M \
  --write_buffer_size=64M ...
```

Using `--myisam_sort_buffer_size=16M` is probably enough for most cases.

Be aware that [myisamchk](#) uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory file system, out of memory errors can easily occur. If this happens, run [myisamchk](#) with the `--tmpdir=path` [\[351\]](#) option to specify a directory located on a file system that has more space.

When performing repair operations, [myisamchk](#) also needs a lot of disk space:

- Twice the size of the data file (the original file and a copy). This space is not needed if you do a repair with `--quick` [\[351\]](#); in this case, only the index file is re-created. *This space must be available on the same file system as the original data file*, as the copy is created in the same directory as the original.
- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space must be available on the same file system as the original data file.

- When using `--recover` [351] or `--sort-recover` [351] (but not when using `--safe-recover` [351]), you need space on disk for sorting. This space is allocated in the temporary directory (specified by `TMPDIR` or `--tmpdir=path` [351]). The following formula yields the amount of space required:

```
(largest_key + row_pointer_length) * number_of_rows * 2
```

You can check the length of the keys and the `row_pointer_length` with `myisamchk -dv tbl_name` (see Section 4.6.3.5, “Obtaining Table Information with `myisamchk`”). The `row_pointer_length` and `number_of_rows` values are the `Datafile pointer` and `Data records` values in the table description. To determine the `largest_key` value, check the `Key` lines in the table description. The `Len` column indicates the number of bytes for each key part. For a multiple-column index, the key size is the sum of the `Len` values for all key parts.

If you have a problem with disk space during repair, you can try `--safe-recover` [351] instead of `--recover` [351].

4.6.4. myisamlog — Display MyISAM Log File Contents

`myisamlog` processes the contents of a MyISAM log file.

Invoke `myisamlog` like this:

```
shell> myisamlog [options] [log_file [tbl_name] ...]
shell> isamlog [options] [log_file [tbl_name] ...]
```

The default operation is update (`-u`). If a recovery is done (`-r`), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is `myisam.log` for `myisamlog` and `isam.log` for `isamlog` if no `log_file` argument is given. If tables are named on the command line, only those tables are updated.

`myisamlog` supports the following options:

- `-?`, `-I`
Display a help message and exit.
- `-c N`
Execute only `N` commands.
- `-f N`
Specify the maximum number of open files.
- `-i`
Display extra information before exiting.
- `-o offset`
Specify the starting offset.
- `-p N`
Remove `N` components from path.
- `-r`
Perform a recovery operation.

- `-R record_pos_file record_pos`
Specify record position file and record position.
- `-u`
Perform an update operation.
- `-v`
Verbose mode. Print more output about what the program does. This option can be given multiple times to produce more and more output.
- `-w write_file`
Specify the write file.
- `-V`
Display version information.

4.6.5. myisampack — Generate Compressed, Read-Only MyISAM Tables

The `myisampack` utility compresses MyISAM tables. `myisampack` works by compressing each column in the table separately. Usually, `myisampack` packs the data file 40% to 70%.

When the table is used later, the server reads into memory the information needed to decompress columns. This results in much better performance when accessing individual rows, because you only have to uncompress exactly one row.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` does not work, MySQL falls back to normal read/write file operations.

Please note the following:

- If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process. It is safest to compress tables with the server stopped.
- After packing a table, it becomes read only. This is generally intended (such as when accessing packed tables on a CD).

Invoke `myisampack` like this:

```
shell> myisampack [options] file_name ...
```

Each file name argument should be the name of an index (`.MYI`) file. If you are not in the database directory, you should specify the path name to the file. It is permissible to omit the `.MYI` extension.

After you compress a table with `myisampack`, you should use `myisamchk -rq` to rebuild its indexes. [Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

`myisampack` supports the following options. It also reads option files and supports the options for processing them described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

- `--help [361], -?`
Display a help message and exit.

- `--backup [362], -b`

Make a backup of each table's data file using the name `tbl_name.OLD`.

- `--character-sets-dir=path [362]`

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--debug[=debug_options] [362], -# [debug_options]`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o'`.

- `--force [362], -f`

Produce a packed table even if it becomes larger than the original or if the intermediate file from an earlier invocation of `myisampack` exists. (`myisampack` creates an intermediate file named `tbl_name.TMD` in the database directory while it compresses the table. If you kill `myisampack`, the `.TMD` file might not be deleted.) Normally, `myisampack` exits with an error if it finds that `tbl_name.TMD` exists. With `--force [362]`, `myisampack` packs the table anyway.

- `--join=big_tbl_name [362], -j big_tbl_name`

Join all tables named on the command line into a single packed table `big_tbl_name`. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

`big_tbl_name` must not exist prior to the join operation. All source tables named on the command line to be merged into `big_tbl_name` must exist. The source tables are read for the join operation but not modified. The join operation does not create a `.frm` file for `big_tbl_name`, so after the join operation finishes, copy the `.frm` file from one of the source tables and name it `big_tbl_name.frm`.

- `--silent [362], -s`

Silent mode. Write output only when errors occur.

- `--test [362], -t`

Do not actually pack the table, just test packing it.

- `--tmpdir=path [362], -T path`

Use the named directory as the location where `myisampack` creates temporary files.

- `--verbose [362], -v`

Verbose mode. Write information about the progress of the packing operation and its result.

- `--version [362], -V`

Display version information and exit.

- `--wait [362], -w`

Wait and retry if the table is in use. If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```

shell> ls -l station.*
-rw-rw-r-- 1 monty my 994128 Apr 17 19:00 station.MYD
-rw-rw-r-- 1 monty my 53248 Apr 17 19:00 station.MYI
-rw-rw-r-- 1 monty my 5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file: station
Isam-version: 2
Creation time: 1996-03-13 10:08:58
Recover time: 1997-02-02 3:06:43
Data records: 1192 Deleted blocks: 0
Datafile parts: 1192 Deleted data: 0
Datafile pointer (bytes): 2 Keyfile pointer (bytes): 2
Max datafile length: 54657023 Max keyfile length: 33554431
Recordlength: 834
Record format: Fixed length

table description:
Key Start Len Index Type Root Blocksize Rec/key
1 2 4 unique unsigned long 1024 1024 1
2 32 30 multip. text 10240 1024 1

Field Start Length Type
1 1 1
2 2 4
3 6 4
4 10 1
5 11 20
6 31 1
7 32 30
8 62 35
9 97 35
10 132 35
11 167 4
12 171 16
13 187 35
14 222 4
15 226 16
16 242 20
17 262 20
18 282 20
19 302 30
20 332 4
21 336 4
22 340 1
23 341 8
24 349 8
25 357 8
26 365 2
27 367 2
28 369 4
29 373 4
30 377 1
31 378 2
32 380 8
33 388 4
34 392 4
35 396 4
36 400 4
37 404 1
38 405 4
39 409 4
40 413 4
41 417 4

```

```

42  421  4
43  425  4
44  429 20
45  449 30
46  479  1
47  480  1
48  481 79
49  560 79
50  639 79
51  718 79
52  797  8
53  805  1
54  806  1
55  807 20
56  827  4
57  831  4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal:      20  empty-space:   16  empty-zero:    12  empty-fill:   11
pre-space:   0  end-space:    12  table-lookups:  5  zero:         7
Original trees: 57 After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> ls -l station.*
-rw-rw-r--  1 monty  my           127874 Apr 17 19:00 station.MYD
-rw-rw-r--  1 monty  my           55296 Apr 17 19:04 station.MYI
-rw-rw-r--  1 monty  my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:      station
Isam-version:    2
Creation time:   1996-03-13 10:08:58
Recover time:   1997-04-17 19:04:26
Data records:      1192 Deleted blocks:      0
Datafile parts:    1192 Deleted data:      0
Datafile pointer (bytes): 3 Keyfile pointer (bytes): 1
Max datafile length: 16777215 Max keyfile length: 131071
Recordlength:      834
Record format: Compressed

table description:
Key Start Len Index  Type          Root  Blocksize  Rec/key
1   2     4   unique unsigned long  10240  1024       1
2   32    30  multip. text   54272  1024       1

Field Start Length Type          Huff tree  Bits
1     1     1   constant      1         0
2     2     4   zerofill(1)   2         9
3     6     4   no zeros, zerofill(1) 2         9
4    10     1   table-lookup  3         9
5    11    20   table-lookup  4         0
6    31     1   zerofill(1)   3         9
7    32    30   no endspace, not_always 5         9
8    62    35   no endspace, not_always, no empty 6         9
9    97    35   no empty      7         9
10   132   35   no endspace, not_always, no empty 6         9
11   167   4    zerofill(1)   2         9
12   171   16   no endspace, not_always, no empty 5         9
13   187   35   no endspace, not_always, no empty 6         9
14   222   4    zerofill(1)   2         9
15   226   16   no endspace, not_always, no empty 5         9

```

16	242	20	no endspace, not_always	8	9
17	262	20	no endspace, no empty	8	9
18	282	20	no endspace, no empty	5	9
19	302	30	no endspace, no empty	6	9
20	332	4	always zero	2	9
21	336	4	always zero	2	9
22	340	1		3	9
23	341	8	table-lookup	9	0
24	349	8	table-lookup	10	0
25	357	8	always zero	2	9
26	365	2		2	9
27	367	2	no zeros, zerofill(1)	2	9
28	369	4	no zeros, zerofill(1)	2	9
29	373	4	table-lookup	11	0
30	377	1		3	9
31	378	2	no zeros, zerofill(1)	2	9
32	380	8	no zeros	2	9
33	388	4	always zero	2	9
34	392	4	table-lookup	12	0
35	396	4	no zeros, zerofill(1)	13	9
36	400	4	no zeros, zerofill(1)	2	9
37	404	1		2	9
38	405	4	no zeros	2	9
39	409	4	always zero	2	9
40	413	4	no zeros	2	9
41	417	4	always zero	2	9
42	421	4	no zeros	2	9
43	425	4	always zero	2	9
44	429	20	no empty	3	9
45	449	30	no empty	3	9
46	479	1		14	4
47	480	1		14	4
48	481	79	no endspace, no empty	15	9
49	560	79	no empty	2	9
50	639	79	no empty	2	9
51	718	79	no endspace	16	9
52	797	8	no empty	2	9
53	805	1		17	1
54	806	1		3	9
55	807	20	no empty	3	9
56	827	4	no zeros, zerofill(2)	2	9
57	831	4	no zeros, zerofill(1)	2	9

myisampack displays the following kinds of information:

- **normal**

The number of columns for which no extra packing is used.

- **empty-space**

The number of columns containing values that are only spaces. These occupy one bit.

- **empty-zero**

The number of columns containing values that are only binary zeros. These occupy one bit.

- **empty-fill**

The number of integer columns that do not occupy the full byte range of their type. These are changed to a smaller type. For example, a **BIGINT** column (eight bytes) can be stored as a **TINYINT** column (one byte) if all its values are in the range from **-128** to **127**.

- **pre-space**

The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- `end-space`

The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- `table-lookup`

The column had only a small number of different values, which were converted to an `ENUM` before Huffman compression.

- `zero`

The number of columns for which all values are zero.

- `Original trees`

The initial number of Huffman trees.

- `After join`

The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, the `Field` lines displayed by `myisamchk -dvv` include additional information about each column:

- `Type`

The data type. The value may contain any of the following descriptors:

- `constant`

All rows have the same value.

- `no endspace`

Do not store endspace.

- `no endspace, not_always`

Do not store endspace and do not do endspace compression for all values.

- `no endspace, no empty`

Do not store endspace. Do not store empty values.

- `table-lookup`

The column was converted to an `ENUM`.

- `zerofill(N)`

The most significant `N` bytes in the value are always 0 and are not stored.

- `no zeros`

Do not store zeros.

- `always zero`

Zero values are stored using one bit.

- `Huff tree`

The number of the Huffman tree associated with the column.

- `Bits`

The number of bits used in the Huffman tree.

After you run `myisampack`, you must run `myisamchk` to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

After you have installed the packed table into the MySQL database directory, you should execute `mysqladmin flush-tables` to force `mysqld` to start using the new table.

To unpack a packed table, use the `--unpack [352]` option to `myisamchk`.

4.6.6. mysqlaccess — Client for Checking Access Privileges

`mysqlaccess` is a diagnostic tool that Yves Carrier has provided for the MySQL distribution. It checks the access privileges for a host name, user name, and database combination. Note that `mysqlaccess` checks access using only the `user`, `db`, and `host` tables. It does not check table, column, or routine privileges specified in the `tables_priv`, `columns_priv`, or `procs_priv` tables.

Invoke `mysqlaccess` like this:

```
shell> mysqlaccess [host_name [user_name [db_name]]] [options]
```

`mysqlaccess` supports the following options.

Table 4.10. mysqlaccess Options

Format	Option File	Description
<code>--brief [368]</code>	<code>brief [368]</code>	Generate reports in single-line tabular format
<code>--commit [368]</code>	<code>commit [368]</code>	Copy the new access privileges from the temporary tables to the original grant tables
<code>--copy [368]</code>	<code>copy [368]</code>	Reload the temporary grant tables from original ones
<code>--db=db_name [368]</code>	<code>db [368]</code>	Specify the database name
<code>--debug=# [368]</code>	<code>debug [368]</code>	Specify the debug level
<code>--help [368]</code>		Display help message and exit
<code>--host=host_name [368]</code>	<code>host [368]</code>	Connect to the MySQL server on the given host
<code>--howto [368]</code>	<code>howto [368]</code>	Display some examples that show how to use <code>mysqlaccess</code>
<code>--old_server [369]</code>	<code>old_server [369]</code>	Assume that the server is an old MySQL server (prior to MySQL 3.21)

Format	Option File	Description
--password[=password] [369]	password [369]	The password to use when connecting to the server
--plan [369]	plan [369]	Display suggestions and ideas for future releases
--preview [369]	preview [369]	Show the privilege differences after making changes to the temporary grant tables
--relnotes [369]	relnotes [369]	Display the release notes
--rhost=host_name [369]	rhost [369]	Connect to the MySQL server on the given host
--rollback [369]	rollback [369]	Undo the most recent changes to the temporary grant tables.
--spassword[=password] [369]	spassword [369]	The password to use when connecting to the server as the superuser
--superuser=user_name [369]	superuser [369]	Specify the user name for connecting as the superuser
--table [369]	table [369]	Generate reports in table format
--user=user_name, [369]	user [369]	MySQL user name to use when connecting to server
--version [369]		Display version information and exit

- `--help [368]`, `-?`

Display a help message and exit.

- `--brief [368]`, `-b`

Generate reports in single-line tabular format.

- `--commit [368]`

Copy the new access privileges from the temporary tables to the original grant tables. The grant tables must be flushed for the new privileges to take effect. (For example, execute a `mysqladmin reload` command.)

- `--copy [368]`

Reload the temporary grant tables from original ones.

- `--db=db_name [368]`, `-d db_name`

Specify the database name.

- `--debug=N [368]`

Specify the debug level. *N* can be an integer from 0 to 3.

- `--host=host_name [368]`, `-h host_name`

The host name to use in the access privileges.

- `--howto [368]`

Display some examples that show how to use `mysqlaccess`.

- `--old_server` [369]

Assume that the server is an old MySQL server (before MySQL 3.21) that does not yet know how to handle full `WHERE` clauses.

- `--password[=password]` [369], `-p[password]`

The password to use when connecting to the server. If you omit the `password` value following the `--password` [369] or `-p` option on the command line, `mysqlaccess` prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#).

- `--plan` [369]

Display suggestions and ideas for future releases.

- `--preview` [369]

Show the privilege differences after making changes to the temporary grant tables.

- `--relnotes` [369]

Display the release notes.

- `--rhost=host_name` [369], `-H host_name`

Connect to the MySQL server on the given host.

- `--rollback` [369]

Undo the most recent changes to the temporary grant tables.

- `--spassword[=password]` [369], `-P[password]`

The password to use when connecting to the server as the superuser. If you omit the `password` value following the `--spassword` [369] or `-P` option on the command line, `mysqlaccess` prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#).

- `--superuser=user_name` [369], `-U user_name`

Specify the user name for connecting as the superuser.

- `--table` [369], `-t`

Generate reports in table format.

- `--user=user_name` [369], `-u user_name`

The user name to use in the access privileges.

- `--version` [369], `-v`

Display version information and exit.

If your MySQL distribution is installed in some nonstandard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL = '/usr/local/bin/mysql'; # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a `Broken pipe` error will occur when you run `mysqlaccess`.

4.6.7. mysqlbinlog — Utility for Processing Binary Log Files

The server's binary log consists of files containing “events” that describe modifications to database contents. The server writes these files in binary format. To display their contents in text format, use the `mysqlbinlog` utility. You can also use `mysqlbinlog` to display the contents of relay log files written by a slave server in a replication setup because relay logs have the same format as binary logs. The binary log and relay log are discussed further in [Section 5.2.4, “The Binary Log”](#), and [Section 16.2.2, “Replication Relay and Status Logs”](#).

Invoke `mysqlbinlog` like this:

```
shell> mysqlbinlog [options] log_file ...
```

For example, to display the contents of the binary log file named `binlog.000003`, use this command:

```
shell> mysqlbinlog binlog.000003
```

The output includes events contained in `binlog.000003`. For statement-based logging, event information includes the SQL statement, the ID of the server on which it was executed, the timestamp when the statement was executed, how much time it took, and so forth. For row-based logging, the event indicates a row change rather than an SQL statement. See [Section 16.1.2, “Replication Formats”](#), for information about logging modes.

Events are preceded by header comments that provide additional information. For example:

```
# at 141
#100309 9:28:36 server id 123 end_log_pos 245
  Query thread_id=3350 exec_time=11 error_code=0
```

In the first line, the number following `at` indicates the starting position of the event in the binary log file.

The second line starts with a date and time indicating when the statement started on the server where the event originated. For replication, this timestamp is propagated to slave servers. `server_id` is the `server_id` [551] value of the server where the event originated. `end_log_pos` indicates where the next event starts (that is, it is the end position of the current event + 1). `thread_id` indicates which thread executed the event. `exec_time` is the time spent executing the event, on a master server. On a slave, it is the difference of the end execution time on the slave minus the beginning execution time on the master. The difference serves as an indicator of how much replication lags behind the master. `error_code` indicates the result from executing the event. Zero means that no error occurred.

The output from `mysqlbinlog` can be re-executed (for example, by using it as input to `mysql`) to redo the statements in the log. This is useful for recovery operations after a server crash. For other usage examples, see the discussion later in this section and in [Section 7.5, “Point-in-Time \(Incremental\) Recovery Using the Binary Log”](#).

Normally, you use `mysqlbinlog` to read binary log files directly and apply them to the local MySQL server. It is also possible to read binary logs from a remote server by using the `--read-from-remote-server` [376] option. To read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are `--host` [375], `--password` [375], `--port` [376], `--protocol` [376], `--socket` [376], and `--user` [377]; they are ignored except when you also use the `--read-from-remote-server` [376] option.

`mysqlbinlog` supports the following options, which can be specified on the command line or in the `[mysqlbinlog]` and `[client]` groups of an option file. `mysqlbinlog` also supports the options for

processing option files described at [Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”](#).

Table 4.11. mysqlbinlog Options

Format	Option File	Description	Introduced	Removed
--base64-output=value [372]	base64-output [372]	Print binary log entries using base-64 encoding		
--character-sets-dir=path [373]	character-sets-dir [373]	The directory where character sets are installed		
--database=db_name [373]	database [373]	List entries for just this database		
--debug[=debug_options] [374]	debug [374]	Write a debugging log		
--debug-check [374]	debug-check [374]	Print debugging information when the program exits		
--debug-info [374]	debug-info [374]	Print debugging information, memory and CPU statistics when the program exits		
--default-auth=plugin [374]	default-auth=plugin [374]	The authentication plugin to use	5.5.10	
--disable-log-bin [375]	disable-log-bin [375]	Disable binary logging		
--force-if-open [375]	force-if-open [375]	Read binary log files even if open or not closed properly		
--force-read [375]	force-read [375]	If mysqlbinlog reads a binary log event that it does not recognize, it prints a warning		
--help [372]		Display help message and exit		
--hexdump [375]	hexdump [375]	Display a hex dump of the log in comments		
--host=host_name [375]	host [375]	Connect to the MySQL server on the given host		
--local-load=path [375]	local-load [375]	Prepare local temporary files for LOAD DATA INFILE in the specified directory		
--offset=# [375]	offset [375]	Skip the first N entries in the log		
--password[=password] [375]	password [375]	The password to use when connecting to the server		
--plugin-dir=path [375]	plugin-dir=path [375]	The directory where plugins are located	5.5.10	
--port=port_num [376]	port [376]	The TCP/IP port number to use for the connection		
--position=# [376]	position [376]	Deprecated. Use --start-position		5.5.3
--protocol=type [376]	protocol [376]	The connection protocol to use		

Format	Option File	Description	Introduced	Removed
<code>--read-from-remote-server</code> [376]	<code>read-from-remote-server</code> [376]	Read binary log from MySQL server rather than local log file		
<code>--result-file=name</code> [376]	<code>result-file</code> [376]	Direct output to the given file		
<code>--server-id=id</code> [376]	<code>server-id</code> [376]	Extract only those events created by the server having the given server ID		
<code>--set-charset=charset_name</code> [376]	<code>set-charset</code> [376]	Add a SET NAMES charset_name statement to the output		
<code>--short-form</code> [376]	<code>short-form</code> [376]	Display only the statements contained in the log		
<code>--socket=path</code> [376]	<code>socket</code> [376]	For connections to localhost		
<code>--start-datetime=datetime</code> [377]	<code>start-datetime</code> [377]	Read binary log from first event with timestamp equal to or later than datetime argument		
<code>--start-position=#</code> [377]	<code>start-position</code> [377]	Read binary log from first event with position equal to or greater than argument		
<code>--stop-datetime=datetime</code> [377]	<code>stop-datetime</code> [377]	Stop reading binary log at first event with timestamp equal to or greater than datetime argument		
<code>--stop-position=#</code> [377]	<code>stop-position</code> [377]	Stop reading binary log at first event with position equal to or greater than argument		
<code>--to-last-log</code> [377]	<code>to-last-log</code> [377]	Do not stop at the end of requested binary log from a MySQL server, but rather continue printing to end of last binary log		
<code>--user=user_name</code>	<code>user</code> [377]	MySQL user name to use when connecting to server		
<code>--verbose</code> [377]		Reconstruct row events as SQL statements		
<code>--version</code> [378]		Display version information and exit		

- `--help` [372], `-?`

Display a help message and exit.

- `--base64-output [=value]` [372]

This option determines when events should be displayed encoded as base-64 strings using `BINLOG` statements. The option has these permissible values (not case sensitive):

- `AUTO` ("automatic") or `UNSPEC` ("unspecified") displays `BINLOG` statements automatically when necessary (that is, for format description events and row events). If no `--base64-output` [372] option is given, the effect is the same as `--base64-output=AUTO` [372].

**Note**

Automatic `BINLOG` display is the only safe behavior if you intend to use the output of `mysqlbinlog` to re-execute binary log file contents. The other option values are intended only for debugging or testing purposes because they may produce output that does not include all events in executable form.

- `ALWAYS` displays `BINLOG` statements whenever possible. If the `--base64-output` [372] option is given without a value, the effect is the same as `--base64-output=ALWAYS` [372].

**Note**

Changes to replication in MySQL 5.6 make output generated by this option unusable, so `ALWAYS` is deprecated as of MySQL 5.5.8 and will be an invalid value in MySQL 5.6

- `NEVER` causes `BINLOG` statements not to be displayed. `mysqlbinlog` exits with an error if a row event is found that must be displayed using `BINLOG`.
- `DECODE-ROWS` specifies to `mysqlbinlog` that you intend for row events to be decoded and displayed as commented SQL statements by also specifying the `--verbose` [377] option. Like `NEVER`, `DECODE-ROWS` suppresses display of `BINLOG` statements, but unlike `NEVER`, it does not exit with an error if a row event is found.

For examples that show the effect of `--base64-output` [372] and `--verbose` [377] on row event output, see [Section 4.6.7.2, “mysqlbinlog Row Event Display”](#).

- `--bind-address=ip_address` [277]

On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

This option is supported beginning with MySQL 5.5.8.

- `--character-sets-dir=path` [373]

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--database=db_name` [373], `-d db_name`

This option causes `mysqlbinlog` to output entries from the binary log (local log only) that occur while `db_name` is been selected as the default database by `USE`.

The `--database` [373] option for `mysqlbinlog` is similar to the `--binlog-do-db` [1894] option for `mysqld`, but can be used to specify only one database. If `--database` [373] is given multiple times, only the last instance is used.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--binlog-do-db` [1894] depend on whether statement-based or row-based logging is in use.

Statement-based logging. The `--database` [373] option works as follows:

- While `db_name` is the default database, statements are output whether they modify tables in `db_name` or a different database.

- Unless `db_name` is selected as the default database, statements are not output, even if they modify tables in `db_name`.
- There is an exception for `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE`. The database being *created*, *altered*, or *dropped* is considered to be the default database when determining whether to output the statement.

Suppose that the binary log was created by executing these statements using statement-based-logging:

```
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j) VALUES(200);
USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i) VALUES(102);
INSERT INTO db2.t2 (j) VALUES(201);
USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j) VALUES(202);
INSERT INTO t2 (j) VALUES(203);
```

`mysqlbinlog --database=test` does not output the first two `INSERT` statements because there is no default database. It outputs the three `INSERT` statements following `USE test`, but not the three `INSERT` statements following `USE db2`.

`mysqlbinlog --database=db2` does not output the first two `INSERT` statements because there is no default database. It does not output the three `INSERT` statements following `USE test`, but does output the three `INSERT` statements following `USE db2`.

Row-based logging. `mysqlbinlog` outputs only entries that change tables belonging to `db_name`. The default database has no effect on this. Suppose that the binary log just described was created using row-based logging rather than statement-based logging. `mysqlbinlog --database=test` outputs only those entries that modify `t1` in the test database, regardless of whether `USE` was issued or what the default database is.

If a server is running with `binlog_format` [1899] set to `MIXED` and you want it to be possible to use `mysqlbinlog` with the `--database` [373] option, you must ensure that tables that are modified are in the database selected by `USE`. (In particular, no cross-database updates should be used.)



Note

Prior to MySQL Cluster NDB 7.2.2, this option did not work correctly with MySQL Cluster tables unless, unless the binary log was generated using `--log-bin-use-v1-row-events=0` [1893]. (Bug #13067813)

- `--debug[=debug_options]` [374], `-# [debug_options]`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysqlbinlog.trace'`.

- `--debug-check` [374]

Print some debugging information when the program exits.

- `--debug-info` [374]

Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin` [374]

The client-side authentication plugin to use. See [Section 6.3.6, “Pluggable Authentication”](#).

This option was added in MySQL 5.5.10.

- `--disable-log-bin [375], -D`

Disable binary logging. This is useful for avoiding an endless loop if you use the `--to-last-log [377]` option and are sending the output to the same MySQL server. This option also is useful when restoring after a crash to avoid duplication of the statements you have logged.

This option requires that you have the `SUPER [697]` privilege. It causes `mysqlbinlog` to include a `SET sql_log_bin = 0` statement in its output to disable binary logging of the remaining output. The `SET` statement is ineffective unless you have the `SUPER [697]` privilege.

- `--force-if-open [375], -F`

Read binary log files even if they are open or were not closed properly.

- `--force-read [375], -f`

With this option, if `mysqlbinlog` reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, `mysqlbinlog` stops if it reads such an event.

- `--hexdump [375], -H`

Display a hex dump of the log in comments, as described in [Section 4.6.7.1, “mysqlbinlog Hex Dump Format”](#). The hex output can be helpful for replication debugging.

- `--host=host_name [375], -h host_name`

Get the binary log from the MySQL server on the given host.

- `--local-load=path [375], -l path`

Prepare local temporary files for `LOAD DATA INFILE` in the specified directory.



Important

These temporary files are not automatically removed by `mysqlbinlog` or any other MySQL program.

- `--offset=N [375], -o N`

Skip the first *N* entries in the log.

- `--password[=password] [375], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password [375]` or `-p` option on the command line, `mysqlbinlog` prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--plugin-dir=path [375]`

The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` [374] option is used to specify an authentication plugin but `mysqlbinlog` does not find it. See Section 6.3.6, “Pluggable Authentication”.

This option was added in MySQL 5.5.10.

- `--port=port_num` [376], `-P port_num`

The TCP/IP port number to use for connecting to a remote server.

- `--position=N` [376]

Deprecated. Use `--start-position` [377] instead. `--position` [376] was removed in MySQL 5.5.3.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [376]

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--read-from-remote-server` [376], `-R`

Read the binary log from a MySQL server rather than reading a local log file. Any connection parameter options are ignored unless this option is given as well. These options are `--host` [375], `--password` [375], `--port` [376], `--protocol` [376], `--socket` [376], and `--user` [377].

This option requires that the remote server be running. It works only for binary log files on the remote server, not relay log files.

- `--result-file=name` [376], `-r name`

Direct output to the given file.

- `--server-id=id` [376]

Display only those events created by the server having the given server ID.

- `--server-id-bits=N` [376]

Use only the first *N* bits of the `server_id` [551] to identify the server. If the binary log was written by a `mysqld` with `server-id-bits` set to less than 32 and user data stored in the most significant bit, running `mysqlbinlog` with `--server-id-bits` set to 32 enables this data to be seen.

This option is supported only by the versions of `mysqlbinlog` supplied with the MySQL Cluster distribution, or built from the MySQL Cluster sources.

- `--set-charset=charset_name` [376]

Add a `SET NAMES charset_name` statement to the output to specify the character set to be used for processing log files.

- `--short-form` [376], `-s`

Display only the statements contained in the log, without any extra information or row-based events. This is for testing only, and should not be used in production systems.

- `--socket=path` [376], `-S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--start-datetime=datetime` [377]

Start reading the binary log at the first event having a timestamp equal to or later than the `datetime` argument. The `datetime` value is relative to the local time zone on the machine where you run `mysqlbinlog`. The value should be in a format accepted for the `DATETIME` or `TIMESTAMP` data types. For example:

```
shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
```

This option is useful for point-in-time recovery. See [Section 7.3, “Example Backup and Recovery Strategy”](#).

- `--start-position=N` [377], `-j N`

Start reading the binary log at the first event having a position equal to or greater than `N`. This option applies to the first log file named on the command line.

This option is useful for point-in-time recovery. See [Section 7.3, “Example Backup and Recovery Strategy”](#).

- `--stop-datetime=datetime` [377]

Stop reading the binary log at the first event having a timestamp equal to or later than the `datetime` argument. This option is useful for point-in-time recovery. See the description of the `--start-datetime` [377] option for information about the `datetime` value.

This option is useful for point-in-time recovery. See [Section 7.3, “Example Backup and Recovery Strategy”](#).

- `--stop-position=N` [377]

Stop reading the binary log at the first event having a position equal to or greater than `N`. This option applies to the last log file named on the command line.

This option is useful for point-in-time recovery. See [Section 7.3, “Example Backup and Recovery Strategy”](#).

- `--to-last-log` [377], `-t`

Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server` [376].

- `--user=user_name` [377], `-u user_name`

The MySQL user name to use when connecting to a remote server.

- `--verbose` [377], `-v`

Reconstruct row events and display them as commented SQL statements. If this option is given twice, the output includes comments to indicate column data types and some metadata.

For examples that show the effect of `--base64-output` [372] and `--verbose` [377] on row event output, see [Section 4.6.7.2, “mysqlbinlog Row Event Display”](#).

- `--version` [378], `-V`

Display version information and exit.

You can also set the following variable by using `--var_name=value` syntax:

- `open_files_limit`

Specify the number of open file descriptors to reserve.

You can pipe the output of `mysqlbinlog` into the `mysql` client to execute the events contained in the binary log. This technique is used to recover from a crash when you have an old backup (see [Section 7.5, “Point-in-Time \(Incremental\) Recovery Using the Binary Log”](#)). For example:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p
```

Or:

```
shell> mysqlbinlog binlog.[0-9]* | mysql -u root -p
```

You can also redirect the output of `mysqlbinlog` to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the `mysql` program:

```
shell> mysqlbinlog binlog.000001 > tmpfile
shell> ... edit tmpfile ...
shell> mysql -u root -p < tmpfile
```

When `mysqlbinlog` is invoked with the `--start-position` [377] option, it displays only those events with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the `--stop-datetime` [377] option (to be able to say, for example, “roll forward my databases to how they were today at 10:30 a.m.”).

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using multiple connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first `mysql` process terminates, the server drops the temporary table. When the second `mysql` process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* `mysql` process to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 > /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

`mysqlbinlog` can produce output that reproduces a `LOAD DATA INFILE` operation without the original data file. `mysqlbinlog` copies the data to a temporary file and writes a `LOAD DATA LOCAL INFILE` statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the `--local-load` [375] option.

Because `mysqlbinlog` converts `LOAD DATA INFILE` statements to `LOAD DATA LOCAL INFILE` statements (that is, it adds `LOCAL`), both the client and the server that you use to process the statements must be configured with the `LOCAL` capability enabled. See [Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”](#).



Warning

The temporary files created for `LOAD DATA LOCAL` statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like `original_file_name-#-#`.

4.6.7.1. `mysqlbinlog` Hex Dump Format

The `--hexdump` [\[375\]](#) option causes `mysqlbinlog` to produce a hex dump of the binary log contents:

```
shell> mysqlbinlog --hexdump master-bin.000001
```

The hex output consists of comment lines beginning with `#`, so the output might look like this for the preceding command:

```
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1 end_log_pos 98
# Position Timestamp Type Master ID Size Master Pos Flags
# 00000004 9d fc 5c 43 0f 01 00 00 00 5e 00 00 00 62 00 00 00 00 00
# 00000017 04 00 35 2e 30 2e 31 35 2d 64 65 62 75 67 2d 6c |..5.0.15.debug.l|
# 00000027 6f 67 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |og.....|
# 00000037 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
# 00000047 00 00 00 00 9d fc 5c 43 13 38 0d 00 08 00 12 00 |.....C.8.....|
# 00000057 04 04 04 04 12 00 00 4b 00 04 1a |.....K...|
# Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
# at startup
ROLLBACK;
```

Hex dump output currently contains the elements in the following list. This format is subject to change. (For more information about binary log format, see [MySQL Internals: The Binary Log](#).)

- **Position:** The byte position within the log file.
- **Timestamp:** The event timestamp. In the example shown, `'9d fc 5c 43'` is the representation of `'051024 17:24:13'` in hexadecimal.
- **Type:** The event type code. In the example shown, `'0f'` indicates a `FORMAT_DESCRIPTION_EVENT`. The following table lists the possible type codes.

Type	Name	Meaning
00	<code>UNKNOWN_EVENT</code>	This event should never be present in the log.
01	<code>START_EVENT_V3</code>	This indicates the start of a log file written by MySQL 4 or earlier.
02	<code>QUERY_EVENT</code>	The most common type of events. These contain statements executed on the master.
03	<code>STOP_EVENT</code>	Indicates that master has stopped.
04	<code>ROTATE_EVENT</code>	Written when the master switches to a new log file.

Type	Name	Meaning
05	INTVAR_EVENT	Used for <code>AUTO_INCREMENT</code> values or when the <code>LAST_INSERT_ID()</code> [1194] function is used in the statement.
06	LOAD_EVENT	Used for <code>LOAD DATA INFILE</code> in MySQL 3.23.
07	SLAVE_EVENT	Reserved for future use.
08	CREATE_FILE_EVENT	Used for <code>LOAD DATA INFILE</code> statements. This indicates the start of execution of such a statement. A temporary file is created on the slave. Used in MySQL 4 only.
09	APPEND_BLOCK_EVENT	Contains data for use in a <code>LOAD DATA INFILE</code> statement. The data is stored in the temporary file on the slave.
0a	EXEC_LOAD_EVENT	Used for <code>LOAD DATA INFILE</code> statements. The contents of the temporary file is stored in the table on the slave. Used in MySQL 4 only.
0b	DELETE_FILE_EVENT	Rollback of a <code>LOAD DATA INFILE</code> statement. The temporary file should be deleted on the slave.
0c	NEW_LOAD_EVENT	Used for <code>LOAD DATA INFILE</code> in MySQL 4 and earlier.
0d	RAND_EVENT	Used to send information about random values if the <code>RAND()</code> [1128] function is used in the statement.
0e	USER_VAR_EVENT	Used to replicate user variables.
0f	FORMAT_DESCRIPTION_EVENT	This indicates the start of a log file written by MySQL 5 or later.
10	XID_EVENT	Event indicating commit of an XA transaction.
11	BEGIN_LOAD_QUERY_EVENT	Used for <code>LOAD DATA INFILE</code> statements in MySQL 5 and later.
12	EXECUTE_LOAD_QUERY_EVENT	Used for <code>LOAD DATA INFILE</code> statements in MySQL 5 and later.
13	TABLE_MAP_EVENT	Information about a table definition. Used in MySQL 5.1.5 and later.
14	PRE_GA_WRITE_ROWS_EVENT	Row data for a single table that should be created. Used in MySQL 5.1.5 to 5.1.17.
15	PRE_GA_UPDATE_ROWS_EVENT	Row data for a single table that needs to be updated. Used in MySQL 5.1.5 to 5.1.17.
16	PRE_GA_DELETE_ROWS_EVENT	Row data for a single table that should be deleted. Used in MySQL 5.1.5 to 5.1.17.
17	WRITE_ROWS_EVENT	Row data for a single table that should be created. Used in MySQL 5.1.18 and later.
18	UPDATE_ROWS_EVENT	Row data for a single table that needs to be updated. Used in MySQL 5.1.18 and later.
19	DELETE_ROWS_EVENT	Row data for a single table that should be deleted. Used in MySQL 5.1.18 and later.
1a	INCIDENT_EVENT	Something out of the ordinary happened. Added in MySQL 5.1.18.

- `Master ID`: The server ID of the master that created the event.

- **Size:** The size in bytes of the event.
- **Master Pos:** The position of the next event in the original master log file.
- **Flags:** 16 flags. Currently, the following flags are used. The others are reserved for future use.

Flag	Name	Meaning
01	LOG_EVENT_BINLOG_IN_USE_F	Log file correctly closed. (Used only in <code>FORMAT_DESCRIPTION_EVENT</code> .) If this flag is set (if the flags are, for example, '01 00') in a <code>FORMAT_DESCRIPTION_EVENT</code> , the log file has not been properly closed. Most probably this is because of a master crash (for example, due to power failure).
02		Reserved for future use.
04	LOG_EVENT_THREAD_SPECIFIC_F	Set if the event is dependent on the connection it was executed in (for example, '04 00'), for example, if the event uses temporary tables.
08	LOG_EVENT_SUPPRESS_USE_F	Set in some circumstances when the event is not dependent on the default database.

4.6.7.2. mysqlbinlog Row Event Display

The following examples illustrate how `mysqlbinlog` displays row events that specify data modifications. These correspond to events with the `WRITE_ROWS_EVENT`, `UPDATE_ROWS_EVENT`, and `DELETE_ROWS_EVENT` type codes. The `--base64-output=DECODE-ROWS` [372] and `--verbose` [377] options may be used to affect row event output.

Suppose that the server is using row-based binary logging and that you execute the following sequence of statements:

```
CREATE TABLE t
(
  id INT NOT NULL,
  name VARCHAR(20) NOT NULL,
  date DATE NULL
) ENGINE = InnoDB;

START TRANSACTION;
INSERT INTO t VALUES(1, 'apple', NULL);
UPDATE t SET name = 'pear', date = '2009-01-01' WHERE id = 1;
DELETE FROM t WHERE id = 1;
COMMIT;
```

By default, `mysqlbinlog` displays row events encoded as base-64 strings using `BINLOG` statements. Omitting extraneous lines, the output for the row events produced by the preceding statement sequence looks like this:

```
shell> mysqlbinlog log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258  Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAAANoAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAAVhcHBsZQ==
/*!*/;
...
```

```
# at 302
#080828 15:03:08 server id 1  end_log_pos 356  Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAAC4BAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAAAngAAAGQBAAAQABEAAAAAAAAEAAA///AEAAAAFYXBwbGX4AQAAAAARwZWFyIbIP
'/*!*/;
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442  Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAABAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAAKgAAALoBAAAQABEAAAAAAAAEAAA//4AQAAAAARwZWFyIbIP
'/*!*/;
```

To see the row events as comments in the form of “pseudo-SQL” statements, run `mysqlbinlog` with the `--verbose` [377] or `-v` option. The output will contain lines beginning with `###`:

```
shell> mysqlbinlog -v log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258  Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAANoAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAAKAAAAIBAAAQABEAAAAAAAAEAAA//8AQAAAAVhcHBsZQ==
'/*!*/;
### INSERT INTO test.t
### SET
###   @1=1
###   @2='apple'
###   @3=NULL
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356  Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAAC4BAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAAAngAAAGQBAAAQABEAAAAAAAAEAAA///AEAAAAFYXBwbGX4AQAAAAARwZWFyIbIP
'/*!*/;
### UPDATE test.t
### WHERE
###   @1=1
###   @2='apple'
###   @3=NULL
### SET
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442  Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAABAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAAKgAAALoBAAAQABEAAAAAAAAEAAA//4AQAAAAARwZWFyIbIP
'/*!*/;
### DELETE FROM test.t
### WHERE
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
```

Specify `--verbose` [377] or `-v` twice to also display data types and some metadata for each column. The output will contain an additional comment following each column change:

```

shell> mysqlbinlog -vv log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258  Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAANoAAAAABEAAAAAAAAABHRlc3QAAXQAawMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
### INSERT INTO test.t
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356  Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAAC4BAAAAABEAAAAAAAAABHRlc3QAAXQAawMPCgIUAAQ=
fAS3SBgBAAAAAGAAQABAAAQABEAAAAAAAAEAAA//AEAAAAFYXBwbGX4AQAAARwZWZyIbIP
'/*!*/;
### UPDATE test.t
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442  Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAABAAAAABEAAAAAAAAABHRlc3QAAXQAawMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAAEAA//4AQAAARwZWZyIbIP
'/*!*/;
### DELETE FROM test.t
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */

```

You can tell `mysqlbinlog` to suppress the `BINLOG` statements for row events by using the `--base64-output=DECODE-ROWS` [372] option. This is similar to `--base64-output=NEVER` [372] but does not exit with an error if a row event is found. The combination of `--base64-output=DECODE-ROWS` [372] and `--verbose` [377] provides a convenient way to see row events only as SQL statements:

```

shell> mysqlbinlog -v --base64-output=DECODE-ROWS log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258  Write_rows: table id 17 flags: STMT_END_F
### INSERT INTO test.t
### SET
###   @1=1
###   @2='apple'
###   @3=NULL
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356  Update_rows: table id 17 flags: STMT_END_F
### UPDATE test.t
### WHERE
###   @1=1
###   @2='apple'

```

```

### @3=NULL
### SET
### @1=1
### @2='pear'
### @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1 end_log_pos 442 Delete_rows: table id 17 flags: STMT_END_F
### DELETE FROM test.t
### WHERE
### @1=1
### @2='pear'
### @3='2009:01:01'

```



Note

You should not suppress `BINLOG` statements if you intend to re-execute `mysqlbinlog` output.

The SQL statements produced by `--verbose` [377] for row events are much more readable than the corresponding `BINLOG` statements. However, they do not correspond exactly to the original SQL statements that generated the events. The following limitations apply:

- The original column names are lost and replaced by `@N`, where `N` is a column number.
- Character set information is not available in the binary log, which affects string column display:
 - There is no distinction made between corresponding binary and nonbinary string types (`BINARY` and `CHAR`, `VARBINARY` and `VARCHAR`, `BLOB` and `TEXT`). The output uses a data type of `STRING` for fixed-length strings and `VARSTRING` for variable-length strings.
 - For multi-byte character sets, the maximum number of bytes per character is not present in the binary log, so the length for string types is displayed in bytes rather than in characters. For example, `STRING(4)` will be used as the data type for values from either of these column types:

```

CHAR(4) CHARACTER SET latin1
CHAR(2) CHARACTER SET ucs2

```

- Due to the storage format for events of type `UPDATE_ROWS_EVENT`, `UPDATE` statements are displayed with the `WHERE` clause preceding the `SET` clause.

Proper interpretation of row events requires the information from the format description event at the beginning of the binary log. Because `mysqlbinlog` does not know in advance whether the rest of the log contains row events, by default it displays the format description event using a `BINLOG` statement in the initial part of the output.

If the binary log is known not to contain any events requiring a `BINLOG` statement (that is, no row events), the `--base64-output=NEVER` [372] option can be used to prevent this header from being written.

4.6.8. `mysqldumpslow` — Summarize Slow Query Log Files

The MySQL slow query log contains information about queries that take a long time to execute (see [Section 5.2.5, “The Slow Query Log”](#)). `mysqldumpslow` parses MySQL slow query log files and prints a summary of their contents.

Normally, `mysqldumpslow` groups queries that are similar except for the particular values of number and string data values. It “abstracts” these values to `N` and `'S'` when displaying summary output. The `-a` and `-n` options can be used to modify value abstracting behavior.

Invoke `mysqldumpslow` like this:

```
shell> mysqldumpslow [options] [log_file ...]
```

`mysqldumpslow` supports the following options.

Table 4.12. `mysqldumpslow` Options

Format	Option File	Description
<code>-a [385]</code>		Do not abstract all numbers to N and strings to S
<code>-n num [385]</code>		Abstract numbers with at least the specified digits
<code>--debug [385]</code>	<code>debug [385]</code>	Write debugging information
<code>-g pattern [385]</code>		Only consider statements that match the pattern
<code>--help [385]</code>		Display help message and exit
<code>-h name [385]</code>		Host name of the server in the log file name
<code>-i name [385]</code>		Name of the server instance
<code>-l [385]</code>		Do not subtract lock time from total time
<code>-r [386]</code>		Reverse the sort order
<code>-s value [386]</code>		How to sort output
<code>-t num [386]</code>		Display only first num queries
<code>--verbose [386]</code>	<code>verbose [386]</code>	Verbose mode

- `--help [385]`
Display a help message and exit.
- `-a`
Do not abstract all numbers to N and strings to 'S'.
- `--debug [385]`, `-d`
Run in debug mode.
- `-g pattern`
Consider only queries that match the (grep-style) pattern.
- `-h host_name`
Host name of MySQL server for `*-slow.log` file name. The value can contain a wildcard. The default is `*` (match all).
- `-i name`
Name of server instance (if using `mysql.server` startup script).
- `-l`
Do not subtract lock time from total time.
- `-n N`

Abstract numbers with at least *N* digits within names.

- `-r`

Reverse the sort order.

- `-s sort_type`

How to sort the output. The value of *sort_type* should be chosen from the following list:

- `t, at`: Sort by query time or average query time
- `l, al`: Sort by lock time or average lock time
- `r, ar`: Sort by rows sent or average rows sent
- `c`: Sort by count

By default, `mysqldumpslow` sorts by average query time (equivalent to `-s at`).

- `-t N`

Display only the first *N* queries in the output.

- `--verbose [386], -v`

Verbose mode. Print more information about what the program does.

Example of usage:

```
shell> mysqldumpslow

Reading mysql slow query log from /usr/local/mysql/data/mysqld51-apple-slow.log
Count: 1 Time=4.32s (4s) Lock=0.00s (0s) Rows=0.0 (0), root[root]@localhost
insert into t2 select * from t1

Count: 3 Time=2.53s (7s) Lock=0.00s (0s) Rows=0.0 (0), root[root]@localhost
insert into t2 select * from t1 limit N

Count: 3 Time=2.13s (6s) Lock=0.00s (0s) Rows=0.0 (0), root[root]@localhost
insert into t1 select * from t1
```

4.6.9. mysqlhotcopy — A Database Backup Program

`mysqlhotcopy` is a Perl script that was originally written and contributed by Tim Bunce. It uses `FLUSH TABLES`, `LOCK TABLES`, and `cp` or `scp` to make a database backup. It is a fast way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. `mysqlhotcopy` works only for backing up `MyISAM` and `ARCHIVE` tables. It runs on Unix.

To use `mysqlhotcopy`, you must have read access to the files for the tables that you are backing up, the `SELECT [697]` privilege for those tables, the `RELOAD [696]` privilege (to be able to execute `FLUSH TABLES`), and the `LOCK TABLES [696]` privilege (to be able to lock the tables).

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mysqlhotcopy db_name./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde (“~”):

```
shell> mysqlhotcopy db_name./~regex/
```

`mysqlhotcopy` supports the following options, which can be specified on the command line or in the `[mysqlhotcopy]` and `[client]` groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.13. `mysqlhotcopy` Options

Format	Option File	Description	Introduced
<code>--addtodest [388]</code>	<code>addtodest [388]</code>	Do not rename target directory (if it exists); merely add files to it	
<code>--allowold [388]</code>	<code>allowold [388]</code>	Do not abort if a target exists; rename it by adding an <code>_old</code> suffix	
<code>--checkpoint=db_name.tbl_name [388]</code>	<code>checkpoint [388]</code>	Insert checkpoint entries	
<code>--chroot=path [388]</code>	<code>chroot [388]</code>	Base directory of the chroot jail in which <code>mysqld</code> operates	
<code>--debug [388]</code>	<code>debug [388]</code>	Write a debugging log	
<code>--dryrun [388]</code>	<code>dryrun [388]</code>	Report actions without performing them	
<code>--flushlog [388]</code>	<code>flushlog [388]</code>	Flush logs after all tables are locked	
<code>--help [388]</code>		Display help message and exit	
<code>--host=host_name [388]</code>	<code>host [388]</code>	Connect to the MySQL server on the given host	
<code>--keepold [388]</code>	<code>keepold [388]</code>	Do not delete previous (renamed) target when done	
<code>--method [388]</code>	<code>method [388]</code>	The method for copying files	
<code>--noindices [388]</code>	<code>noindices [388]</code>	Do not include full index files in the backup	
<code>--old_server [389]</code>	<code>old_server [389]</code>	Connect to server that does not support <code>FLUSH TABLES tbl_list WITH READ LOCK</code>	5.5.3
<code>--password[=password] [388]</code>	<code>password [388]</code>	The password to use when connecting to the server	
<code>--port=port_num [389]</code>	<code>port [389]</code>	The TCP/IP port number to use for the connection	
<code>--quiet [389]</code>	<code>quiet [389]</code>	Be silent except for errors	
<code>--regexp [389]</code>	<code>regexp [389]</code>	Copy all databases with names that match the given regular expression	
<code>--resetmaster [389]</code>	<code>resetmaster [389]</code>	Reset the binary log after locking all the tables	
<code>--resetslave [389]</code>	<code>resetslave [389]</code>	Reset the <code>master.info</code> file after locking all the tables	
<code>--socket=path [389]</code>	<code>socket [389]</code>	For connections to localhost	
<code>--tmpdir=path [389]</code>	<code>tmpdir [389]</code>	The temporary directory	
<code>--user=user_name, [389]</code>	<code>user [389]</code>	MySQL user name to use when connecting to server	

- `--help [388], -?`

Display a help message and exit.
- `--addtodest [388]`

Do not rename target directory (if it exists); merely add files to it.
- `--allowold [388]`

Do not abort if a target exists; rename it by adding an `_old` suffix.
- `--checkpoint=db_name.tbl_name [388]`

Insert checkpoint entries into the specified database `db_name` and table `tbl_name`.
- `--chroot=path [388]`

Base directory of the `chroot` jail in which `mysqld` operates. The `path` value should match that of the `--chroot [438]` option given to `mysqld`.
- `--debug [388]`

Enable debug output.
- `--dryrun [388], -n`

Report actions without performing them.
- `--flushlog [388]`

Flush logs after all tables are locked.
- `--host=host_name [388], -h host_name`

The host name of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to `localhost` using a Unix socket file.
- `--keepold [388]`

Do not delete previous (renamed) target when done.
- `--method=command [388]`

The method for copying files (`cp` or `scp`). The default is `cp`.
- `--noindices [388]`

Do not include full index files for `MyISAM` tables in the backup. This makes the backup smaller and faster. The indexes for reloaded tables can be reconstructed later with `myisamchk -rq`.
- `--password=password [388], -ppassword`

The password to use when connecting to the server. The password value is not optional for this option, unlike for other MySQL programs.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--port=port_num` [389], `-P port_num`

The TCP/IP port number to use when connecting to the local server.

- `--old_server` [389]

As of MySQL 5.5.3, `mysqlhotcopy` uses `FLUSH TABLES tbl_list WITH READ LOCK` to flush and lock tables. Use the `--old_server` [389] option if the server is older than 5.5.3, which is when that statement was introduced. This option was added in MySQL 5.5.3.

- `--quiet` [389], `-q`

Be silent except for errors.

- `--record_log_pos=db_name.tbl_name` [389]

Record master and slave status in the specified database `db_name` and table `tbl_name`.

- `--regexp=expr` [389]

Copy all databases with names that match the given regular expression.

- `--resetmaster` [389]

Reset the binary log after locking all the tables.

- `--resetslave` [389]

Reset the `master.info` file after locking all the tables.

- `--socket=path` [389], `-S path`

The Unix socket file to use for connections to `localhost`.

- `--suffix=str` [389]

The suffix to use for names of copied databases.

- `--tmpdir=path` [389]

The temporary directory. The default is `/tmp`.

- `--user=user_name` [389], `-u user_name`

The MySQL user name to use when connecting to the server.

Use `perldoc` for additional `mysqlhotcopy` documentation, including information about the structure of the tables needed for the `--checkpoint` [388] and `--record_log_pos` [389] options:

```
shell> perldoc mysqlhotcopy
```

4.6.10. `mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine

`mysql_convert_table_format` converts the tables in a database to use a particular storage engine (MyISAM by default). `mysql_convert_table_format` is written in Perl and requires that the `DBI` and `DBD::mysql` Perl modules be installed (see [Section 2.14, “Perl Installation Notes”](#)).

Invoke `mysql_convert_table_format` like this:

```
shell> mysql_convert_table_format [options]db_name
```

The `db_name` argument indicates the database containing the tables to be converted.

`mysql_convert_table_format` supports the options described in the following list.

- `--help` [390]

Display a help message and exit.

- `--force` [390]

Continue even if errors occur.

- `--host=host_name` [390]

Connect to the MySQL server on the given host.

- `--password=password` [390]

The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--port=port_num` [390]

The TCP/IP port number to use for the connection.

- `--socket=path` [390]

For connections to `localhost`, the Unix socket file to use.

- `--type=engine_name` [390]

Specify the storage engine that the tables should be converted to use. The default is `MyISAM` if this option is not given.

- `--user=user_name` [390]

The MySQL user name to use when connecting to the server.

- `--verbose` [390]

Verbose mode. Print more information about what the program does.

- `--version` [390]

Display version information and exit.

4.6.11. `mysql_find_rows` — Extract SQL Statements from Files

`mysql_find_rows` reads files containing SQL statements and extracts statements that match a given regular expression or that contain `USE db_name` or `SET` statements. The utility expects statements to be terminated with semicolon (;) characters.

Invoke `mysql_find_rows` like this:

```
shell> mysql_find_rows [options] [file_name ...]
```

Each *file_name* argument should be the name of file containing SQL statements. If no file names are given, `mysql_find_rows` reads the standard input.

Examples:

```
mysql_find_rows --regexp=problem_table --rows=20 < update.log
mysql_find_rows --regexp=problem_table update-log.1 update-log.2
```

`mysql_find_rows` supports the following options:

- `--help` [391], `--Information` [391]
Display a help message and exit.
- `--regexp=pattern` [391]
Display queries that match the pattern.
- `--rows=N` [391]
Quit after displaying *N* queries.
- `--skip-use-db` [391]
Do not include `USE db_name` statements in the output.
- `--start_row=N` [391]
Start output from this row.

4.6.12. `mysql_fix_extensions` — Normalize Table File Name Extensions

`mysql_fix_extensions` converts the extensions for `MyISAM` (or `ISAM`) table files to their canonical forms. It looks for files with extensions matching any lettercase variant of `.frm`, `.myd`, `.myi`, `.isd`, and `.ism` and renames them to have extensions of `.frm`, `.MYD`, `.MYI`, `.ISD`, and `.ISM`, respectively. This can be useful after transferring the files from a system with case-insensitive file names (such as Windows) to a system with case-sensitive file names.

Invoke `mysql_fix_extensions` like this, where *data_dir* is the path name to the MySQL data directory.

```
shell> mysql_fix_extensions data_dir
```

4.6.13. `mysql_setpermission` — Interactively Set Permissions in Grant Tables

`mysql_setpermission` is a Perl script that was originally written and contributed by Luuk de Boer. It interactively sets permissions in the MySQL grant tables. `mysql_setpermission` is written in Perl and requires that the `DBI` and `DBD::mysql` Perl modules be installed (see [Section 2.14](#), “Perl Installation Notes”).

Invoke `mysql_setpermission` like this:

```
shell> mysql_setpermission [options]
```

options should be either `--help` [392] to display the help message, or options that indicate how to connect to the MySQL server. The account used when you connect determines which permissions you have when attempting to modify existing permissions in the grant tables.

`mysql_setpermissions` also reads options from the `[client]` and `[perl]` groups in the `.my.cnf` file in your home directory, if the file exists.

`mysql_setpermission` supports the following options:

- `--help` [392]

Display a help message and exit.

- `--host=host_name` [392]

Connect to the MySQL server on the given host.

- `--password=password` [392]

The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

- `--port=port_num` [392]

The TCP/IP port number to use for the connection.

- `--socket=path` [392]

For connections to `localhost`, the Unix socket file to use.

- `--user=user_name` [392]

The MySQL user name to use when connecting to the server.

4.6.14. `mysql_waitpid` — Kill Process and Wait for Its Termination

`mysql_waitpid` signals a process to terminate and waits for the process to exit. It uses the `kill()` system call and Unix signals, so it runs on Unix and Unix-like systems.

Invoke `mysql_waitpid` like this:

```
shell> mysql_waitpid [options] pid wait_time
```

`mysql_waitpid` sends signal 0 to the process identified by `pid` and waits up to `wait_time` seconds for the process to terminate. `pid` and `wait_time` must be positive integers.

If process termination occurs within the wait time or the process does not exist, `mysql_waitpid` returns 0. Otherwise, it returns 1.

If the `kill()` system call cannot handle signal 0, `mysql_waitpid()` uses signal 1 instead.

`mysql_waitpid` supports the following options:

- `--help` [392], `-, -I`

Display a help message and exit.

- `--verbose` [393], `-v`

Verbose mode. Display a warning if signal 0 could not be used and signal 1 is used instead.

- `--version` [393], `-V`

Display version information and exit.

4.6.15. `mysql_zap` — Kill Processes That Match a Pattern

`mysql_zap` kills processes that match a pattern. It uses the `ps` command and Unix signals, so it runs on Unix and Unix-like systems.

Invoke `mysql_zap` like this:

```
shell> mysql_zap [-signal] [-?Ift] pattern
```

A process matches if its output line from the `ps` command contains the pattern. By default, `mysql_zap` asks for confirmation for each process. Respond `y` to kill the process, or `q` to exit `mysql_zap`. For any other response, `mysql_zap` does not attempt to kill the process.

If the `-signal` option is given, it specifies the name or number of the signal to send to each process. Otherwise, `mysql_zap` tries first with `TERM` (signal 15) and then with `KILL` (signal 9).

`mysql_zap` supports the following additional options:

- `--help` [393], `-?`, `-I`

Display a help message and exit.

- `-f`

Force mode. `mysql_zap` attempts to kill each process without confirmation.

- `-t`

Test mode. Display information about each process but do not kill it.

4.7. MySQL Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

In shell scripts, you can use the `my_print_defaults` program to parse option files and see what options would be used by a given program. The following example shows the output that `my_print_defaults` might produce when asked to show the options found in the `[client]` and `[mysql]` groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add

only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

4.7.1. `msql2mysql` — Convert mSQL Programs for Use with MySQL

Initially, the MySQL C API was developed to be very similar to that for the mSQL database system. Because of this, mSQL programs often can be converted relatively easily for use with MySQL by changing the names of the C API functions.

The `msql2mysql` utility performs the conversion of mSQL C API function calls to their MySQL equivalents. `msql2mysql` converts the input file in place, so make a copy of the original before converting it. For example, use `msql2mysql` like this:

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

Then examine `client-prog.c` and make any post-conversion revisions that may be necessary.

`msql2mysql` uses the `replace` utility to make the function name substitutions. See [Section 4.8.2, “replace — A String-Replacement Utility”](#).

4.7.2. `mysql_config` — Get Compile Options for Compiling Clients

`mysql_config` provides you with useful information for compiling your MySQL client and connecting it to MySQL.

`mysql_config` supports the following options.

- `--cflags` [\[394\]](#)

Compiler flags to find include files and critical compiler flags and defines used when compiling the `libmysqlclient` library. The options returned are tied to the specific compiler that was used when the library was created and might clash with the settings for your own compiler. Use `--include` [\[394\]](#) for more portable options that contain only include paths.

- `--include` [\[394\]](#)

Compiler options to find MySQL include files.

- `--libmysqld-libs` [\[394\]](#), `--embedded` [\[394\]](#)

Libraries and options required to link with the MySQL embedded server.

- `--libs` [\[394\]](#)

Libraries and options required to link with the MySQL client library.

- `--libs_r` [\[394\]](#)

Libraries and options required to link with the thread-safe MySQL client library.

- `--plugindir` [\[394\]](#)

The default plugin directory path name, defined when configuring MySQL.

- `--port [395]`

The default TCP/IP port number, defined when configuring MySQL.

- `--socket [395]`

The default Unix socket file, defined when configuring MySQL.

- `--version [395]`

Version number for the MySQL distribution.

If you invoke `mysql_config` with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
--cflags      [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
--include     [-I/usr/local/mysql/include/mysql]
--libs        [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz
              -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
--libs_r      [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
              -lpthread -lz -lcrypt -lnsl -lm -lpthread]
--socket      [/tmp/mysql.sock]
--port        [3306]
--version     [4.0.16]
--libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread -lz
                -lcrypt -lnsl -lm -lpthread -lrt]
```

You can use `mysql_config` within a command line to include the value that it displays for a particular option. For example, to compile a MySQL client program, use `mysql_config` as follows:

```
shell> CFG=/usr/local/mysql/bin/mysql_config
shell> sh -c "gcc -o progname ` $CFG --include ` progname.c ` $CFG --libs `"
```

When you use `mysql_config` this way, be sure to invoke it within backtick (“`) characters. That tells the shell to execute it and substitute its output into the surrounding command.

4.7.3. my_print_defaults — Display Options from Option Files

`my_print_defaults` displays the options that are present in option groups of option files. The output indicates what options will be used by programs that read the specified option groups. For example, the `mysqlcheck` program reads the `[mysqlcheck]` and `[client]` option groups. To see what options are present in those groups in the standard option files, invoke `my_print_defaults` like this:

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=secret
--host=localhost
```

The output consists of options, one per line, in the form that they would be specified on the command line.

`my_print_defaults` supports the following options.

- `--help [395], -?`

Display a help message and exit.

- `--config-file=file_name` [396], `--defaults-file=file_name` [396], `-c file_name`

Read only the given option file.

- `--debug=debug_options` [396], `-# debug_options`

Write a debugging log. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/my_print_defaults.trace'`.

- `--defaults-extra-file=file_name` [396], `--extra-file=file_name` [396], `-e file_name`

Read this option file after the global option file but (on Unix) before the user option file.

- `--defaults-group-suffix=suffix` [396], `-g suffix`

In addition to the groups named on the command line, read groups that have the given suffix.

- `--no-defaults` [248], `-n`

Return an empty string.

- `--verbose` [396], `-v`

Verbose mode. Print more information about what the program does.

- `--version` [396], `-V`

Display version information and exit.

4.7.4. `resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols

`resolve_stack_dump` resolves a numeric stack dump to symbols.

Invoke `resolve_stack_dump` like this:

```
shell> resolve_stack_dump [options] symbols_file [numeric_dump_file]
```

The symbols file should include the output from the `nm --numeric-sort mysqld` command. The numeric dump file should contain a numeric stack track from `mysqld`. If no numeric dump file is named on the command line, the stack trace is read from the standard input.

`resolve_stack_dump` supports the following options.

- `--help` [396], `-h`

Display a help message and exit.

- `--numeric-dump-file=file_name` [396], `-n file_name`

Read the stack trace from the given file.

- `--symbols-file=file_name` [396], `-s file_name`

Use the given symbols file.

- `--version` [396], `-V`

Display version information and exit.

4.8. Miscellaneous Programs

4.8.1. `pererror` — Explain Error Codes

For most system errors, MySQL displays, in addition to an internal text message, the system error code in one of the following styles:

```
message ... (errno: #)
message ... (Errcode: #)
```

You can find out what the error code means by examining the documentation for your system or by using the `pererror` utility.

`pererror` prints a description for a system error code or for a storage engine (table handler) error code.

Invoke `pererror` like this:

```
shell> pererror [options] errorcode ...
```

Example:

```
shell> pererror 13 64
OS error code 13: Permission denied
OS error code 64: Machine is not on the network
```

To obtain the error message for a MySQL Cluster error code, invoke `pererror` with the `--ndb` [397] option:

```
shell> pererror --ndb errorcode
```

Note that the meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

`pererror` supports the following options.

- `--help` [397], `--info` [397], `-I`, `-?`

Display a help message and exit.

- `--ndb` [397]

Print the error message for a MySQL Cluster error code.

- `--silent` [397], `-s`

Silent mode. Print only the error message.

- `--verbose` [397], `-v`

Verbose mode. Print error code and message. This is the default behavior.

- `--version` [397], `-V`

Display version information and exit.

4.8.2. `replace` — A String-Replacement Utility

The `replace` utility program changes strings in place in files or on the standard input.

Invoke `replace` in one of the following ways:

```
shell> replace from to [from to] ... -- file_name [file_name] ...
shell> replace from to [from to] ... < file_name
```

`from` represents a string to look for and `to` represents its replacement. There can be one or more pairs of strings.

Use the `--` option to indicate where the string-replacement list ends and the file names begin. In this case, any file named on the command line is modified in place, so you may want to make a copy of the original before converting it. `replace` prints a message indicating which of the input files it actually modifies.

If the `--` option is not given, `replace` reads the standard input and writes to the standard output.

`replace` uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps `a` and `b` in the given files, `file1` and `file2`:

```
shell> replace a b b a -- file1 file2 ...
```

The `replace` program is used by `msql2mysql`. See [Section 4.7.1, “msql2mysql — Convert mSQL Programs for Use with MySQL”](#).

`replace` supports the following options.

- `-?`, `-I`
Display a help message and exit.
- `-#debug_options`
Enable debugging.
- `-s`
Silent mode. Print less information what the program does.
- `-v`
Verbose mode. Print more information about what the program does.
- `-V`
Display version information and exit.

4.8.3. `resolveip` — Resolve Host name to IP Address or Vice Versa

The `resolveip` utility resolves host names to IP addresses and vice versa.

Invoke `resolveip` like this:

```
shell> resolveip [options] {host_name|ip-addr} ...
```

`resolveip` supports the following options.

- `--help [398]`, `--info [398]`, `-?`, `-I`
Display a help message and exit.

- `--silent` [399], `-s`

Silent mode. Produce less output.

- `--version` [399], `-V`

Display version information and exit.

Chapter 5. MySQL Server Administration

Table of Contents

5.1. The MySQL Server	401
5.1.1. Server Option and Variable Reference	402
5.1.2. Server Configuration Defaults	433
5.1.3. Server Command Options	433
5.1.4. Server System Variables	467
5.1.5. Using System Variables	573
5.1.6. Server Status Variables	585
5.1.7. Server SQL Modes	611
5.1.8. Server Plugins	618
5.1.9. IPv6 Support	623
5.1.10. Server-Side Help	627
5.1.11. Server Response to Signals	627
5.1.12. The Shutdown Process	628
5.2. MySQL Server Logs	629
5.2.1. Selecting General Query and Slow Query Log Output Destinations	630
5.2.2. The Error Log	632
5.2.3. The General Query Log	633
5.2.4. The Binary Log	634
5.2.5. The Slow Query Log	647
5.2.6. Server Log Maintenance	648
5.3. Running Multiple MySQL Instances on One Machine	650
5.3.1. Setting Up Multiple Data Directories	651
5.3.2. Running Multiple MySQL Instances on Windows	652
5.3.3. Running Multiple MySQL Instances on Unix	655
5.3.4. Using Client Programs in a Multiple-Server Environment	656
5.4. Tracing <code>mysqld</code> Using DTrace	657
5.4.1. <code>mysqld</code> DTrace Probe Reference	659

MySQL Server (`mysqld`) is the main program that does most of the work in a MySQL installation. This chapter provides an overview of MySQL Server and covers general server administration:

- Server configuration.
- The server log files.
- Management of multiple servers on a single machine.

For additional information on administrative topics, see also:

- [Chapter 6, Security](#)
- [Chapter 7, Backup and Recovery](#)
- [Chapter 16, Replication](#)

5.1. The MySQL Server

`mysqld` is the MySQL server. The following discussion covers these MySQL server configuration topics:

- Startup options that the server supports. You can specify these options on the command line, through configuration files, or both.

- Server system variables. These variables reflect the current state and values of the startup options, some of which can be modified while the server is running.
- Server status variables. These variables contain counters and statistics about runtime operation.
- How to set the server SQL mode. This setting modifies certain aspects of SQL syntax and semantics, for example for compatibility with code from other database systems, or to control the error handling for particular situations.
- The server shutdown process. There are performance and reliability considerations depending on the type of table (transactional or non-transactional) and whether you use replication.

**Note**

Not all storage engines are supported by all MySQL server binaries and configurations. To find out how to determine which storage engines your MySQL server installation supports, see [Section 13.7.5.17, “SHOW ENGINES Syntax”](#).

5.1.1. Server Option and Variable Reference

The following table provides a list of all the command line options, server and status variables applicable within `mysqld`.

The table lists command-line options (Cmd-line), options valid in configuration files (Option file), server system variables (System Var), and status variables (Status var) in one unified list, with notification of where each option/variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding server system or status variable, the variable name is noted immediately below the corresponding option. For status variables, the scope of the variable is shown (Scope) as either global, session, or both. Please see the corresponding sections for details on setting and using the options and variables. Where appropriate, a direct link to further information on the item as available.

For a version of this table that is specific to MySQL Cluster, see [Section 17.3.4.1, “MySQL Cluster `mysqld` Option and Variable Reference”](#).

Table 5.1. Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
abort-slave-event-count [1870]	Yes	Yes				
Aborted_clients [596]				Yes	Global	No
Aborted_connects [596]				Yes	Global	No
allow-suspicious-udfs [434]	Yes	Yes				
ansi [434]	Yes	Yes				
audit-log [764]	Yes	Yes				
audit_log_buffer_size [764]			Yes		Global	No
audit_log_file [765]			Yes		Global	No
audit_log_flush [765]			Yes		Global	Yes
audit_log_policy [766]			Yes		Global	Yes
audit_log_rotate_on_size [766]			Yes		Global	Yes
audit_log_strategy [767]			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
authentication_windows_log_level [482]	Yes	Yes				
authentication_windows_use_principal_name [483]	Yes	Yes				
auto_increment_increment [1865]	Yes	Yes	Yes		Both	Yes
auto_increment_offset [1868]	Yes	Yes	Yes		Both	Yes
autocommit [484]	Yes	Yes	Yes		Both	Yes
automatic_sp_privileges [484]			Yes		Global	Yes
back_log [485]	Yes	Yes	Yes		Global	No
basedir [435]	Yes	Yes	Yes		Global	No
big-tables [435]	Yes	Yes			Both	Yes
- Variable: big_tables			Yes		Both	Yes
bind-address [435]	Yes	Yes	Yes		Global	No
Binlog_cache_disk_use [596]				Yes	Global	No
binlog_cache_size [598]	Yes	Yes	Yes		Global	Yes
Binlog_cache_use [596]				Yes	Global	No
binlog_direct_non_transactional_updates [1898]	Yes	Yes	Yes		Both	Yes
binlog-do-db [1894]	Yes	Yes				
binlog-format [436]	Yes	Yes			Both	Yes
- Variable: binlog_format			Yes		Both	Yes
binlog-ignore-db [1896]	Yes	Yes				
binlog-row-event-max-size [1891]	Yes	Yes				
Binlog_stmt_cache_disk_use [596]				Yes	Global	No
binlog_stmt_cache_size [1901]	Yes	Yes	Yes		Global	Yes
Binlog_stmt_cache_use [597]				Yes	Global	No
bootstrap [436]	Yes	Yes				
bulk_insert_buffer_size [486]	Yes	Yes	Yes		Both	Yes
Bytes_received [597]				Yes	Both	No
Bytes_sent [597]				Yes	Both	No
character_set_client [486]			Yes		Both	Yes
character-set-client-handshake [437]	Yes	Yes				
character_set_connection [487]			Yes		Both	Yes
character_set_database [487]^a			Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
character-set-filesystem [437]	Yes	Yes			Both	Yes
- Variable: <code>character_set_filesystem</code>			Yes		Both	Yes
character_set_results [488]			Yes		Both	Yes
character-set-server [438]	Yes	Yes			Both	Yes
- Variable: <code>character_set_server</code>			Yes		Both	Yes
character_set_system [488]			Yes		Global	No
character-sets-dir [437]	Yes	Yes			Global	No
- Variable: <code>character_sets_dir</code>			Yes		Global	No
chroot [438]	Yes	Yes				
collation_connection [489]			Yes		Both	Yes
collation_database [489]^b			Yes		Both	Yes
collation-server [438]	Yes	Yes			Both	Yes
- Variable: <code>collation_server</code>			Yes		Both	Yes
Com_admin_commands [597]				Yes	Both	No
Com_alter_db [597]				Yes	Both	No
Com_alter_db_upgrade [597]				Yes	Both	No
Com_alter_event [597]				Yes	Both	No
Com_alter_function [597]				Yes	Both	No
Com_alter_procedure [597]				Yes	Both	No
Com_alter_server [597]				Yes	Both	No
Com_alter_table [597]				Yes	Both	No
Com_alter_tablespace [597]				Yes	Both	No
Com_analyze [597]				Yes	Both	No
Com_assign_to_keycache [597]				Yes	Both	No
Com_begin [597]				Yes	Both	No
Com_binlog [597]				Yes	Both	No
Com_call_procedure [597]				Yes	Both	No
Com_change_db [597]				Yes	Both	No
Com_change_master [597]				Yes	Both	No
Com_check [597]				Yes	Both	No
Com_checksum [597]				Yes	Both	No
Com_commit [597]				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_create_db [597]				Yes	Both	No
Com_create_event [597]				Yes	Both	No
Com_create_function [597]				Yes	Both	No
Com_create_index [597]				Yes	Both	No
Com_create_procedure [597]				Yes	Both	No
Com_create_server [597]				Yes	Both	No
Com_create_table [597]				Yes	Both	No
Com_create_trigger [597]				Yes	Both	No
Com_create_udf [597]				Yes	Both	No
Com_create_user [597]				Yes	Both	No
Com_create_view [597]				Yes	Both	No
Com_dealloc_sql [597]				Yes	Both	No
Com_delete [597]				Yes	Both	No
Com_delete_multi [597]				Yes	Both	No
Com_do [597]				Yes	Both	No
Com_drop_db [597]				Yes	Both	No
Com_drop_event [597]				Yes	Both	No
Com_drop_function [597]				Yes	Both	No
Com_drop_index [597]				Yes	Both	No
Com_drop_procedure [597]				Yes	Both	No
Com_drop_server [597]				Yes	Both	No
Com_drop_table [597]				Yes	Both	No
Com_drop_trigger [597]				Yes	Both	No
Com_drop_user [597]				Yes	Both	No
Com_drop_view [597]				Yes	Both	No
Com_empty_query [597]				Yes	Both	No
Com_execute_sql [597]				Yes	Both	No
Com_flush [597]				Yes	Both	No
Com_grant [597]				Yes	Both	No
Com_ha_close [597]				Yes	Both	No
Com_ha_open [597]				Yes	Both	No
Com_ha_read [597]				Yes	Both	No
Com_help [597]				Yes	Both	No
Com_insert [597]				Yes	Both	No
Com_insert_select [597]				Yes	Both	No
Com_install_plugin [597]				Yes	Both	No
Com_kill [597]				Yes	Both	No
Com_load [597]				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_lock_tables [597]				Yes	Both	No
Com_optimize [597]				Yes	Both	No
Com_preload_keys [597]				Yes	Both	No
Com_prepare_sql [597]				Yes	Both	No
Com_purge [597]				Yes	Both	No
Com_purge_before_date [597]				Yes	Both	No
Com_release_savepoint [597]				Yes	Both	No
Com_rename_table [597]				Yes	Both	No
Com_rename_user [597]				Yes	Both	No
Com_repair [597]				Yes	Both	No
Com_replace [597]				Yes	Both	No
Com_replace_select [597]				Yes	Both	No
Com_reset [597]				Yes	Both	No
Com_resignal [597]				Yes	Both	No
Com_revoke [597]				Yes	Both	No
Com_revoke_all [597]				Yes	Both	No
Com_rollback [597]				Yes	Both	No
Com_rollback_to_savepoint [597]				Yes	Both	No
Com_savepoint [597]				Yes	Both	No
Com_select [597]				Yes	Both	No
Com_set_option [597]				Yes	Both	No
Com_show_authors [597]				Yes	Both	No
Com_show_binlog_events [597]				Yes	Both	No
Com_show_binlogs [597]				Yes	Both	No
Com_show_charsets [597]				Yes	Both	No
Com_show_collations [597]				Yes	Both	No
Com_show_contributors [597]				Yes	Both	No
Com_show_create_db [597]				Yes	Both	No
Com_show_create_event [597]				Yes	Both	No
Com_show_create_func [597]				Yes	Both	No
Com_show_create_proc [597]				Yes	Both	No
Com_show_create_table [597]				Yes	Both	No
Com_show_create_trigger [597]				Yes	Both	No
Com_show_databases [597]				Yes	Both	No
Com_show_engine_logs [597]				Yes	Both	No
Com_show_engine_mutex [597]				Yes	Both	No
Com_show_engine_status [597]				Yes	Both	No
Com_show_errors [597]				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_show_events [597]				Yes	Both	No
Com_show_fields [597]				Yes	Both	No
Com_show_function_code [597]				Yes	Both	No
Com_show_function_status [597]				Yes	Both	No
Com_show_grants [597]				Yes	Both	No
Com_show_keys [597]				Yes	Both	No
Com_show_logs [597]				Yes	Both	No
Com_show_master_status [597]				Yes	Both	No
Com_show_ndb_status [597]				Yes	Both	No
Com_show_new_master [597]				Yes	Both	No
Com_show_open_tables [597]				Yes	Both	No
Com_show_plugins [597]				Yes	Both	No
Com_show_privileges [597]				Yes	Both	No
Com_show_procedure_code [597]				Yes	Both	No
Com_show_procedure_status [597]				Yes	Both	No
Com_show_processlist [597]				Yes	Both	No
Com_show_profile [597]				Yes	Both	No
Com_show_profiles [597]				Yes	Both	No
Com_show_relaylog_events [597]				Yes	Both	No
Com_show_slave_hosts [597]				Yes	Both	No
Com_show_slave_status [597]				Yes	Both	No
Com_show_status [597]				Yes	Both	No
Com_show_storage_engines [597]				Yes	Both	No
Com_show_table_status [597]				Yes	Both	No
Com_show_tables [597]				Yes	Both	No
Com_show_triggers [597]				Yes	Both	No
Com_show_variables [597]				Yes	Both	No
Com_show_warnings [597]				Yes	Both	No
Com_signal [597]				Yes	Both	No
Com_slave_start [597]				Yes	Both	No
Com_slave_stop [597]				Yes	Both	No
Com_stmt_close [597]				Yes	Both	No
Com_stmt_execute [597]				Yes	Both	No
Com_stmt_fetch [597]				Yes	Both	No
Com_stmt_prepare [597]				Yes	Both	No
Com_stmt_reprepare [597]				Yes	Both	No
Com_stmt_reset [597]				Yes	Both	No
Com_stmt_send_long_data [597]				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_truncate [597]				Yes	Both	No
Com_uninstall_plugin [597]				Yes	Both	No
Com_unlock_tables [597]				Yes	Both	No
Com_update [597]				Yes	Both	No
Com_update_multi [597]				Yes	Both	No
Com_xa_commit [597]				Yes	Both	No
Com_xa_end [597]				Yes	Both	No
Com_xa_prepare [597]				Yes	Both	No
Com_xa_recover [597]				Yes	Both	No
Com_xa_rollback [597]				Yes	Both	No
Com_xa_start [597]				Yes	Both	No
completion_type [489]	Yes	Yes	Yes		Both	Yes
Compression [597]				Yes	Session	No
concurrent_insert [490]	Yes	Yes	Yes		Global	Yes
connect_timeout [492]	Yes	Yes	Yes		Global	Yes
Connections [598]				Yes	Global	No
console [438]	Yes	Yes				
core-file [439]	Yes	Yes				
Created_tmp_disk_tables [598]				Yes	Both	No
Created_tmp_files [598]				Yes	Global	No
Created_tmp_tables [598]				Yes	Both	No
datadir [439]	Yes	Yes	Yes		Global	No
date_format [492]			Yes		Global	No
datetime_format [492]			Yes		Global	No
debug [439]	Yes	Yes	Yes		Both	Yes
debug_sync [493]			Yes		Session	Yes
debug-sync-timeout [440]	Yes	Yes				
default-storage-engine [441]	Yes	Yes			Both	Yes
- Variable: default_storage_engine			Yes		Both	Yes
default-time-zone [441]	Yes	Yes				
default_week_format [494]	Yes	Yes	Yes		Both	Yes
defaults-extra-file [248]	Yes					
defaults-file [248]	Yes					

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
defaults-group-suffix [248]	Yes					
delay-key-write [441]	Yes	Yes			Global	Yes
- Variable: delay_key_write			Yes		Global	Yes
Delayed_errors [598]				Yes	Global	No
delayed_insert_limit [495]	Yes	Yes	Yes		Global	Yes
Delayed_insert_threads [598]				Yes	Global	No
delayed_insert_time [496]	Yes	Yes	Yes		Global	Yes
delayed_queue_size [496]	Yes	Yes	Yes		Global	Yes
Delayed_writes [598]				Yes	Global	No
des-key-file [442]	Yes	Yes				
disconnect-slave-event-count [1870]	Yes	Yes				
div_precision_increment [497]	Yes	Yes	Yes		Both	Yes
enable-locking [442]	Yes	Yes				
enable-named-pipe [442]	Yes	Yes				
- Variable: named_pipe						
enable-pstack [442]	Yes	Yes				
engine-condition-pushdown [443]	Yes	Yes			Both	Yes
- Variable: engine_condition_pushdown			Yes		Both	Yes
error_count [498]			Yes		Session	No
event-scheduler [443]	Yes	Yes			Global	Yes
- Variable: event_scheduler			Yes		Global	Yes
exit-info [443]	Yes	Yes				
expire_logs_days [499]	Yes	Yes	Yes		Global	Yes
external-locking [444]	Yes	Yes				
- Variable: skip_external_locking						
external_user [499]			Yes		Session	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
federated	Yes	Yes				
flush [499]	Yes	Yes	Yes		Global	Yes
Flush_commands [598]				Yes	Global	No
flush_time [500]	Yes	Yes	Yes		Global	Yes
foreign_key_checks [500]			Yes		Both	Yes
ft_boolean_syntax [500]	Yes	Yes	Yes		Global	Yes
ft_max_word_len [500]	Yes	Yes	Yes		Global	No
ft_min_word_len [500]	Yes	Yes	Yes		Global	No
ft_query_expansion_limit [502]	Yes	Yes	Yes		Global	No
ft_stopword_file [502]	Yes	Yes	Yes		Global	No
gdb [444]	Yes	Yes				
general-log [445]	Yes	Yes			Global	Yes
- Variable: general_log			Yes		Global	Yes
general_log_file [500]	Yes	Yes	Yes		Global	Yes
group_concat_max_len [503]	Yes	Yes	Yes		Both	Yes
Handler_commit [598]				Yes	Both	No
Handler_delete [599]				Yes	Both	No
Handler_discover [2161]				Yes	Both	No
Handler_prepare [599]				Yes	Both	No
Handler_read_first [599]				Yes	Both	No
Handler_read_key [599]				Yes	Both	No
Handler_read_last [599]				Yes	Both	No
Handler_read_next [599]				Yes	Both	No
Handler_read_prev [599]				Yes	Both	No
Handler_read_rnd [599]				Yes	Both	No
Handler_read_rnd_next [599]				Yes	Both	No
Handler_rollback [599]				Yes	Both	No
Handler_savepoint [599]				Yes	Both	No
Handler_savepoint_rollback [599]				Yes	Both	No
Handler_update [600]				Yes	Both	No
Handler_write [600]				Yes	Both	No
have_compress [504]			Yes		Global	No
have_crypt [504]			Yes		Global	No
have_csv [504]			Yes		Global	No
have_dynamic_loading [504]			Yes		Global	No
have_geometry [504]			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
have_innodb [504]			Yes		Global	No
have_ndbcluster [2147]			Yes		Global	No
have_openssl [504]			Yes		Global	No
have_partitioning [504]			Yes		Global	No
have_profiling [504]			Yes		Global	No
have_query_cache [505]			Yes		Global	No
have_rtree_keys [505]			Yes		Global	No
have_ssl [505]			Yes		Global	No
have_symlink [505]			Yes		Global	No
help [434]	Yes	Yes				
hostname [505]			Yes		Global	No
identity [505]			Yes		Session	Yes
ignore-builtin-innodb [1546]	Yes	Yes			Global	No
- Variable: ignore_builtin_innodb			Yes		Global	No
init_connect [505]	Yes	Yes	Yes		Global	Yes
init-file [445]	Yes	Yes			Global	No
- Variable: init_file			Yes		Global	No
init-rpl-role	Yes	Yes				
init_slave [1885]	Yes	Yes	Yes		Global	Yes
innodb [1546]	Yes	Yes				
innodb_adaptive_flushing [1547]	Yes	Yes	Yes		Global	Yes
innodb_adaptive_hash_index [1547]	Yes	Yes	Yes		Global	Yes
innodb_additional_mem_pool_size [1548]	Yes	Yes	Yes		Global	No
innodb_autoextend_increment [1548]	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode [1548]	Yes	Yes	Yes		Global	No
innodb_buffer_pool_instances [1548]	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_pages_data [600]				Yes	Global	No
Innodb_buffer_pool_pages_dirty [600]				Yes	Global	No
Innodb_buffer_pool_pages_flushed [600]				Yes	Global	No
Innodb_buffer_pool_pages_free [600]				Yes	Global	No
Innodb_buffer_pool_pages_latched [600]				Yes	Global	No
Innodb_buffer_pool_pages_misc [600]				Yes	Global	No
Innodb_buffer_pool_pages_total [600]				Yes	Global	No
Innodb_buffer_pool_read_ahead [600]				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted [600]				Yes	Global	No
Innodb_buffer_pool_read_requests [600]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
InnoDB_buffer_pool_reads [600]				Yes	Global	No
innodb_buffer_pool_size [1549]	Yes	Yes	Yes		Global	No
InnoDB_buffer_pool_wait_free [601]				Yes	Global	No
InnoDB_buffer_pool_write_requests [601]				Yes	Global	No
innodb_change_buffering [1550]	Yes	Yes	Yes		Global	Yes
innodb_checksums [1551]	Yes	Yes	Yes		Global	No
innodb_commit_concurrency [1552]	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets [1552]	Yes	Yes	Yes		Global	Yes
innodb_data_file_path [1552]	Yes	Yes	Yes		Global	No
InnoDB_data_fsyncs [601]				Yes	Global	No
innodb_data_home_dir [1552]	Yes	Yes	Yes		Global	No
InnoDB_data_pending_fsyncs [601]				Yes	Global	No
InnoDB_data_pending_reads [601]				Yes	Global	No
InnoDB_data_pending_writes [601]				Yes	Global	No
InnoDB_data_read [601]				Yes	Global	No
InnoDB_data_reads [601]				Yes	Global	No
InnoDB_data_writes [601]				Yes	Global	No
InnoDB_data_written [601]				Yes	Global	No
InnoDB_dblwr_pages_written [601]				Yes	Global	No
InnoDB_dblwr_writes [601]				Yes	Global	No
innodb_doublewrite [1553]	Yes	Yes	Yes		Global	No
innodb_fast_shutdown [1553]	Yes	Yes	Yes		Global	Yes
innodb_file_format [1554]	Yes	Yes	Yes		Global	Yes
innodb_file_format_check [1554]	Yes	Yes	Yes		Global	No
innodb_file_format_max [1555]	Yes	Yes	Yes		Global	Yes
innodb_file_per_table [1555]	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_commit [1556]	Yes	Yes	Yes		Global	Yes
innodb_flush_method [1557]	Yes	Yes	Yes		Global	No
innodb_force_load_corrupted [1558]	Yes	Yes	Yes		Global	No
innodb_force_recovery [1558]	Yes	Yes	Yes		Global	No
InnoDB_have_atomic_builtins [601]				Yes	Global	No
innodb_io_capacity [1559]	Yes	Yes	Yes		Global	Yes
innodb_large_prefix [1560]	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout [1560]	Yes	Yes	Yes		Both	Yes
innodb_locks_unsafe_for_binlog [1561]	Yes	Yes	Yes		Global	No
innodb_log_buffer_size [1563]	Yes	Yes	Yes		Global	No
innodb_log_file_size [1564]	Yes	Yes	Yes		Global	No
innodb_log_files_in_group [1564]	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_log_group_home_dir [1564]	Yes	Yes	Yes		Global	No
Innodb_log_waits [602]				Yes	Global	No
Innodb_log_write_requests [602]				Yes	Global	No
Innodb_log_writes [602]				Yes	Global	No
innodb_max_dirty_pages_pct [1565]	Yes	Yes	Yes		Global	Yes
innodb_max_purge_size [1565]	Yes	Yes	Yes		Global	Yes
innodb_mirrored_log_groups [1566]	Yes	Yes	Yes		Global	No
innodb_old_blocks_size [1566]	Yes	Yes	Yes		Global	Yes
innodb_old_blocks_time [1566]	Yes	Yes	Yes		Global	Yes
innodb_open_files [1567]	Yes	Yes	Yes		Global	No
Innodb_os_log_fsyncls [602]				Yes	Global	No
Innodb_os_log_pending_fsyncls [602]				Yes	Global	No
Innodb_os_log_pending_writes [602]				Yes	Global	No
Innodb_os_log_written [602]				Yes	Global	No
Innodb_page_size [602]				Yes	Global	No
Innodb_pages_created [602]				Yes	Global	No
Innodb_pages_read [602]				Yes	Global	No
Innodb_pages_written [602]				Yes	Global	No
innodb_purge_batch_size [1567]	Yes	Yes	Yes		Global	No
innodb_purge_threads [1567]	Yes	Yes	Yes		Global	No
innodb_read_ahead_threshold [1568]	Yes	Yes	Yes		Global	Yes
innodb_read_io_threads [1568]	Yes	Yes	Yes		Global	No
innodb_replication_delay [1569]	Yes	Yes	Yes		Global	Yes
innodb_rollback_on_timeout [1569]	Yes	Yes	Yes		Global	No
innodb_rollback_segments [1570]	Yes	Yes	Yes		Global	Yes
Innodb_row_lock_current_waits [602]				Yes	Global	No
Innodb_row_lock_time [602]				Yes	Global	No
Innodb_row_lock_time_avg [602]				Yes	Global	No
Innodb_row_lock_time_max [602]				Yes	Global	No
Innodb_row_lock_waits [603]				Yes	Global	No
Innodb_rows_deleted [603]				Yes	Global	No
Innodb_rows_inserted [603]				Yes	Global	No
Innodb_rows_read [603]				Yes	Global	No
Innodb_rows_updated [603]				Yes	Global	No
innodb_spin_wait_delay [1570]	Yes	Yes	Yes		Global	Yes
innodb_stats_method [1570]	Yes	Yes	Yes		Global	Yes
innodb_stats_on_metadata [1571]	Yes	Yes	Yes		Global	Yes
innodb_stats_sample_pages [1571]	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb-status-file [1546]	Yes	Yes				
innodb_strict_mode [1572]	Yes	Yes	Yes		Both	Yes
innodb_support_xa [1572]	Yes	Yes	Yes		Both	Yes
innodb_sync_spin_mutex [1573]	Yes	Yes	Yes		Global	Yes
innodb_table_locks [1573]	Yes	Yes	Yes		Both	Yes
innodb_thread_concurrency [1573]	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay [1574]	Yes	Yes	Yes		Global	Yes
Innodb_truncated_status_writes [603]				Yes	Global	No
innodb_use_native_aio [1574]	Yes	Yes	Yes		Global	No
innodb_use_sys_malloc [1575]	Yes	Yes	Yes		Global	No
innodb_version [1575]			Yes		Global	No
innodb_write_io_threads [1575]	Yes	Yes	Yes		Global	No
insert_id [506]			Yes		Session	Yes
install	Yes					
install-manual	Yes					
interactive_timeout [506]	Yes	Yes	Yes		Both	Yes
join_buffer_size [507]	Yes	Yes	Yes		Both	Yes
keep_files_on_create [507]	Yes	Yes	Yes		Both	Yes
Key_blocks_not_flushed [603]				Yes	Global	No
Key_blocks_unused [603]				Yes	Global	No
Key_blocks_used [603]				Yes	Global	No
key_buffer_size [508]	Yes	Yes	Yes		Global	Yes
key_cache_age_threshold [509]	Yes	Yes	Yes		Global	Yes
key_cache_block_size [509]	Yes	Yes	Yes		Global	Yes
key_cache_division_limit [510]	Yes	Yes	Yes		Global	Yes
Key_read_requests [603]				Yes	Global	No
Key_reads [603]				Yes	Global	No
Key_write_requests [603]				Yes	Global	No
Key_writes [603]				Yes	Global	No
language [446]	Yes	Yes	Yes		Global	No
large_files_support [511]			Yes		Global	No
large_page_size [511]			Yes		Global	No
large-pages [446]	Yes	Yes			Global	No
- Variable: large_pages			Yes		Global	No
last_insert_id [511]			Yes		Session	Yes
Last_query_cost [603]				Yes	Session	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
lc-messages [447]	Yes	Yes			Both	Yes
- Variable: lc_messages			Yes		Both	Yes
lc-messages-dir [447]	Yes	Yes			Global	No
- Variable: lc_messages_dir			Yes		Global	No
lc_time_names [512]			Yes		Both	Yes
license [513]			Yes		Global	No
local_infile [513]			Yes		Global	Yes
lock_wait_timeout [513]	Yes	Yes	Yes		Both	Yes
locked_in_memory [514]			Yes		Global	No
log [447]	Yes	Yes	Yes		Global	Yes
log_bin [514]			Yes		Global	No
log-bin [1892]	Yes	Yes	Yes		Global	No
log-bin-index [1892]	Yes	Yes				
log-bin-trust-function-creators [1892]	Yes	Yes			Global	Yes
- Variable: log_bin_trust_function_creators			Yes		Global	Yes
log-bin-trust-routine-creators	Yes	Yes			Global	Yes
- Variable: log_bin_trust_routine_creators			Yes		Global	Yes
log_bin_use_v1_row_events [1901]	Yes	Yes	Yes		Global	No
log-bin-use-v1-row-events [1893]	Yes	Yes			Global	No
- Variable: log_bin_use_v1_row_events			Yes		Global	No
log-error [448]	Yes	Yes			Global	No
- Variable: log_error			Yes		Global	No
log-isam [448]	Yes	Yes				
log-output [449]	Yes	Yes			Global	Yes
- Variable: log_output			Yes		Global	Yes
log-queries-not-using-indexes [449]	Yes	Yes			Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
- Variable: log_queries_not_using_indexes			Yes		Global	Yes
log-short-format [450]	Yes	Yes				
log-slave-updates [1870]	Yes	Yes			Global	No
- Variable: log_slave_updates			Yes		Global	No
log_slave_updates [457]	Yes	Yes	Yes		Global	No
log-slow-admin-statements [450]	Yes	Yes				
log-slow-queries [450]	Yes	Yes			Global	Yes
- Variable: log_slow_queries			Yes		Global	Yes
log-slow-slave-statements [1871]	Yes	Yes				
log-tc [450]	Yes	Yes				
log-tc-size [451]	Yes	Yes				
log-warnings [451]	Yes	Yes			Both	Yes
- Variable: log_warnings			Yes		Both	Yes
long_query_time [517]	Yes	Yes	Yes		Both	Yes
low-priority-updates [452]	Yes	Yes			Both	Yes
- Variable: low_priority_updates			Yes		Both	Yes
lower_case_file_system [518]	Yes	Yes	Yes		Global	No
lower_case_table_names [518]	Yes	Yes	Yes		Global	No
master-connect-retry [1884]	Yes	Yes				
master-host [1884]	Yes	Yes				
master-info-file [1872]	Yes	Yes				
master-password [1884]	Yes	Yes				
master-port [1884]	Yes	Yes				
master-retry-count [1872]	Yes	Yes				
master-ssl [1885]	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
master-ssl-ca [1885]	Yes	Yes				
master-ssl-capath [1885]	Yes	Yes				
master-ssl-cert [1885]	Yes	Yes				
master-ssl-cipher [1885]	Yes	Yes				
master-ssl-key [1885]	Yes	Yes				
master-user [1884]	Yes	Yes				
max_allowed_packet [519]	Yes	Yes	Yes		Global	Yes
max_binlog_cache_size [1902]	Yes	Yes	Yes		Global	Yes
max-binlog-dump-events [1897]	Yes	Yes				
max_binlog_size [1903]	Yes	Yes	Yes		Global	Yes
max_binlog_stmt_cache_size [1904]	Yes	Yes	Yes		Global	Yes
max_connect_errors [519]	Yes	Yes	Yes		Global	Yes
max_connections [519]	Yes	Yes	Yes		Global	Yes
max_delayed_threads [520]	Yes	Yes	Yes		Both	Yes
max_error_count [520]	Yes	Yes	Yes		Both	Yes
max_heap_table_size [521]	Yes	Yes	Yes		Both	Yes
max_insert_delayed_threads [521]			Yes		Both	Yes
max_join_size [522]	Yes	Yes	Yes		Both	Yes
max_length_for_sort_data [522]	Yes	Yes	Yes		Both	Yes
max_long_data_size [523]	Yes	Yes	Yes		Global	No
max_prepared_stmt_count [523]	Yes	Yes	Yes		Global	Yes
max_relay_log_size [523]	Yes	Yes	Yes		Global	Yes
max_seeks_for_key [524]	Yes	Yes	Yes		Both	Yes
max_sort_length [524]	Yes	Yes	Yes		Both	Yes
max_sp_recursion_depth [525]	Yes	Yes	Yes		Both	Yes
Max_used_connections [604]				Yes	Global	No
max_user_connections [525]	Yes	Yes	Yes		Both	Yes
max_write_lock_count [526]	Yes	Yes	Yes		Global	Yes
memlock [453]	Yes	Yes	Yes		Global	No
metadata_locks_cache_size [526]			Yes		Global	No
min-examined-row-limit [452]	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
myisam-block-size [454]	Yes	Yes				
myisam_data_pointer_size [527]	Yes	Yes	Yes		Global	Yes
myisam_max_sort_size [528]	Yes	Yes	Yes		Global	Yes
myisam_mmap_size [528]	Yes	Yes	Yes		Global	No
myisam-recover [454]	Yes	Yes				
- Variable: myisam_recover_options						
myisam-recover-options [454]	Yes	Yes				
- Variable: myisam_recover_options						
myisam_recover_options [529]			Yes		Global	No
myisam_repair_threads [529]	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size [530]	Yes	Yes	Yes		Both	Yes
myisam_stats_method [530]	Yes	Yes	Yes		Both	Yes
myisam_use_mmap [531]	Yes	Yes	Yes		Global	Yes
named_pipe [531]			Yes		Global	No
Ndb_api_bytes_received_count [2164]				Yes	Global	No
Ndb_api_bytes_received_count_session [2164]				Yes	Session	No
Ndb_api_bytes_received_count_slave [2164]				Yes	Global	No
Ndb_api_bytes_sent_count [2164]				Yes	Global	No
Ndb_api_bytes_sent_count_session [2163]				Yes	Session	No
Ndb_api_bytes_sent_count_slave [2164]				Yes	Global	No
Ndb_api_event_bytes_count [2171]				Yes	Global	No
Ndb_api_event_bytes_count_injector [2171]				Yes	Global	No
Ndb_api_event_data_count [2171]				Yes	Global	No
Ndb_api_event_data_count_injector [2171]				Yes	Global	No
Ndb_api_event_nondata_count [2171]				Yes	Global	No
Ndb_api_event_nondata_count_injector [2171]				Yes	Global	No
Ndb_api_pk_op_count [2167]				Yes	Global	No
Ndb_api_pk_op_count_session [2166]				Yes	Session	No
Ndb_api_pk_op_count_slave [2167]				Yes	Global	No
Ndb_api_pruned_scan_count [2169]				Yes	Global	No
Ndb_api_pruned_scan_count_session [2168]				Yes	Session	No
Ndb_api_pruned_scan_count_slave [2169]				Yes	Global	No
Ndb_api_range_scan_count [2168]				Yes	Global	No
Ndb_api_range_scan_count_session [2168]				Yes	Session	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Ndb_api_range_scan_count_slave [2168]				Yes	Global	No
Ndb_api_read_row_count [2170]				Yes	Global	No
Ndb_api_read_row_count_session [2169]				Yes	Session	No
Ndb_api_read_row_count_slave [2170]				Yes	Global	No
Ndb_api_scan_batch_count [2169]				Yes	Global	No
Ndb_api_scan_batch_count_session [2169]				Yes	Session	No
Ndb_api_scan_batch_count_slave [2169]				Yes	Global	No
Ndb_api_table_scan_count [2168]				Yes	Global	No
Ndb_api_table_scan_count_session [2167]				Yes	Session	No
Ndb_api_table_scan_count_slave [2168]				Yes	Global	No
Ndb_api_trans_abort_count [2166]				Yes	Global	No
Ndb_api_trans_abort_count_session [2165]				Yes	Session	No
Ndb_api_trans_abort_count_slave [2165]				Yes	Global	No
Ndb_api_trans_close_count [2166]				Yes	Global	No
Ndb_api_trans_close_count_session [2166]				Yes	Session	No
Ndb_api_trans_close_count_slave [2166]				Yes	Global	No
Ndb_api_trans_commit_count [2165]				Yes	Global	No
Ndb_api_trans_commit_count_session [2165]				Yes	Session	No
Ndb_api_trans_commit_count_slave [2165]				Yes	Global	No
Ndb_api_trans_local_read_row_count [2170]				Yes	Global	No
Ndb_api_trans_local_read_row_count_session [2170]				Yes	Session	No
Ndb_api_trans_local_read_row_count_slave [2170]				Yes	Global	No
Ndb_api_trans_start_count [2165]				Yes	Global	No
Ndb_api_trans_start_count_session [2164]				Yes	Session	No
Ndb_api_trans_start_count_slave [2165]				Yes	Global	No
Ndb_api_uk_op_count [2167]				Yes	Global	No
Ndb_api_uk_op_count_session [2167]				Yes	Session	No
Ndb_api_uk_op_count_slave [2167]				Yes	Global	No
Ndb_api_wait_exec_complete_count [2162]				Yes	Global	No
Ndb_api_wait_exec_complete_count_session [2161]				Yes	Session	No
Ndb_api_wait_exec_complete_count_slave [2161]				Yes	Global	No
Ndb_api_wait_meta_request_count [2163]				Yes	Global	No
Ndb_api_wait_meta_request_count_session [2162]				Yes	Session	No
Ndb_api_wait_meta_request_count_slave [2162]				Yes	Global	No
Ndb_api_wait_nanos_count [2163]				Yes	Global	No
Ndb_api_wait_nanos_count_session [2163]				Yes	Session	No
Ndb_api_wait_nanos_count_slave [2163]				Yes	Global	No
Ndb_api_wait_scan_result_count [2162]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Ndb_api_wait_scan_result_count	session [2162]			Yes	Session	No
Ndb_api_wait_scan_result_count	slave [2162]			Yes	Global	No
ndb_autoincrement	Yes	fetch_sz [2163]	Yes		Both	Yes
ndb-batch-size [2138]	Yes	Yes	Yes		Global	No
ndb-blob-read-batch-bytes [2140]	Yes	Yes	Yes		Both	Yes
ndb-blob-write-batch-bytes [2140]	Yes	Yes	Yes		Both	Yes
ndb_cache_check	Yes [2149]	Yes	Yes		Global	Yes
ndb-cluster-connection-pool [2139]	Yes	Yes	Yes	Yes	Global	No
Ndb_cluster_node_id [2171]				Yes	Both	No
Ndb_config_from_host [2172]				Yes	Both	No
Ndb_conflict_fn_epoch [2172]				Yes	Global	No
Ndb_conflict_fn_epoch_trans [2172]				Yes	Global	No
Ndb_conflict_fn_max [2172]				Yes	Global	No
Ndb_conflict_fn_old [2172]				Yes	Global	No
Ndb_conflict_trans_conflict_commit_count [2173]				Yes	Global	No
Ndb_conflict_trans_detect_iter_count [2173]				Yes	Global	No
Ndb_conflict_trans_reject_count [2173]				Yes	Global	No
Ndb_conflict_trans_row_conflict_count [2172]				Yes	Global	No
Ndb_conflict_trans_row_reject_count [2173]				Yes	Global	No
ndb-connectstring [2141]	Yes	Yes				
ndb-deferred-constraints [2141]	Yes	Yes			Both	Yes
- Variable: ndb_deferred_constraints			Yes		Both	Yes
ndb_deferred_constraints [2149]	Yes	Yes	Yes		Both	Yes
ndb_distribution [2136]	Yes	Yes	Yes		Both	Yes
ndb-distribution [2141]	Yes	Yes			Both	Yes
- Variable: ndb_distribution			Yes		Both	Yes
ndb_execute_count [2173]				Yes	Global	No
ndb_extra_logging [2150]	Yes	Yes	Yes		Global	Yes
ndb_force_send [2165]	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
ndb_join_pushdown [2151]			Yes		Global	No
ndb-log-apply-status [2142]	Yes	Yes			Global	No
- Variable: ndb_log_apply_status			Yes		Global	No
ndb_log_bin [2153]	Yes		Yes		Both	Yes
ndb_log_binlog_index [2153]	Yes		Yes		Global	Yes
ndb_log_empty_epochs [2160]	Yes	Yes	Yes		Global	Yes
ndb-log-empty-epochs [2144]	Yes	Yes	Yes		Global	Yes
ndb_log_orig			Yes		Global	No
ndb-log-transaction-id [2143]	Yes	Yes			Global	No
- Variable: ndb_log_transaction_id			Yes		Global	No
ndb_log_transaction_id [2154]			Yes		Global	No
ndb-log-update-as-write [2373]	Yes	Yes	Yes		Global	Yes
ndb_log_updated_ops [2372]	Yes	Yes	Yes		Global	Yes
ndb-mgmd-host [2142]	Yes	Yes				
ndb-nodeid [2143]	Yes	Yes		Yes	Global	No
Ndb_number_of_data_nodes [2173]				Yes	Global	No
ndb_optimization_delay [2147]			Yes		Global	Yes
ndb_optimized_node_selection [2173]	Yes	Yes				
ndb_pruned_scan_count [2174]				Yes	Global	No
Ndb_pushed_queries_defined [2173]				Yes	Global	No
Ndb_pushed_queries_dropped [2173]				Yes	Global	No
Ndb_pushed_queries_executed [2174]				Yes	Global	No
ndb_pushed_reads [2174]				Yes	Global	No
ndb_report_thresh_log_epoch [2155]	Yes	Yes				
ndb_report_thresh_log_mem_usage [2155]	Yes	Yes				
ndb_scan_count [2174]				Yes	Global	No
ndb_table_no_logging [2156]			Yes		Session	Yes
ndb_table_temporary [2157]			Yes		Session	Yes
ndb-transid-mysql-connection-map [2145]	Yes					
ndb_use_copying_alter_table [2157]			Yes		Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
ndb_use_exact_count [2157]			Yes		Both	Yes
ndb_use_transactions [2157]	Yes	Yes	Yes		Both	Yes
ndb-wait-connected [2146]	Yes	Yes	Yes		Global	No
ndb-wait-setup [2146]	Yes	Yes	Yes		Global	No
ndbcluster [2142]	Yes	Yes				
- Variable: have_ndbcluster						
ndbinfo_database [2158]			Yes		Global	No
ndbinfo_max_bytes [2159]	Yes		Yes		Both	Yes
ndbinfo_max_rows [2159]	Yes		Yes		Both	Yes
ndbinfo_show_hidden [2159]	Yes		Yes		Both	Yes
ndbinfo_table_prefix [2160]	Yes		Yes		Both	Yes
ndbinfo_version [2160]			Yes		Global	No
net_buffer_length [504]	Yes	Yes	Yes		Both	Yes
net_read_timeout [504]	Yes	Yes	Yes		Both	Yes
net_retry_count [537]	Yes	Yes	Yes		Both	Yes
net_write_timeout [504]	Yes	Yes	Yes		Both	Yes
new [533]	Yes	Yes	Yes		Both	Yes
no-defaults [248]	Yes					
Not_flushed_delayed_rows [604]				Yes	Global	No
old [534]	Yes	Yes	Yes		Global	No
old-alter-table [455]	Yes	Yes			Both	Yes
- Variable: old_alter_table			Yes		Both	Yes
old_passwords [534]			Yes		Both	Yes
old-style-user-limits [455]	Yes	Yes				
one-thread [456]	Yes	Yes				
Open_files [604]				Yes	Global	No
open-files-limit [456]	Yes	Yes			Global	No
- Variable: open_files_limit			Yes		Global	No
Open_streams [604]				Yes	Global	No
Open_table_definitions [604]				Yes	Global	No
Open_tables [604]				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Opened_files [604]				Yes	Global	No
Opened_table_definitions [604]				Yes	Both	No
Opened_tables [604]				Yes	Both	No
optimizer_prune_level [535]	Yes	Yes	Yes		Both	Yes
optimizer_search_depth [536]	Yes	Yes	Yes		Both	Yes
optimizer_switch [539]	Yes	Yes	Yes		Both	Yes
partition [456]	Yes	Yes			Global	No
- Variable: have_partitioning			Yes		Global	No
performance_schema [2554]	Yes	Yes	Yes		Global	No
Performance_schema_cond_classes_lost [2559]				Yes	Global	No
Performance_schema_cond_instances_lost [2559]				Yes	Global	No
performance_schema_events_wait_history_long_size [2554]	Yes	Yes	Yes		Global	No
performance_schema_events_wait_history_size [2554]	Yes	Yes	Yes		Global	No
Performance_schema_file_classes_lost [2559]				Yes	Global	No
Performance_schema_file_handles_lost [2559]				Yes	Global	No
Performance_schema_file_instances_lost [2559]				Yes	Global	No
Performance_schema_locker_lost [2559]				Yes	Global	No
performance_schema_max_cond_classes [2554]	Yes	Yes	Yes		Global	No
performance_schema_max_cond_instances [2556]	Yes	Yes	Yes		Global	No
performance_schema_max_file_classes [2555]	Yes	Yes	Yes		Global	No
performance_schema_max_file_handles [2555]	Yes	Yes	Yes		Global	No
performance_schema_max_file_instances [2556]	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_classes [2557]	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_instances [2556]	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_classes [2557]	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_instances [2556]	Yes	Yes	Yes		Global	No
performance_schema_max_table_handles [2557]	Yes	Yes	Yes		Global	No
performance_schema_max_table_instances [2556]	Yes	Yes	Yes		Global	No
performance_schema_max_thread_classes [2557]	Yes	Yes	Yes		Global	No
performance_schema_max_thread_instances [2556]	Yes	Yes	Yes		Global	No
Performance_schema_mutex_classes_lost [2559]				Yes	Global	No
Performance_schema_mutex_instances_lost [2559]				Yes	Global	No
Performance_schema_rwlock_classes_lost [2559]				Yes	Global	No
Performance_schema_rwlock_instances_lost [2559]				Yes	Global	No
Performance_schema_table_handles_lost [2559]				Yes	Global	No
Performance_schema_table_instances_lost [2559]				Yes	Global	No
Performance_schema_thread_classes_lost [2559]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Performance_schema_thread_instances_lost [2559]				Yes	Global	No
pid-file [456]	Yes	Yes			Global	No
- Variable: pid_file			Yes		Global	No
plugin [457]	Yes	Yes				
plugin_dir [537]	Yes	Yes	Yes		Global	No
plugin-load [457]	Yes	Yes				
port [458]	Yes	Yes	Yes		Global	No
port-open-timeout [458]	Yes	Yes				
preload_buffer_size [538]	Yes	Yes	Yes		Both	Yes
Prepared_stmt_count [604]				Yes	Global	No
print-defaults [248]	Yes					
profiling [539]			Yes		Both	Yes
profiling_history_size [539]	Yes	Yes	Yes		Both	Yes
protocol_version [539]			Yes		Global	No
proxy_user [539]			Yes		Session	No
pseudo_thread_id [539]			Yes		Session	Yes
Qcache_free_blocks [604]				Yes	Global	No
Qcache_free_memory [605]				Yes	Global	No
Qcache_hits [605]				Yes	Global	No
Qcache_inserts [605]				Yes	Global	No
Qcache_lowmem_prunes [605]				Yes	Global	No
Qcache_not_cached [605]				Yes	Global	No
Qcache_queries_in_cache [605]				Yes	Global	No
Qcache_total_blocks [605]				Yes	Global	No
Queries [605]				Yes	Both	No
query_alloc_block_size [539]	Yes	Yes	Yes		Both	Yes
query_cache_limit [543]	Yes	Yes	Yes		Global	Yes
query_cache_min_size [541]	Yes	Yes	Yes		Global	Yes
query_cache_size [543]	Yes	Yes	Yes		Global	Yes
query_cache_type [543]	Yes	Yes	Yes		Both	Yes
query_cache_wlock_invalidate [542]	Yes	Yes	Yes		Both	Yes
query_prealloc_size [543]	Yes	Yes	Yes		Both	Yes
Questions [605]				Yes	Both	No
rand_seed1 [543]			Yes		Session	Yes
rand_seed2 [543]			Yes		Session	Yes
range_alloc_block_size [544]	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
read_buffer_size [544]	Yes	Yes	Yes		Both	Yes
read_only [544]	Yes	Yes	Yes		Global	Yes
read_rnd_buffer_size [545]	Yes	Yes	Yes		Both	Yes
relay-log [1873]	Yes	Yes			Global	No
- Variable: relay_log			Yes		Global	No
relay-log-index [1874]	Yes	Yes			Global	No
- Variable: relay_log_index			Yes		Global	No
relay_log_index [1875]	Yes	Yes	Yes		Global	No
relay-log-info-file [1874]	Yes	Yes				
- Variable: relay_log_info_file						
relay_log_info_file [1875]	Yes	Yes	Yes		Global	No
relay_log_purge [546]	Yes	Yes	Yes		Global	Yes
relay_log_recovery [1876]	Yes	Yes	Yes		Global	No
relay-log-recovery [1875]	Yes	Yes				
- Variable: relay_log_recovery						
relay_log_space_limit [546]	Yes	Yes	Yes		Global	No
remove	Yes					
replicate-do-db [1875]	Yes	Yes				
replicate-do-table [1878]	Yes	Yes				
replicate-ignore-db [1877]	Yes	Yes				
replicate-ignore-table [1878]	Yes	Yes				
replicate-rewrite-db [1878]	Yes	Yes				
replicate-same-server-id [1879]	Yes	Yes				
replicate-wild-do-table [1879]	Yes	Yes				
replicate-wild-ignore-table [1880]	Yes	Yes				
report-host [1880]	Yes	Yes			Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
- Variable: report_host			Yes		Global	No
report- password [1880]	Yes	Yes			Global	No
- Variable: report_password			Yes		Global	No
report-port [1881]	Yes	Yes			Global	No
- Variable: report_port			Yes		Global	No
report- user [1881]	Yes	Yes			Global	No
- Variable: report_user			Yes		Global	No
rpl_recovery_rank [1887]			Yes		Global	Yes
Rpl_semi_sync_master_clients [605]				Yes	Global	No
rpl_semi_sync_master_enabled [548]			Yes		Global	Yes
Rpl_semi_sync_master_net_avg_wait_time [605]				Yes	Global	No
Rpl_semi_sync_master_net_wait_time [605]				Yes	Global	No
Rpl_semi_sync_master_net_waits [606]				Yes	Global	No
Rpl_semi_sync_master_no_times [606]				Yes	Global	No
Rpl_semi_sync_master_no_tx [606]				Yes	Global	No
Rpl_semi_sync_master_status [606]				Yes	Global	No
Rpl_semi_sync_master_timefunc_failures [606]				Yes	Global	No
rpl_semi_sync_master_timeout [548]			Yes		Global	Yes
rpl_semi_sync_master_trace_level [549]			Yes		Global	Yes
Rpl_semi_sync_master_tx_avg_wait_time [606]				Yes	Global	No
Rpl_semi_sync_master_tx_wait_time [606]				Yes	Global	No
Rpl_semi_sync_master_tx_waits [606]				Yes	Global	No
rpl_semi_sync_master_wait_no_slave [549]			Yes		Global	Yes
Rpl_semi_sync_master_wait_pos_backtraverse [606]				Yes	Global	No
Rpl_semi_sync_master_wait_sessions [607]				Yes	Global	No
Rpl_semi_sync_master_yes_tx [607]				Yes	Global	No
rpl_semi_sync_slave_enabled [550]			Yes		Global	Yes
Rpl_semi_sync_slave_status [607]				Yes	Global	No
rpl_semi_sync_slave_trace_level [550]			Yes		Global	Yes
Rpl_status [607]				Yes	Global	No
safe-mode [459]	Yes	Yes				
safe-show- database [459]	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
safe-user-create [459]	Yes	Yes				
safemalloc-mem-limit	Yes	Yes				
secure-auth [460]	Yes	Yes			Global	Yes
- Variable: secure_auth			Yes		Global	Yes
secure-file-priv [460]	Yes	Yes			Global	No
- Variable: secure_file_priv			Yes		Global	No
Select_full_join [607]				Yes	Both	No
Select_full_range_join [607]				Yes	Both	No
Select_range [607]				Yes	Both	No
Select_range_check [607]				Yes	Both	No
Select_scan [607]				Yes	Both	No
server-id [1858]	Yes	Yes			Global	Yes
- Variable: server_id			Yes		Global	Yes
server-id-bits [2144]	Yes	Yes			Global	No
- Variable: server_id_bits			Yes		Global	No
shared_memory [551]			Yes		Global	No
shared_memory_base_name [551]			Yes		Global	No
show-slave-auth-info [1882]	Yes	Yes				
skip-character-set-client-handshake [437]	Yes	Yes				
skip-concurrent-insert [460]	Yes	Yes				
- Variable: concurrent_insert						
skip-event-scheduler [461]	Yes	Yes				
skip-external-locking [460]	Yes	Yes			Global	No
- Variable: skip_external_locking			Yes		Global	No
skip-grant-tables [461]	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
skip-host-cache [461]	Yes	Yes				
skip-locking [444]	Yes	Yes				
skip-log-warnings [451]	Yes					
skip-name-resolve [461]	Yes	Yes			Global	No
- Variable: skip_name_resolve			Yes		Global	No
skip-ndbcluster [2145]	Yes	Yes				
skip-networking [461]	Yes	Yes			Global	No
- Variable: skip_networking			Yes		Global	No
skip-new	Yes	Yes				
skip-partition [462]	Yes	Yes				
skip-safemalloc [462]	Yes	Yes				
skip-show-database [463]	Yes	Yes			Global	No
- Variable: skip_show_database			Yes		Global	No
skip-slave-start [1882]	Yes	Yes				
skip-ssl [749]	Yes	Yes				
skip-stack-trace [463]	Yes	Yes				
skip-symbolic-links [462]	Yes					
skip-symlink	Yes	Yes				
skip-thread-priority [463]	Yes	Yes				
slave_allow_batching [2155]	Yes	Yes	Yes		Global	Yes
slave_compressed_protocol [1887]	Yes	Yes	Yes		Global	Yes
slave_exec_mode [1887]	Yes	Yes	Yes		Global	Yes
Slave_heartbeat_period [607]				Yes	Global	No
slave-load-tmpdir [1882]	Yes	Yes			Global	No
- Variable: slave_load_tmpdir			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
slave_max_allowed_packet [1869]			Yes		Global	Yes
slave-max-allowed-packet [1869]	Yes	Yes				
- Variable: slave_max_allowed_packet						
slave-net-timeout [1883]	Yes	Yes			Global	Yes
- Variable: slave_net_timeout			Yes		Global	Yes
Slave_open_temp_tables [607]				Yes	Global	No
Slave_received_heartbeats [607]				Yes	Global	No
Slave_retried_transactions [608]				Yes	Global	No
Slave_running [608]				Yes	Global	No
slave-skip-errors [1883]	Yes	Yes			Global	No
- Variable: slave_skip_errors			Yes		Global	No
slave_transaction_retries [1889]	Yes	Yes	Yes		Global	Yes
slave_type_conversions [1889]	Yes	Yes	Yes		Global	No
Slow_launch_threads [608]				Yes	Both	No
slow_launch_time [552]	Yes	Yes	Yes		Global	Yes
Slow_queries [608]				Yes	Both	No
slow-query-log [463]	Yes	Yes			Global	Yes
- Variable: slow_query_log			Yes		Global	Yes
slow_query_log_file [553]	Yes	Yes	Yes		Global	Yes
slow-start-timeout [464]	Yes	Yes				
socket [464]	Yes	Yes	Yes		Global	No
sort_buffer_size [554]	Yes	Yes	Yes		Both	Yes
Sort_merge_passes [608]				Yes	Both	No
Sort_range [608]				Yes	Both	No
Sort_rows [608]				Yes	Both	No
Sort_scan [608]				Yes	Both	No
sporadic-binlog-dump-fail [1897]	Yes	Yes				
sql_auto_is_null [554]			Yes		Both	Yes
sql_big_selects [555]			Yes		Both	Yes
sql_big_tables [435]			Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
sql_buffer_result [555]			Yes		Both	Yes
sql_log_bin [555]			Yes		Both	Yes
sql_log_off [556]			Yes		Both	Yes
sql_log_update [556]			Yes		Session	Yes
sql_low_priority_updates [517]			Yes		Both	Yes
sql_max_join_size [522]			Yes		Both	Yes
sql-mode [464]	Yes	Yes			Both	Yes
- Variable: sql_mode			Yes		Both	Yes
sql_notes [556]			Yes		Both	Yes
sql_quote_show_create [556]			Yes		Both	Yes
sql_safe_updates [557]			Yes		Both	Yes
sql_select_limit [557]			Yes		Both	Yes
sql_slave_skip_counter [1890]			Yes		Global	Yes
sql_warnings [557]			Yes		Both	Yes
ssl [749]	Yes	Yes				
Ssl_accept_renegotiates [608]				Yes	Global	No
Ssl_accepts [608]				Yes	Global	No
ssl-ca [750]	Yes	Yes			Global	No
- Variable: ssl_ca			Yes		Global	No
Ssl_callback_cache_hits [608]				Yes	Global	No
ssl-capath [750]	Yes	Yes			Global	No
- Variable: ssl_capath			Yes		Global	No
ssl-cert [750]	Yes	Yes			Global	No
- Variable: ssl_cert			Yes		Global	No
ssl-cipher [750]	Yes	Yes			Global	No
- Variable: ssl_cipher			Yes		Global	No
Ssl_cipher [608]				Yes	Both	No
Ssl_cipher_list [608]				Yes	Both	No
Ssl_client_connects [608]				Yes	Global	No
Ssl_connect_renegotiates [609]				Yes	Global	No
Ssl_ctx_verify_depth [609]				Yes	Global	No
Ssl_ctx_verify_mode [609]				Yes	Global	No
Ssl_default_timeout [609]				Yes	Both	No
Ssl_finished_accepts [609]				Yes	Global	No
Ssl_finished_connects [609]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
ssl-key [751]	Yes	Yes			Global	No
- Variable: ssl_key			Yes		Global	No
Ssl_session_cache_hits [609]				Yes	Global	No
Ssl_session_cache_misses [609]				Yes	Global	No
Ssl_session_cache_mode [609]				Yes	Global	No
Ssl_session_cache_overflows [609]				Yes	Global	No
Ssl_session_cache_size [609]				Yes	Global	No
Ssl_session_cache_timeouts [609]				Yes	Global	No
Ssl_sessions_reused [609]				Yes	Both	No
Ssl_used_session_cache_entries [609]				Yes	Global	No
Ssl_verify_depth [609]				Yes	Both	No
Ssl_verify_mode [610]				Yes	Both	No
ssl-verify-server-cert [751]	Yes	Yes				
Ssl_version [610]				Yes	Both	No
standalone [462]	Yes	Yes				
storage_engine [559]			Yes		Both	Yes
stored_program_cache [559]	Yes	Yes	Yes		Global	Yes
symbolic-links [462]	Yes	Yes				
sync_binlog [1904]	Yes	Yes	Yes		Global	Yes
sync_frm [559]	Yes	Yes	Yes		Global	Yes
sync_master_info [1890]	Yes	Yes	Yes		Global	Yes
sync_relay_log [1890]	Yes	Yes	Yes		Global	Yes
sync_relay_log_info [1891]	Yes	Yes	Yes		Global	Yes
sysdate-is-now [465]	Yes	Yes				
system_time_zone [560]			Yes		Global	No
table_definition_cache [560]	Yes	Yes	Yes		Global	Yes
table_lock_wait_timeout [561]	Yes	Yes	Yes		Global	Yes
Table_locks_immediate [610]				Yes	Global	No
Table_locks_waited [610]				Yes	Global	No
table_open_cache [561]	Yes	Yes	Yes		Global	Yes
table_type [559]			Yes		Both	Yes
tc-heuristic-recover [465]	Yes	Yes				
Tc_log_max_pages_used [610]				Yes	Global	No
Tc_log_page_size [610]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Tc_log_page_waits [610]				Yes	Global	No
temp-pool [465]	Yes	Yes				
thread_cache_size [561]	Yes	Yes	Yes		Global	Yes
thread_concurrency [562]	Yes	Yes	Yes		Global	No
thread_handling [562]	Yes	Yes	Yes		Global	No
thread_pool_algorithm [563]	Yes	Yes	Yes		Global	No
thread_pool_high_priority_connection [563]	Yes	Yes	Yes		Both	Yes
thread_pool_max_used_threads [564]	Yes	Yes	Yes		Global	Yes
thread_pool_prio_kickup_timer [565]	Yes	Yes	Yes		Both	Yes
thread_pool_size [565]	Yes	Yes	Yes		Global	No
thread_pool_stall_limit [566]	Yes	Yes	Yes		Global	Yes
thread_stack [566]	Yes	Yes	Yes		Global	No
Threads_cached [610]				Yes	Global	No
Threads_connected [610]				Yes	Global	No
Threads_created [610]				Yes	Global	No
Threads_running [611]				Yes	Global	No
time_format [567]			Yes		Global	No
time_zone [567]	Yes	Yes	Yes		Both	Yes
timed_mutexes [567]	Yes	Yes	Yes		Global	Yes
timestamp [567]			Yes		Session	Yes
tmp_table_size [568]	Yes	Yes	Yes		Both	Yes
tmpdir [466]	Yes	Yes	Yes		Global	No
transaction_alloc_block_size [569]	Yes	Yes	Yes		Both	Yes
transaction-isolation [465]	Yes	Yes				
- Variable: tx_isolation						
transaction_prealloc_size [569]	Yes	Yes	Yes		Both	Yes
tx_isolation [570]			Yes		Both	Yes
unique_checks [571]			Yes		Both	Yes
updatable_views_with_limit [571]	Yes	Yes	Yes		Both	Yes
Uptime [611]				Yes	Global	No
Uptime_since_flush_status [611]				Yes	Global	No
user [466]	Yes	Yes				
verbose [467]	Yes	Yes				
version [572]			Yes		Global	No
version_comment [572]			Yes		Global	No
version_compile_machine [572]			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
version_compile_os [572]			Yes		Global	No
wait_timeout [572]	Yes	Yes	Yes		Both	Yes
warning_count [573]			Yes		Session	No

^aThis option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

^bThis option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

5.1.2. Server Configuration Defaults

The MySQL server has many operating parameters, which you can change at server startup using command-line options or configuration files (option files). It is also possible to change many parameters at runtime. For general instructions on setting parameters at startup or runtime, see [Section 5.1.3, “Server Command Options”](#), and [Section 5.1.4, “Server System Variables”](#).

MySQL provides a number of preconfigured option files that can be used as a basis for tuning the MySQL server. Look for files named `my-small.cnf`, `my-medium.cnf`, `my-large.cnf`, and `my-huge.cnf`, which are sample option files for small, medium, large, and very large systems. On Windows, the extension is `.ini` rather than `.cnf`.



Note

On Windows, the `.ini` or `.cnf` option file extension might not be displayed.

For a binary distribution, look for the sample files in or under your installation directory. If you have a source distribution, look in the `support-files` directory. To use a sample file as a base configuration file, rename a copy of it and place the copy in the appropriate location. Regarding names and appropriate location, see the general information provided in [Section 4.2.3.3, “Using Option Files”](#). That section also describes option file format and syntax.

5.1.3. Server Command Options

When you start the `mysqld` server, you can specify program options using any of the methods described in [Section 4.2.3, “Specifying Program Options”](#). The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See [Section 4.2.3.3, “Using Option Files”](#).

`mysqld` reads options from the `[mysqld]` and `[server]` groups. `mysqld_safe` reads options from the `[mysqld]`, `[server]`, `[mysqld_safe]`, and `[safe_mysqld]` groups. `mysql.server` reads options from the `[mysqld]` and `[mysql.server]` groups.

An embedded MySQL server usually reads options from the `[server]`, `[embedded]`, and `[xxxxx_SERVER]` groups, where `xxxxx` is the name of the application into which the server is embedded.

`mysqld` accepts many command options. For a brief summary, execute `mysqld --help`. To see the full list, use `mysqld --verbose --help`.

The following list shows some of the most common server options. Additional options are described in other sections:

- Options that affect security: See [Section 6.1.4, “Security-Related `mysqld` Options and Variables”](#).
- SSL-related options: See [Section 6.3.8.4, “SSL Command Options”](#).
- Binary log control options: See [Section 5.2.4, “The Binary Log”](#).

- Replication-related options: See [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).
- Options for loading plugins such as pluggable storage engines: See [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).
- Options specific to particular storage engines: See [Section 14.3.4, “InnoDB Startup Options and System Variables”](#), [Section 14.5.1, “MyISAM Startup Options”](#), and [Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”](#).

You can also set the values of server system variables by using variable names as options, as described at the end of this section.

Some options control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to an option that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to an option for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some options take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued option is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

- `--help` [434], `-?`

Command-Line Format	<code>-?</code>
	<code>--help</code>
Option-File Format	<code>help</code>

Display a short help message and exit. Use both the `--verbose` [467] and `--help` [434] options to see the full message.

- `--allow-suspicious-udfs` [434]

Command-Line Format	<code>--allow-suspicious-udfs</code>	
Option-File Format	<code>allow-suspicious-udfs</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. See [Section 23.3.2.6, “User-Defined Function Security Precautions”](#).

- `--ansi` [434]

Command-Line Format	<code>--ansi</code>
----------------------------	---------------------

	-a
Option-File Format	ansi

Use standard (ANSI) SQL syntax instead of MySQL syntax. For more precise control over the server SQL mode, use the `--sql-mode` [464] option instead. See [Section 1.8.3, “Running MySQL in ANSI Mode”](#), and [Section 5.1.7, “Server SQL Modes”](#).

- `--basedir=path` [435], `-b path` [435]

Command-Line Format	<code>--basedir=path</code>	
	<code>-b</code>	
Option-File Format	basedir	
Option Sets Variable	Yes, <code>basedir</code> [485]	
Variable Name	<code>basedir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The path to the MySQL installation directory. All paths are usually resolved relative to this directory.

- `--big-tables` [435]

Command-Line Format	<code>--big-tables</code>	
Option-File Format	big-tables	
Option Sets Variable	Yes, <code>big_tables</code> [485]	
Variable Name	<code>big-tables</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>

Enable large result sets by saving all temporary sets in files. This option prevents most “table full” errors, but also slows down queries for which in-memory tables would suffice. Since MySQL 3.23.2, the server is able to handle large result sets automatically by using memory for small temporary tables and switching to disk tables where necessary.

- `--bind-address=addr` [435]

Command-Line Format	<code>--bind-address=addr</code>
Option-File Format	<code>bind-address=addr</code>
Variable Name	<code>bind-address</code>
Variable Scope	Global
Dynamic Variable	No

The MySQL server listens on a single network socket for TCP/IP connections. This socket is bound to a single address, but it is possible for an address to map onto multiple network interfaces. The default

address is `0.0.0.0`. To specify an address explicitly, use the `--bind-address=addr` [435] option at server startup, where `addr` is an IPv4 or IPv6 address or a host name. (IPv6 addresses are not supported before MySQL 5.5.3.) If `addr` is a host name, the server resolves the name to an IP address and binds to that address.

The server treats different types of addresses as follows:

- If the address is `0.0.0.0`, the server accepts TCP/IP connections on all server host IPv4 interfaces.
- If the address is `:::`, the server accepts TCP/IP connections on all server host IPv4 and IPv6 interfaces. Use this address to permit both IPv4 and IPv6 connections on all server interfaces.
- If the address is an IPv4-mapped address, the server accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if the server is bound to `::ffff:127.0.0.1`, clients can connect using `--host=127.0.0.1` or `--host>::ffff:127.0.0.1`.
- If the address is a “regular” IPv4 or IPv6 address (such as `127.0.0.1` or `:::1`), the server accepts TCP/IP connections only for that IPv4 or IPv6 address.

If you intend to bind the server to a specific address, be sure that the `mysql.user` grant table contains an account with administrative privileges that you can use to connect to that address. Otherwise, you will not be able to shut down the server. For example, if you bind the server to `:::`, you can connect to it using all existing accounts. But if you bind the server to `:::1`, it accepts connections only on that address. In that case, first make sure that the `'root'@':::1'` account is present in the `mysql.user` table so you can still connect to the server to shut it down.

- `--binlog-format={ROW|STATEMENT|MIXED}` [436]

Command-Line Format	<code>--binlog-format=format</code>	
Option-File Format	<code>binlog-format=format</code>	
Option Sets Variable	Yes, <code>binlog_format</code> [1899]	
Variable Name	<code>binlog_format</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	STATEMENT
	Valid Values	ROW STATEMENT MIXED

Specify whether to use row-based, statement-based, or mixed replication. Statement-based is the default in MySQL 5.5. See [Section 16.1.2, “Replication Formats”](#).

Previous to MySQL 5.5, setting the binary logging format without enabling binary logging prevented the MySQL server from starting. In MySQL 5.5, the server starts in such cases, the `binlog_format` [1899] global system variable is set, and a warning is logged instead of an error. (Bug #42928)

- `--bootstrap` [436]

Command-Line Format	<code>--bootstrap</code>
Option-File Format	<code>bootstrap</code>

This option is used by the `mysql_install_db` script to create the MySQL privilege tables without having to start a full MySQL server.

This option is unavailable if MySQL was configured with the `DISABLE_GRANT_OPTIONS` [156] compiler flag. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

- `--character-sets-dir=path` [437]

Command-Line Format	<code>--character-sets-dir=path</code>	
Option-File Format	<code>character-sets-dir=path</code>	
Option Sets Variable	Yes, <code>character_sets_dir</code> [488]	
Variable Name	<code>character-sets-dir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>directory name</code>

The directory where character sets are installed. See [Section 10.5, “Character Set Configuration”](#).

- `--character-set-client-handshake` [437]

Command-Line Format	<code>--character-set-client-handshake</code>	
Option-File Format	<code>character-set-client-handshake</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>TRUE</code>

Do not ignore character set information sent by the client. To ignore client information and use the default server character set, use `--skip-character-set-client-handshake` [437]; this makes MySQL behave like MySQL 4.0.

- `--character-set-filesystem=charset_name` [437]

Command-Line Format	<code>--character-set-filesystem=name</code>	
Option-File Format	<code>character-set-filesystem</code>	
Option Sets Variable	Yes, <code>character_set_filesystem</code> [487]	
Variable Name	<code>character_set_filesystem</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The file system character set. This option sets the `character_set_filesystem` [487] system variable.

- `--character-set-server=charset_name` [438], `-C charset_name`

Command-Line Format	<code>--character-set-server</code>	
Option-File Format	<code>character-set-server</code>	
Option Sets Variable	Yes, <code>character_set_server</code> [488]	
Variable Name	<code>character_set_server</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

Use `charset_name` as the default server character set. See Section 10.5, “Character Set Configuration”. If you use this option to specify a nondefault character set, you should also use `--collation-server` [438] to specify the collation.

- `--chroot=path` [438], `-r path`

Command-Line Format	<code>--chroot=name</code>	
	<code>-r name</code>	
Option-File Format	<code>chroot</code>	
	Permitted Values	
	Type	<code>file name</code>

Put the `mysqld` server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure. Note that use of this option somewhat limits `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`.

- `--collation-server=collation_name` [438]

Command-Line Format	<code>--collation-server</code>	
Option-File Format	<code>collation-server</code>	
Option Sets Variable	Yes, <code>collation_server</code> [489]	
Variable Name	<code>collation_server</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

Use `collation_name` as the default server collation. See Section 10.5, “Character Set Configuration”.

- `--console` [438]

Command-Line Format	<code>--console</code>
Option-File Format	<code>console</code>

Platform Specific	windows
--------------------------	---------

(Windows only.) Write error log messages to `stderr` and `stdout`. `mysqld` does not close the console window if this option is used.

As of MySQL 5.5.3, passing in both `--log-error` [448] and `--console` [438] will write to the log file, but not to the console.

- `--core-file` [439]

Command-Line Format	<code>--core-file</code>	
Option-File Format	<code>core-file</code>	
	Permitted Values	
	Type	boolean
	Default	OFF

Write a core file if `mysqld` dies. The name and location of the core file is system dependent. On Linux, a core file named `core.pid` is written to the current working directory of the process, which for `mysqld` is the data directory. `pid` represents the process ID of the server process. On Mac OS X, a core file named `core.pid` is written to the `/cores` directory. On Solaris, use the `coreadm` command to specify where to write the core file and how to name it.

For some systems, to get a core file you must also specify the `--core-file-size` [256] option to `mysqld_safe`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”. On some systems, such as Solaris, you do not get a core file if you are also using the `--user` [466] option. There might be additional restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation.

- `--datadir=path` [439], `-h path`

Command-Line Format	<code>--datadir=path</code>	
	<code>-h</code>	
Option-File Format	<code>datadir</code>	
Option Sets Variable	Yes, <code>datadir</code> [492]	
Variable Name	<code>datadir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The path to the data directory.

- `--debug[=debug_options]` [439], `-# [debug_options]`

Command-Line Format	<code>--debug[=debug_options]</code>	
Option-File Format	<code>debug</code>	
Variable Name	<code>debug</code>	
Variable Scope	Global, Session	

Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	'd:t:o,/tmp/mysqld.trace'

If MySQL is configured with `-DWITH_DEBUG=1` [155], you can use this option to get a trace file of what `mysqld` is doing. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:i:o,mysqld.trace'`. See [MySQL Internals: Porting to Other Systems](#).

Using `-DWITH_DEBUG=1` [155] to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` [439] option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

This option may be given multiple times. Values that begin with `+` or `-` are added to or subtracted from the previous value. For example, `--debug=T` [439] `--debug=+P` [439] sets the value to `P:T`.

- `--debug-sync-timeout[=N]` [440]

Command-Line Format	<code>--debug-sync-timeout[=#]</code>	
Option-File Format	<code>debug-sync-timeout</code>	
	Permitted Values	
	Type	numeric

Controls whether the Debug Sync facility for testing and debugging is enabled. Use of Debug Sync requires that MySQL be configured with the `-DENABLE_DEBUG_SYNC=1` [154] option (see [Section 2.10.4, "MySQL Source-Configuration Options"](#)). If Debug Sync is not compiled in, this option is not available. The option value is a timeout in seconds. The default value is 0, which disables Debug Sync. To enable it, specify a value greater than 0; this value also becomes the default timeout for individual synchronization points. If the option is given without a value, the timeout is set to 300 seconds.

For a description of the Debug Sync facility and how to use synchronization points, see [MySQL Internals: Test Synchronization](#).

- `--default-character-set=charset_name` [440]

Command-Line Format	<code>--default-character-set=name</code>	
	<code>-C name</code>	
Option-File Format	<code>default-character-set=name</code>	
Deprecated	5.0	
	Permitted Values	
	Type	string

Use `charset_name` as the default character set. This option is deprecated in favor of `--character-set-server` [438]. See [Section 10.5, "Character Set Configuration"](#). `--default-character-set` [440] was removed in MySQL 5.5.3.

- `--default-collation=collation_name` [440]

Command-Line Format	<code>--default-collation=name</code>
----------------------------	---------------------------------------

Option-File Format	<code>default-collation=name</code>	
Deprecated	4.1.3	
	Permitted Values	
	Type	<code>string</code>

Use `collation_name` as the default collation. This option is deprecated in favor of `--collation-server` [438]. See Section 10.5, “Character Set Configuration”. `--default-collation` [440] was removed in MySQL 5.5.3.

- `--default-storage-engine=type` [441]

Command-Line Format	<code>--default-storage-engine=name</code>	
Option-File Format	<code>default-storage-engine</code>	
Option Sets Variable	Yes, <code>default_storage_engine</code> [494]	
Variable Name	<code>default_storage_engine</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (<= 5.5.4)	
	Type	<code>enumeration</code>
	Default	<code>MyISAM</code>
	Permitted Values (>= 5.5.5)	
	Type	<code>enumeration</code>
	Default	<code>InnoDB</code>

Set the default storage engine (table type) for tables. See Chapter 14, *Storage Engines*.

If you disable the default storage engine at server startup, you must set the default engine to a different engine or the server will not start.

- `--default-time-zone=timezone` [441]

Command-Line Format	<code>--default-time-zone=name</code>	
Option-File Format	<code>default-time-zone</code>	
	Permitted Values	
	Type	<code>string</code>

Set the default server time zone. This option sets the global `time_zone` [567] system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` [560] system variable).

- `--delay-key-write[={OFF|ON|ALL}]` [441]

Command-Line Format	<code>--delay-key-write[=name]</code>	
Option-File Format	<code>delay-key-write</code>	
Option Sets Variable	Yes, <code>delay_key_write</code> [494]	
Variable Name	<code>delay_key_write</code>	

Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	enumeration
Default	ON
Valid Values	ON OFF ALL

Specify how to use delayed key writes. Delayed key writing causes key buffers not to be flushed between writes for MyISAM tables. OFF disables delayed key writes. ON enables delayed key writes for those tables that were created with the DELAY_KEY_WRITE option. ALL delays key writes for all MyISAM tables. See [Section 8.11.2, “Tuning Server Parameters”](#), and [Section 14.5.1, “MyISAM Startup Options”](#).

**Note**

If you set this variable to ALL, you should not use MyISAM tables from within another program (such as another MySQL server or myisamchk) when the tables are in use. Doing so leads to index corruption.

- `--des-key-file=file_name` [442]

Command-Line Format	<code>--des-key-file=file_name</code>
Option-File Format	<code>des-key-file=file_name</code>

Read the default DES keys from this file. These keys are used by the `DES_ENCRYPT()` [1186] and `DES_DECRYPT()` [1186] functions.

- `--enable-locking` [442]

This option is deprecated and was removed in MySQL 5.5.3. Use `--external-locking` [444] instead.

- `--enable-named-pipe` [442]

Command-Line Format	<code>--enable-named-pipe</code>
Option-File Format	<code>enable-named-pipe</code>
Option Sets Variable	Yes, <code>named_pipe</code> [531]
Platform Specific	windows

Enable support for named pipes. This option applies only on Windows.

- `--enable-pstack` [442]

Version Removed	5.5.7
Command-Line Format	<code>--enable-pstack</code>
Option-File Format	<code>enable-pstack</code>
Deprecated	5.1.54 442

	Permitted Values	
Type	boolean	
Default	FALSE	

This option is nonfunctional before MySQL 5.5.7 and removed in 5.5.7.

- `--engine-condition-pushdown={ON|OFF}`

Version Deprecated	5.5.3	
Command-Line Format	<code>--engine-condition-pushdown</code>	
Option-File Format	<code>engine-condition-pushdown</code>	
Option Sets Variable	Yes, <code>engine_condition_pushdown</code> [497]	
Variable Name	<code>engine_condition_pushdown</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Deprecated	5.5.3, by <code>optimizer_switch</code>	
	Permitted Values	
Type	boolean	
Default	ON	

Sets the `engine_condition_pushdown` [497] system variable. For more information, see Section 8.13.3, “Engine Condition Pushdown Optimization”.

- `--event-scheduler[=value]` [443]

Command-Line Format	<code>--event-scheduler[=value]</code>	
Option-File Format	<code>event-scheduler</code>	
Option Sets Variable	Yes, <code>event_scheduler</code> [498]	
Variable Name	<code>event_scheduler</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
Type	enumeration	
Default	OFF	
Valid Values	ON OFF DISABLED	

Enable or disable, and start or stop, the event scheduler.

For detailed information, see [The `--event-scheduler` Option \[2450\]](#).

- `--exit-info[=flags]` [443], `-T [flags]`

Command-Line Format	<code>--exit-info[=flags]</code>
----------------------------	----------------------------------

	-T [flags]
Option-File Format	exit-info
	Permitted Values
	Type numeric

This is a bit mask of different flags that you can use for debugging the `mysqld` server. Do not use this option unless you know *exactly* what it does!

- `--external-locking` [444]

Command-Line Format	<code>--external-locking</code>
Option-File Format	external-locking
Option Sets Variable	Yes, <code>skip_external_locking</code> [552]
Disabled by	skip-external-locking
	Permitted Values
	Type boolean
	Default FALSE

Enable external locking (system locking), which is disabled by default as of MySQL 4.0. Note that if you use this option on a system on which `lockd` does not fully work (such as Linux), it is easy for `mysqld` to deadlock.

External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see [Section 8.10.5, “External Locking”](#).

- `--flush` [444]

Command-Line Format	<code>--flush</code>
Option-File Format	flush
Variable Name	flush
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type boolean
	Default OFF

Flush (synchronize) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See [Section C.5.4.2, “What to Do If MySQL Keeps Crashing”](#).

- `--gdb` [444]

Command-Line Format	<code>--gdb</code>
Option-File Format	gdb
	Permitted Values
	Type boolean
	Default FALSE

Install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling. See [MySQL Internals: Porting to Other Systems](#).

- `--general-log[={0|1}]` [445]

Command-Line Format	<code>--general-log</code>	
Option-File Format	<code>general-log</code>	
Option Sets Variable	Yes, <code>general_log</code> [502]	
Variable Name	<code>general_log</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Specify the initial general query log state. With no argument or an argument of 1, the `--general-log` [445] option enables the log. If omitted or given with an argument of 0, the option disables the log.

- `--init-file=file_name` [445]

Command-Line Format	<code>--init-file=file_name</code>	
Option-File Format	<code>init-file=file_name</code>	
Option Sets Variable	Yes, <code>init_file</code> [506]	
Variable Name	<code>init_file</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

Read SQL statements from this file at startup. Each statement must be on a single line and should not include comments.

This option is unavailable if MySQL was configured with the `DISABLE_GRANT_OPTIONS` [156] compiler flag. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

- `--innodb-xxx`

The InnoDB options are listed in [Section 14.3.4, “InnoDB Startup Options and System Variables”](#).

- `--install [service_name]` [445]

Command-Line Format	<code>--install [service_name]</code>
----------------------------	---------------------------------------

(Windows only) Install the server as a Windows service that starts automatically during Windows startup. The default service name is `MySQL` if no `service_name` value is given. For more information, see [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

- `--install-manual [service_name]` [445]

Command-Line Format	<code>--install-manual [service_name]</code>
----------------------------	--

(Windows only) Install the server as a Windows service that must be started manually. It does not start automatically during Windows startup. The default service name is `MySQL` if no `service_name` value is given. For more information, see [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

- `--language=lang_name`, `-L lang_name` [446]

Command-Line Format	<code>--language=name</code>	
	<code>-L</code>	
Option-File Format	<code>language</code>	
Option Sets Variable	Yes, <code>language</code> [510]	
Variable Name	<code>language</code>	
Variable Scope	Global	
Dynamic Variable	No	
Deprecated	5.6.1	
	Permitted Values	
	Type	<code>directory name</code>
	Default	<code>/usr/local/mysql/share/mysql/english/</code>

The language to use for error messages. `lang_name` can be given as the language name or as the full path name to the directory where the language files are installed. See [Section 10.2, “Setting the Error Message Language”](#).

As of MySQL 5.5, `--lc-messages-dir` [447] and `--lc-messages` [447] should be used rather than `--language` [446], which is deprecated and handled as an alias for `--lc-messages-dir` [447].

- `--large-pages` [446]

Command-Line Format	<code>--large-pages</code>	
Option-File Format	<code>large-pages</code>	
Option Sets Variable	Yes, <code>large_pages</code> [511]	
Variable Name	<code>large_pages</code>	
Variable Scope	Global	
Dynamic Variable	No	
Platform Specific	linux	
	Permitted Values	
	Type (linux)	<code>boolean</code>
	Default	<code>FALSE</code>

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

MySQL 5.5 supports the Linux implementation of large page support (which is called HugeTLB in Linux). See [Section 8.11.4.2, “Enabling Large Page Support”](#). For Solaris support of large pages, see the description of the `--super-large-pages` [\[462\]](#) option.

`--large-pages` [\[446\]](#) is disabled by default.

- `--lc-messages=locale_name` [\[447\]](#)

Command-Line Format	<code>--lc-messages=name</code>	
Option-File Format	<code>lc-messages</code>	
Option Sets Variable	Yes, <code>lc_messages</code> [512]	
Variable Name	<code>lc-messages</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The locale to use for error messages. The server converts the argument to a language name and combines it with the value of the `--lc-messages-dir` to produce the location for the error message file. See [Section 10.2, “Setting the Error Message Language”](#).

- `--lc-messages-dir=path` [\[447\]](#)

Command-Line Format	<code>--lc-messages-dir=path</code>	
Option-File Format	<code>lc-messages-dir</code>	
Option Sets Variable	Yes, <code>lc_messages_dir</code> [512]	
Variable Name	<code>lc-messages-dir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>directory name</code>

The directory where error messages are located. The value is used together with the value of `--lc-messages` to produce the location for the error message file. See [Section 10.2, “Setting the Error Message Language”](#).

- `--log[=file_name]` [\[447\]](#), `-l [file_name]`

Command-Line Format	<code>--log[=name]</code>	
	<code>-l</code>	
Option-File Format	<code>log</code>	
Option Sets Variable	Yes, <code>log</code> [514]	
Variable Name	<code>log</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
Deprecated	5.1.29, by <code>general-log</code>	

	Permitted Values	
Type	string	
Default	OFF	

This option enables logging to the general query log, which contains entries that record client connections and SQL statements received from clients. The log output destination can be selected with the `--log-output` [449] option. If you omit the file name, MySQL uses `host_name.log` as the file name. See [Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”](#), and [Section 5.2.3, “The General Query Log”](#).

The `--log` [447] option is deprecated and is removed (along with the `log` [514] system variable) in MySQL 5.6. Instead, use the `--general_log` [502] option to enable the general query log and the `--general_log_file=file_name` [503] option to set the general query log file name.

- `--log-error[=file_name]` [448]

Command-Line Format	<code>--log-error[=name]</code>	
Option-File Format	log-error	
Option Sets Variable	Yes, <code>log_error</code> [515]	
Variable Name	log_error	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

Log errors and startup messages to this file. See [Section 5.2.2, “The Error Log”](#). If you omit the file name, MySQL uses `host_name.err`. If the file name has no extension, the server adds an extension of `.err`.

- `--log-isam[=file_name]` [448]

Command-Line Format	<code>--log-isam[=name]</code>	
Option-File Format	log-isam	
	Permitted Values	
	Type	file name

Log all `MyISAM` changes to this file (used only when debugging `MyISAM`).

- `--log-long-format` [448]

Command-Line Format	<code>--log-long-format</code>	
	-0	
Option-File Format	log-long-format	
Deprecated	4.1	

Log extra information to the binary log and slow query log, if they have been activated. For example, the user name and timestamp are logged for all queries. This option is deprecated, as it now represents the default logging behavior. (See the description for `--log-short-format` [450].) The `--log-`

`queries-not-using-indexes` [449] option is available for the purpose of logging queries that do not use indexes to the slow query log. `--log-long-format` [448] was removed in MySQL 5.5.3.

- `--log-output=value,...` [449]

Command-Line Format	<code>--log-output=name</code>	
Option-File Format	<code>log-output</code>	
Option Sets Variable	Yes, <code>log_output</code> [515]	
Variable Name	<code>log_output</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	set
	Default	FILE
	Valid Values	TABLE FILE NONE

This option determines the destination for general query log and slow query log output. The option value can be given as one or more of the words `TABLE`, `FILE`, or `NONE`. `TABLE` select logging to the `general_log` [502] and `slow_log` tables in the `mysql` database as a destination. `FILE` selects logging to log files as a destination. `NONE` disables logging. If `NONE` is present in the option value, it takes precedence over any other words that are present. `TABLE` and `FILE` can both be given to select to both log output destinations.

This option selects log output destinations, but does not enable log output. To do that, use the `--general_log` [502] and `--slow_query_log` [552] options. For `FILE` logging, the `--general_log_file` [503] and `--slow_query_log_file` options determine the log file location. For more information, see [Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”](#).

- `--log-queries-not-using-indexes` [449]

Command-Line Format	<code>--log-queries-not-using-indexes</code>	
Option-File Format	<code>log-queries-not-using-indexes</code>	
Option Sets Variable	Yes, <code>log_queries_not_using_indexes</code> [515]	
Variable Name	<code>log_queries_not_using_indexes</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

If you are using this option with the slow query log enabled, queries that are expected to retrieve all rows are logged. See [Section 5.2.5, “The Slow Query Log”](#). This option does not necessarily mean that

no index is used. For example, a query that uses a full index scan uses an index but would be logged because the index would not limit the number of rows.

- `--log-short-format` [450]

Command-Line Format	<code>--log-short-format</code>	
Option-File Format	<code>log-short-format</code>	
	Permitted Values	
	Type	boolean
	Default	FALSE

Log less information to the binary log and slow query log, if they have been activated.

- `--log-slow-admin-statements` [450]

Command-Line Format	<code>--log-slow-admin-statements</code>	
Option-File Format	<code>log-slow-admin-statements</code>	
	Permitted Values	
	Type	boolean
	Default	FALSE

Log slow administrative statements such as `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `ALTER TABLE` to the slow query log.

- `--log-slow-queries[=file_name]` [450]

Command-Line Format	<code>--log-slow-queries[=name]</code>	
Option-File Format	<code>log-slow-queries</code>	
Option Sets Variable	Yes, <code>log_slow_queries</code> [516]	
Variable Name	<code>log_slow_queries</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
Deprecated	5.1.29, by <code>slow-query-log</code>	
	Permitted Values	
	Type	boolean

This option enables logging to the slow query log, which contains entries for all queries that have taken more than `long_query_time` [517] seconds to execute. See the descriptions of the `--log-long-format` [448] and `--log-short-format` [450] options for details. The log output destination can be selected with the `--log-output` [449] option. If you omit the file name, MySQL uses `host_name-slow.log` as the file name. See [Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”](#), and [Section 5.2.5, “The Slow Query Log”](#).

The `--log-slow-queries` [450] option is deprecated and is removed (along with the `log_slow_queries` [516] system variable) in MySQL 5.6. Instead, use the `--slow-query-log` [552] option to enable the slow query log and the `--slow-query-log-file=file_name` [553] option to set the slow query log file name.

- `--log-tc=file_name` [450]

Command-Line Format	<code>--log-tc=name</code>	
Option-File Format	<code>log-tc</code>	
	Permitted Values	
	Type	<code>file name</code>
	Default	<code>tc.log</code>

The name of the memory-mapped transaction coordinator log file (for XA transactions that affect multiple storage engines when the binary log is disabled). The default name is `tc.log`. The file is created under the data directory if not given as a full path name. Currently, this option is unused.

- `--log-tc-size=size` [451]

Command-Line Format	<code>--log-tc-size=#</code>	
Option-File Format	<code>log-tc-size</code>	
	Permitted Values	
	Platform Bit Size	<code>32</code>
	Type	<code>numeric</code>
	Default	<code>24576</code>
	Max Value	<code>4294967295</code>
	Permitted Values	
	Platform Bit Size	<code>64</code>
	Type	<code>numeric</code>
	Default	<code>24576</code>
	Max Value	<code>18446744073709547520</code>

The size in bytes of the memory-mapped transaction coordinator log. The default size is 24KB.

- `--log-warnings[=level]` [451], `-W [level]`

Command-Line Format	<code>--log-warnings[=#]</code>	
	<code>-W [#]</code>	
Option-File Format	<code>log-warnings</code>	
Option Sets Variable	Yes, <code>log_warnings</code> [516]	
Variable Name	<code>log_warnings</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Disabled by	<code>skip-log-warnings</code>	
	Permitted Values	
	Platform Bit Size	<code>64</code>

	Type	numeric
	Default	1
	Range	0 .. 18446744073709547520

Print out warnings such as `Aborted connection...` to the error log. Enabling this option is recommended, for example, if you use replication (you get more information about what is happening, such as messages about network failures and reconnections). This option is enabled (1) by default, and the default `level` value if omitted is 1. To disable this option, use `--log-warnings=0` [451]. If the value is greater than 1, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See [Section C.5.2.11, “Communication Errors and Aborted Connections”](#).

If a slave server was started with `--log-warnings` [451] enabled, the slave prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth. The server logs messages about statements that are unsafe for statement-based logging only if `--log-warnings` [451] is enabled.

- `--low-priority-updates` [452]

Command-Line Format	<code>--low-priority-updates</code>	
Option-File Format	<code>low-priority-updates</code>	
Option Sets Variable	Yes, <code>low_priority_updates</code> [517]	
Variable Name	<code>low_priority_updates</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	FALSE

Give table-modifying operations (`INSERT`, `REPLACE`, `DELETE`, `UPDATE`) lower priority than selects. This can also be done using `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query, or by `SET LOW_PRIORITY_UPDATES=1` to change the priority in one thread. This affects only storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`). See [Section 8.10.2, “Table Locking Issues”](#).

- `--min-examined-row-limit=number` [452]

Command-Line Format	<code>--min-examined-row-limit=#</code>	
Option-File Format	<code>min-examined-row-limit</code>	
Variable Name	<code>min_examined_row_limit</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric

	Default	0
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	0
	Range	0 .. 18446744073709547520

When this option is set, queries which examine fewer than *number* rows are not written to the slow query log. The default is 0.

- `--memlock` [453]

Command-Line Format	<code>--memlock</code>	
Option-File Format	<code>memlock</code>	
Variable Name	<code>locked_in_memory</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Lock the `mysqld` process in memory. This option might help if you have a problem where the operating system is causing `mysqld` to swap to disk.

`--memlock` [453] works on systems that support the `mlockall()` system call; this includes Solaris, most Linux distributions that use a 2.4 or newer kernel, and perhaps other Unix systems. On Linux systems, you can tell whether or not `mlockall()` (and thus this option) is supported by checking to see whether or not it is defined in the system `mman.h` file, like this:

```
shell> grep mlockall /usr/include/sys/mman.h
```

If `mlockall()` is supported, you should see in the output of the previous command something like the following:

```
extern int mlockall (int __flags) __THROW;
```



Important

Use of this option may require you to run the server as `root`, which, for reasons of security, is normally not a good idea. See [Section 6.1.5, “How to Run MySQL as a Normal User”](#).

On Linux and perhaps other systems, you can avoid the need to run the server as `root` by changing the `limits.conf` file. See the notes regarding the `memlock` limit in [Section 8.11.4.2, “Enabling Large Page Support”](#).

You must not try to use this option on a system that does not support the `mlockall()` system call; if you do so, `mysqld` will very likely crash as soon as you try to start it.

- `--myisam-block-size=N` [454]

Command-Line Format	<code>--myisam-block-size=#</code>	
Option-File Format	<code>myisam-block-size</code>	
	Permitted Values	
	Type	numeric
	Default	1024
	Range	1024 .. 16384

The block size to be used for `MyISAM` index pages.

- `--myisam-recover[=option[,option]...]` [454]

This option is renamed as of MySQL 5.5.3 to `--myisam-recover-options` [454]. See the description of that option for more information.

- `--myisam-recover-options[=option[,option]...]` [454]

Version Introduced	5.5.3	
Command-Line Format	<code>--myisam-recover-options[=name]</code>	
Option-File Format	<code>myisam-recover-options</code>	
Option Sets Variable	Yes, <code>myisam_recover_options</code> [529]	
	Permitted Values	
	Type	enumeration
	Default	OFF
	Valid Values	OFF DEFAULT BACKUP FORCE QUICK

Set the `MyISAM` storage engine recovery mode. The option value is any combination of the values of `OFF`, `DEFAULT`, `BACKUP`, `FORCE`, or `QUICK`. If you specify multiple values, separate them by commas. Specifying the option with no argument is the same as specifying `DEFAULT`, and specifying with an explicit value of "" disables recovery (same as a value of `OFF`). If recovery is enabled, each time `mysqld` opens a `MyISAM` table, it checks whether the table is marked as crashed or was not closed properly. (The last option works only if you are running with external locking disabled.) If this is the case, `mysqld` runs a check on the table. If the table was corrupted, `mysqld` attempts to repair it.

The following options affect how the repair works.

Option	Description
<code>OFF</code>	No recovery.
<code>DEFAULT</code>	Recovery without backup, forcing, or quick checking.
<code>BACKUP</code>	If the data file was changed during recovery, save a backup of the <code>tbl_name.MYD</code> file as <code>tbl_name-datetime.BAK</code> .
<code>FORCE</code>	Run recovery even if we would lose more than one row from the <code>.MYD</code> file.
<code>QUICK</code>	Do not check the rows in the table if there are not any delete blocks.

Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options `BACKUP`, `FORCE`. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

This option was named `--myisam-recover` [454], before MySQL 5.5.3. The old option name still works because it is recognized as an unambiguous prefix of the new name, `--myisam-recover-options` [454]. (Option prefix recognition occurs as described in Section 4.2.3, “Specifying Program Options”.)

The option value `OFF` is available as of MySQL 5.5.3.

See Section 14.5.1, “MyISAM Startup Options”.

- `--old-alter-table` [455]

Command-Line Format	<code>--old-alter-table</code>	
Option-File Format	<code>old-alter-table</code>	
Option Sets Variable	Yes, <code>old_alter_table</code> [534]	
Variable Name	<code>old_alter_table</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

When this option is given, the server does not use the optimized method of processing an `ALTER TABLE` operation. It reverts to using a temporary table, copying over the data, and then renaming the temporary table to the original, as used by MySQL 5.0 and earlier. For more information on the operation of `ALTER TABLE`, see Section 13.1.7, “`ALTER TABLE` Syntax”.

- `--old-style-user-limits` [455]

Command-Line Format	<code>--old-style-user-limits</code>	
Option-File Format	<code>old-style-user-limits</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Enable old-style user limits. (Before MySQL 5.0.3, account resource limits were counted separately for each host from which a user connected rather than per account row in the `user` table.) See [Section 6.3.4, “Setting Account Resource Limits”](#).

- `--one-thread` [\[456\]](#)

Command-Line Format	<code>--one-thread</code>
Option-File Format	<code>one-thread</code>

Only use one thread (for debugging under Linux). This option is available only if the server is built with debugging enabled. See [MySQL Internals: Porting to Other Systems](#).

This option is deprecated and is removed in MySQL 5.6. Use `--thread_handling=no-threads` [\[562\]](#) instead.

- `--open-files-limit=count` [\[456\]](#)

Command-Line Format	<code>--open-files-limit=#</code>
Option-File Format	<code>open-files-limit</code>
Option Sets Variable	Yes, <code>open_files_limit</code> [535]
Variable Name	<code>open_files_limit</code>
Variable Scope	Global
Dynamic Variable	No

Changes the number of file descriptors available to `mysqld`. You should try increasing the value of this option if `mysqld` gives you the error `Too many open files`. `mysqld` uses the option value to reserve descriptors with `setrlimit()`. If the requested number of file descriptors cannot be allocated, `mysqld` writes a warning to the error log.

`mysqld` may attempt to allocate more than the requested number of descriptors (if they are available), using the values of `max_connections` [\[519\]](#) and `table_open_cache` [\[561\]](#) to estimate whether more descriptors will be needed.

- `--partition[=value]` [\[456\]](#)

Command-Line Format	<code>--partition</code>
Option-File Format	<code>partition</code>
Option Sets Variable	Yes, <code>have_partitioning</code> [504]
Variable Name	<code>partition</code>
Variable Scope	Global
Dynamic Variable	No
Disabled by	<code>skip-partition</code>
	Permitted Values
	Type <code>boolean</code>
	Default <code>ON</code>

Enables or disables user-defined partitioning support in the MySQL Server.

- `--pid-file=path` [\[456\]](#)

Command-Line Format	<code>--pid-file=file_name</code>	
Option-File Format	<code>pid-file=file_name</code>	
Option Sets Variable	Yes, <code>pid_file</code> [537]	
Variable Name	<code>pid_file</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The path name of the process ID file. The server creates the file in the data directory unless an absolute path name is given to specify a different directory. This file is used by other programs such as `mysqld_safe` to determine the server's process ID.

- `--plugin-xxx` [457]

Specifies an option that pertains to a server plugin. For example, many storage engines can be built as plugins, and for such engines, options for them can be specified with a `--plugin` [457] prefix. Thus, the `--innodb_file_per_table` [1555] option for InnoDB can be specified as `--plugin-innodb_file_per_table` [1555].

For boolean options that can be enabled or disabled, the `--skip` prefix and other alternative formats are supported as well (see Section 4.2.3.2, “Program Option Modifiers”). For example, `--skip-plugin-innodb_file_per_table` [1555] disables `innodb_file_per_table` [1555].

The rationale for the `--plugin` prefix is that it enables plugin options to be specified unambiguously if there is a name conflict with a built-in server option. For example, were a plugin writer to name a plugin “sql” and implement a “mode” option, the option name might be `--sql-mode` [464], which would conflict with the built-in option of the same name. In such cases, references to the conflicting name are resolved in favor of the built-in option. To avoid the ambiguity, users can specify the plugin option as `--plugin-sql-mode`. Use of the `--plugin` prefix for plugin options is recommended to avoid any question of ambiguity.

- `--plugin-load=plugin_list` [457]

Command-Line Format	<code>--plugin-load=plugin_list</code>	
Option-File Format	<code>plugin-load=plugin_list</code>	
	Permitted Values	
	Type	<code>string</code>

This option tells the server to load the named plugins at startup. The option value is a semicolon-separated list of `name=plugin_library` pairs. Each `name` is the name of the plugin, and `plugin_library` is the name of the shared library that contains the plugin code. Each library file must be located in the directory named by the `plugin_dir` [537] system variable. For example, if plugins named `myplug1` and `myplug2` have library files `myplug1.so` and `myplug2.so`, use this option to load them at startup:

```
shell> mysqld --plugin-load="myplug1=myplug1.so;myplug2=myplug2.so"
```

Quotes are used around the argument value here because semicolon (;) is interpreted as a special character by some command interpreters. (Unix shells treat it as a command terminator, for example.)

All plugins to load must be named in the same `--plugin-load [457]` option. If multiple `--plugin-load [457]` options are given, only the last one is used.

If a plugin library is named without any preceding plugin name, the server loads all plugins in the library.

Each plugin is loaded for a single invocation of `mysqld` only. After a restart, the plugin is not loaded unless `--plugin-load [457]` is used again. This is in contrast to `INSTALL PLUGIN`, which adds an entry to the `mysql.plugins` table to cause the plugin to be loaded for every normal server startup.

Under normal startup, the server determines which plugins to load by reading the `mysql.plugins` system table. If the server is started with the `--skip-grant-tables [461]` option, it does not consult the `mysql.plugins` table and does not load plugins listed there. `--plugin-load [457]` enables plugins to be loaded even when `--skip-grant-tables [461]` is given. `--plugin-load [457]` also enables plugins to be loaded at startup under configurations when plugins cannot be loaded at runtime.

For additional information about plugin loading, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

- `--port=port_num [458]`, `-P port_num`

Command-Line Format	<code>--port=#</code>	
	<code>-P</code>	
Option-File Format	<code>port</code>	
Option Sets Variable	Yes, <code>port [538]</code>	
Variable Name	<code>port</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>3306</code>

The port number to use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` system user.

- `--port-open-timeout=num [458]`

Command-Line Format	<code>--port-open-timeout=#</code>	
Option-File Format	<code>port-open-timeout</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>

On some systems, when the server is stopped, the TCP/IP port might not become available immediately. If the server is restarted quickly afterward, its attempt to reopen the port can fail. This option indicates how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. The default is not to wait.

- `--remove [service_name] [458]`

Command-Line Format	<code>--remove [service_name]</code>
----------------------------	--------------------------------------

(Windows only) Remove a MySQL Windows service. The default service name is `MySQL` if no `service_name` value is given. For more information, see [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

- `--safe-mode` [\[459\]](#)

Version Deprecated	5.5.26
Command-Line Format	<code>--safe-mode</code>
Option-File Format	<code>safe-mode</code>
Deprecated	5.5.26

Skip some optimization stages. This option is deprecated and is removed in MySQL 5.6.

- `--safe-show-database` [\[459\]](#)

Version Removed	5.5.3	
Command-Line Format	<code>--safe-show-database</code>	(until 4.1.1)
Option-File Format	<code>safe-show-database</code>	
Variable Name	<code>safe_show_database</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
Deprecated	4.0.2	
	Permitted Values	
	Type	<code>boolean</code>

This option is deprecated and does not do anything because there is a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See [Section 6.2.1, “Privileges Provided by MySQL”](#). `--safe-show-database` [\[459\]](#) was removed in MySQL 5.5.3.

- `--safe-user-create` [\[459\]](#)

Command-Line Format	<code>--safe-user-create</code>	
Option-File Format	<code>safe-user-create</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` [\[696\]](#) privilege for the `mysql.user` table or any column in the table. If you want a user to have the ability to create new users that have those privileges that the user has the right to grant, you should grant the user the following privilege:

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name' ;
```

This ensures that the user cannot change any privilege columns directly, but has to use the [GRANT](#) statement to give privileges to other users.

- [--secure-auth \[460\]](#)

Command-Line Format	<code>--secure-auth</code>
Option-File Format	<code>secure-auth</code>
Option Sets Variable	Yes, <code>secure_auth</code> [550]
Variable Name	<code>secure_auth</code>
Variable Scope	Global
Dynamic Variable	Yes

This option causes the server to block connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format. Use it to prevent all use of passwords employing the old format (and hence insecure communication over the network).

Server startup fails with an error if this option is enabled and the privilege tables are in pre-4.1 format. See [Section C.5.2.4, “Client does not support authentication protocol”](#).

The `mysql` client also has a [--secure-auth \[281\]](#) option, which prevents connections to a server if the server requires a password in old format for the client account.

- [--secure-file-priv=path \[460\]](#)

Command-Line Format	<code>--secure-file-priv=path</code>	
Option-File Format	<code>secure-file-priv=path</code>	
Option Sets Variable	Yes, <code>secure_file_priv</code> [550]	
Variable Name	<code>secure-file-priv</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

This option limits the effect of the `LOAD_FILE()` [\[1105\]](#) function and the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements to work only with files in the specified directory.

- [--shared-memory \[460\]](#)

Enable shared-memory connections by local clients. This option is available only on Windows.

- [--shared-memory-base-name=name \[460\]](#)

The name of shared memory to use for shared-memory connections. This option is available only on Windows. The default name is `MYSQL`. The name is case sensitive.

- [--skip-concurrent-insert \[460\]](#)

Turn off the ability to select and insert at the same time on `MyISAM` tables. (This is to be used only if you think you have found a bug in this feature.) See [Section 8.10.3, “Concurrent Inserts”](#).

- [--skip-external-locking \[460\]](#)

Do not use external locking (system locking). This affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see [Section 8.10.5, “External Locking”](#).

External locking has been disabled by default since MySQL 4.0.

- `--skip-event-scheduler` [461]

Command-Line Format	<code>--skip-event-scheduler</code>
	<code>--disable-event-scheduler</code>
Option-File Format	<code>skip-event-scheduler</code>

Turns the Event Scheduler `OFF`. This is not the same as disabling the Event Scheduler, which requires setting `--event-scheduler=DISABLED` [443]; see [The `--event-scheduler` Option \[2450\]](#), for more information.

- `--skip-grant-tables` [461]

This option causes the server to start without using the privilege system at all, which gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing `mysqladmin flush-privileges` or `mysqladmin reload` command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement after connecting to the server. This option also suppresses loading of plugins that were installed with the `INSTALL PLUGIN` statement, user-defined functions (UDFs), and scheduled events. To cause plugins to be loaded anyway, use the `--plugin-load` [457] option.

`--skip-grant-tables` [461] is unavailable if MySQL was configured with the `DISABLE_GRANT_OPTIONS` [156] compiler flag. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

- `--skip-host-cache` [461]

Disable use of the internal host cache for faster name-to-IP resolution. In this case, the server performs a DNS lookup every time a client connects. See [Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”](#).

- `--skip-innodb` [1546]

Disable the `InnoDB` storage engine. In this case, if the default storage engine is `InnoDB`, the server will not start unless you also use `--default-storage-engine` [494] to set the default to some other engine.

- `--skip-name-resolve` [461]

Do not resolve host names when checking client connections. Use only IP addresses. If you use this option, all `Host` column values in the grant tables must be IP addresses or `localhost`. See [Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”](#).

- `--skip-networking` [461]

Do not listen for TCP/IP connections at all. All interaction with `mysqld` must be made using named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are permitted. See [Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”](#).

- `--skip-partition` [462]

Command-Line Format	<code>--skip-partition</code>
	<code>--disable-partition</code>
Option-File Format	<code>skip-partition</code>

Disables user-defined partitioning. Existing partitioned tables cannot be accessed when the server has been started with this option.

- `--ssl*`

Options that begin with `--ssl` [749] specify whether to permit clients to connect using SSL and indicate where to find SSL keys and certificates. See [Section 6.3.8.4, “SSL Command Options”](#).

- `--standalone` [462]

Command-Line Format	<code>--standalone</code>
Option-File Format	<code>standalone</code>
Platform Specific	windows

Available on Windows only; instructs the MySQL server not to run as a service.

- `--super-large-pages` [462]

Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a “super large pages” feature enables uses of pages up to 256MB. This feature is available for recent SPARC platforms. It can be enabled or disabled by using the `--super-large-pages` [462] or `--skip-super-large-pages` [462] option.

- `--symbolic-links` [462], `--skip-symbolic-links` [462]

Command-Line Format	<code>--symbolic-links</code>
Option-File Format	<code>symbolic-links</code>

Enable or disable symbolic link support. This option has different effects on Windows and Unix:

- On Windows, enabling symbolic links enables you to establish a symbolic link to a database directory by creating a `db_name.sym` file that contains the path to the real directory. See [Section 8.11.3.1.3, “Using Symbolic Links for Databases on Windows”](#).
- On Unix, enabling symbolic links means that you can link a `MyISAM` index file or data file to another directory with the `INDEX DIRECTORY` or `DATA DIRECTORY` options of the `CREATE TABLE` statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See [Section 8.11.3.1.2, “Using Symbolic Links for MyISAM Tables on Unix”](#).
- `--skip-safemalloc` [462]

Version Removed	5.5.6
Command-Line Format	<code>--skip-safemalloc</code>
Option-File Format	<code>skip-safemalloc</code>

Previously, if MySQL was configured with full debugging support, all MySQL programs check for memory overruns during each memory allocation and memory freeing operation. This checking is very slow,

so for the server you can avoid it when you do not need it by using the `--skip-safemalloc` [462] option.

`safemalloc`, along with this option, was removed in MySQL 5.5.6.

- `--skip-show-database` [463]

Command-Line Format	<code>--skip-show-database</code>
Option-File Format	<code>skip-show-database</code>
Option Sets Variable	Yes, <code>skip_show_database</code> [552]
Variable Name	<code>skip_show_database</code>
Variable Scope	Global
Dynamic Variable	No

With this option, the `SHOW DATABASES` statement is permitted only to users who have the `SHOW DATABASES` [697] privilege, and the statement displays all database names. Without this option, `SHOW DATABASES` is permitted to all users, but displays each database name only if the user has the `SHOW DATABASES` [697] privilege or some privilege for the database. Note that *any* global privilege is considered a privilege for the database.

- `--skip-stack-trace` [463]

Command-Line Format	<code>--skip-stack-trace</code>
Option-File Format	<code>skip-stack-trace</code>

Do not write stack traces. This option is useful when you are running `mysqld` under a debugger. On some systems, you also must use this option to get a core file. See [MySQL Internals: Porting to Other Systems](#).

- `--skip-thread-priority` [463]

Command-Line Format	<code>--skip-thread-priority</code>
Option-File Format	<code>skip-thread-priority</code>
Deprecated	5.1.29

Disable using thread priorities for faster response time. This option is deprecated and is removed in MySQL 5.6.

- `--slow-query-log[={0|1}]` [463]

Command-Line Format	<code>--slow-query-log</code>
Option-File Format	<code>slow-query-log</code>
Option Sets Variable	Yes, <code>slow_query_log</code> [552]
Variable Name	<code>slow_query_log</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>boolean</code>
	Default <code>OFF</code>

Specify the initial slow query log state. With no argument or an argument of 1, the `--slow-query-log` [463] option enables the log. If omitted or given with an argument of 0, the option disables the log.

- `--slow-start-timeout=timeout` [464]

Version Introduced	5.5.20	
Command-Line Format	<code>--slow-start-timeout=#</code>	
Option-File Format	<code>slow-start-timeout=#</code>	
	Permitted Values	
	Type (windows)	numeric
	Default	15000

This option controls the Windows service control manager's service start timeout. The value is the maximum number of milliseconds that the service control manager waits before trying to kill the windows service during startup. The default value is 15000 (15 seconds). If the MySQL service takes too long to start, you may need to increase this value. A value of 0 means there is no timeout.

- `--socket=path` [464]

Command-Line Format	<code>--socket=name</code>	
Option-File Format	<code>socket</code>	
Option Sets Variable	Yes, <code>socket</code> [553]	
Variable Name	<code>socket</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name
	Default	<code>/tmp/mysql.sock</code>

On Unix, this option specifies the Unix socket file to use when listening for local connections. The default value is `/tmp/mysql.sock`. If this option is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory. On Windows, the option specifies the pipe name to use when listening for local connections that use a named pipe. The default value is `MySQL` (not case sensitive).

- `--sql-mode=value[,value[,value...]]` [464]

Command-Line Format	<code>--sql-mode=name</code>	
Option-File Format	<code>sql-mode</code>	
Option Sets Variable	Yes, <code>sql_mode</code> [556]	
Variable Name	<code>sql_mode</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	

Set the SQL mode. See [Section 5.1.7, “Server SQL Modes”](#).

- `--sysdate-is-now` [465]

Command-Line Format	<code>--sysdate-is-now</code>	
Option-File Format	<code>sysdate-is-now</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

`SYSDATE()` [1146] by default returns the time at which it executes, not the time at which the statement in which it occurs begins executing. This differs from the behavior of `NOW()` [1143]. This option causes `SYSDATE()` [1146] to be an alias for `NOW()` [1143]. For information about the implications for binary logging and replication, see the description for `SYSDATE()` [1146] in Section 12.7, “Date and Time Functions” and for `SET TIMESTAMP` in Section 5.1.4, “Server System Variables”.

- `--tc-heuristic-recover={COMMIT|ROLLBACK}` [465]

Command-Line Format	<code>--tc-heuristic-recover=name</code>	
Option-File Format	<code>tc-heuristic-recover</code>	
	Permitted Values	
	Type	<code>enumeration</code>
	Valid Values	<code>COMMIT</code> <code>RECOVER</code>

The type of decision to use in the heuristic recovery process. Currently, this option is unused.

- `--temp-pool` [465]

Command-Line Format	<code>--temp-pool</code>	
Option-File Format	<code>temp-pool</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>TRUE</code>

This option causes most temporary files created by the server to use a small set of names, rather than a unique name for each new file. This works around a problem in the Linux kernel dealing with creating many new files with different names. With the old behavior, Linux seems to “leak” memory, because it is being allocated to the directory entry cache rather than to the disk cache. This option is ignored except on Linux.

- `--transaction-isolation=level` [465]

Command-Line Format	<code>--transaction-isolation=name</code>	
Option-File Format	<code>transaction-isolation</code>	
Option Sets Variable	Yes, <code>tx_isolation</code> [570]	
	Permitted Values	
	Type	<code>enumeration</code>

	Valid Values	<p>READ-UNCOMMITTED</p> <p>READ-COMMITTED</p> <p>REPEATABLE-READ</p> <p>SERIALIZABLE</p>
--	---------------------	--

Sets the default transaction isolation level. The `level` value can be [READ-UNCOMMITTED \[1402\]](#), [READ-COMMITTED \[1402\]](#), [REPEATABLE-READ \[1401\]](#), or [SERIALIZABLE \[1402\]](#). See [Section 13.3.6, “SET TRANSACTION Syntax”](#).

The default transaction isolation level can also be set at runtime using the `SET TRANSACTION` statement or by setting the `tx_isolation [570]` system variable.

- `--tmpdir=path [466]`, `-t path`

Command-Line Format	<code>--tmpdir=path</code>	
	<code>-t</code>	
Option-File Format	<code>tmpdir</code>	
Option Sets Variable	Yes, <code>tmpdir [568]</code>	
Variable Name	<code>tmpdir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The path of the directory to use for creating temporary files. It might be useful if your default `/tmp` directory resides on a partition that is too small to hold temporary tables. This option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (“:”) on Unix and semicolon characters (“;”) on Windows. If the MySQL server is acting as a replication slave, you should not set `--tmpdir [466]` to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. For more information about the storage location of temporary files, see [Section C.5.4.4, “Where MySQL Stores Temporary Files”](#). A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

- `--user={user_name|user_id} [466]`, `-u {user_name|user_id}`

Command-Line Format	<code>--user=name</code>	
	<code>-u name</code>	
Option-File Format	<code>user</code>	
	Permitted Values	
	Type	<code>string</code>

Run the `mysqld` server as the user having the name `user_name` or the numeric user ID `user_id`. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

This option is *mandatory* when starting `mysqld` as `root`. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as `root`. See [Section 6.1.1, “Security Guidelines”](#).

To avoid a possible security hole where a user adds a `--user=root` [466] option to a `my.cnf` file (thus causing the server to run as `root`), `mysqld` uses only the first `--user` [466] option specified and produces a warning if there are multiple `--user` [466] options. Options in `/etc/my.cnf` and `$MYSQL_HOME/my.cnf` are processed before command-line options, so it is recommended that you put a `--user` [466] option in `/etc/my.cnf` and specify a value other than `root`. The option in `/etc/my.cnf` is found before any other `--user` [466] options, which ensures that the server runs as a user other than `root`, and that a warning results if any other `--user` [466] option is found.

- `--verbose` [467], `-v` [467]

Use this option with the `--help` [434] option for detailed help.

- `--version` [467], `-V`

Display version information and exit.

You can assign a value to a server system variable by using an option of the form `--var_name=value`. For example, `--key_buffer_size=32M` [508] sets the `key_buffer_size` [508] variable to a value of 32MB.

Note that when you assign a value to a variable, MySQL might automatically correct the value to stay within a given range, or adjust the value to the closest permissible value if only certain values are permitted.

If you want to restrict the maximum value to which a variable can be set at runtime with `SET`, you can define this by using the `--maximum-var_name=value` command-line option.

You can change the values of most system variables for a running server with the `SET` statement. See [Section 13.7.4, “SET Syntax”](#).

[Section 5.1.4, “Server System Variables”](#), provides a full description for all variables, and additional information for setting them at server startup and runtime. [Section 8.11.2, “Tuning Server Parameters”](#), includes information on optimizing the server by tuning system variables.

5.1.4. Server System Variables

The MySQL server maintains many system variables that indicate how it is configured. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

There are several ways to see the names and values of system variables:

- To see the values that a server will use based on its compiled-in defaults and any option files that it reads, use this command:

```
mysqld --verbose --help
```

- To see the values that a server will use based on its compiled-in defaults, ignoring the settings in any option files, use this command:

```
mysqld --no-defaults --verbose --help
```

- To see the current values used by a running server, use the `SHOW VARIABLES` statement.

This section provides a description of each system variable. Variables with no version indicated are present in all MySQL 5.5 releases. For historical information concerning their implementation, please see <http://dev.mysql.com/doc/refman/5.0/en/>, and <http://dev.mysql.com/doc/refman/4.1/en/>.

The following table lists all available system variables:

Table 5.2. System Variable Summary

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
audit_log_buffer_size [764]			Yes	Global	No
audit_log_file [765]			Yes	Global	No
audit_log_flush [765]			Yes	Global	Yes
audit_log_policy [766]			Yes	Global	Yes
audit_log_rotate_on_size [766]			Yes	Global	Yes
audit_log_strategy [767]			Yes	Global	No
auto_increment_increment [1865]	Yes	Yes	Yes	Both	Yes
auto_increment_offset [1868]	Yes	Yes	Yes	Both	Yes
autocommit [484]	Yes	Yes	Yes	Both	Yes
automatic_sp_privileges [484]			Yes	Global	Yes
back_log [485]	Yes	Yes	Yes	Global	No
basedir [435]	Yes	Yes	Yes	Global	No
big-tables [435]	Yes	Yes			Yes
- Variable: <code>big_tables</code>			Yes	Both	Yes
bind-address [435]	Yes	Yes	Yes	Global	No
binlog_cache_size [1890]	Yes	Yes	Yes	Global	Yes
binlog_direct_non_transactional_updates [1898]	Yes	Yes	Yes	Both	Yes
binlog-format [436]	Yes	Yes			Yes
- Variable: <code>binlog_format</code>			Yes	Both	Yes
binlog_stmt_cache_size [1891]	Yes	Yes	Yes	Global	Yes
bulk_insert_buffer_size [486]	Yes	Yes	Yes	Both	Yes
character_set_client [486]			Yes	Both	Yes
character_set_connection [487]			Yes	Both	Yes
character_set_database [487]^a			Yes	Both	Yes
character-set-filesystem [437]	Yes	Yes			Yes
- Variable: <code>character_set_filesystem</code>			Yes	Both	Yes
character_set_results [488]			Yes	Both	Yes
character-set-server [438]	Yes	Yes			Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- Variable: character_set_server			Yes	Both	Yes
character_set_system [488]			Yes	Global	No
character-sets-dir [437]	Yes	Yes			No
- Variable: character_sets_dir			Yes	Global	No
collation_connection [489]			Yes	Both	Yes
collation_database [489]^b			Yes	Both	Yes
collation-server [438]	Yes	Yes			Yes
- Variable: collation_server			Yes	Both	Yes
completion_type [489]	Yes	Yes	Yes	Both	Yes
concurrent_insert [490]	Yes	Yes	Yes	Global	Yes
connect_timeout [492]	Yes	Yes	Yes	Global	Yes
datadir [439]	Yes	Yes	Yes	Global	No
date_format [492]			Yes	Global	No
datetime_format [492]			Yes	Global	No
debug [439]	Yes	Yes	Yes	Both	Yes
debug_sync [493]			Yes	Session	Yes
default-storage-engine [441]	Yes	Yes			Yes
- Variable: default_storage_engine			Yes	Both	Yes
default_week_format [494]	Yes	Yes	Yes	Both	Yes
delay-key-write [441]	Yes	Yes			Yes
- Variable: delay_key_write			Yes	Global	Yes
delayed_insert_limit [495]	Yes	Yes	Yes	Global	Yes
delayed_insert_timeout [496]	Yes	Yes	Yes	Global	Yes
delayed_queue_size [496]	Yes	Yes	Yes	Global	Yes
div_precision_increment [497]	Yes	Yes	Yes	Both	Yes
engine-condition-pushdown [443]	Yes	Yes			Yes
- Variable: engine_condition_pushdown			Yes	Both	Yes
error_count [498]			Yes	Session	No
event-scheduler [443]	Yes	Yes			Yes

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- Variable: event_scheduler			Yes	Global	Yes
expire_logs_days [498]	Yes	Yes	Yes	Global	Yes
external_user [499]			Yes	Session	No
flush [499]	Yes	Yes	Yes	Global	Yes
flush_time [500]	Yes	Yes	Yes	Global	Yes
foreign_key_checks [500]			Yes	Both	Yes
ft_boolean_syntax [500]	Yes	Yes	Yes	Global	Yes
ft_max_word_len [501]	Yes	Yes	Yes	Global	No
ft_min_word_len [501]	Yes	Yes	Yes	Global	No
ft_query_expansion_limit [502]	Yes	Yes	Yes	Global	No
ft_stopword_file [502]	Yes	Yes	Yes	Global	No
general-log [445]	Yes	Yes			Yes
- Variable: general_log			Yes	Global	Yes
general_log_file [503]	Yes	Yes	Yes	Global	Yes
group_concat_max_len [503]	Yes	Yes	Yes	Both	Yes
have_compress [504]			Yes	Global	No
have_crypt [504]			Yes	Global	No
have_csv [504]			Yes	Global	No
have_dynamic_loading [504]			Yes	Global	No
have_geometry [504]			Yes	Global	No
have_innodb [504]			Yes	Global	No
have_ndbcluster [2147]			Yes	Global	No
have_openssl [504]			Yes	Global	No
have_partitioning [504]			Yes	Global	No
have_profiling [504]			Yes	Global	No
have_query_cache [505]			Yes	Global	No
have_rtree_keys [505]			Yes	Global	No
have_ssl [505]			Yes	Global	No
have_symlink [505]			Yes	Global	No
hostname [505]			Yes	Global	No
identity [505]			Yes	Session	Yes
ignore-builtin-innodb [1546]	Yes	Yes			No
- Variable: ignore_builtin_innodb			Yes	Global	No
init_connect [505]	Yes	Yes	Yes	Global	Yes
init-file [445]	Yes	Yes			No

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- Variable: init_file			Yes	Global	No
init_slave [1885]	Yes	Yes	Yes	Global	Yes
innodb_adaptive_flushing [1547]	Yes	Yes	Yes	Global	Yes
innodb_adaptive_hash_index [1547]	Yes	Yes	Yes	Global	Yes
innodb_additional_mem_pool_size [1548]	Yes	Yes	Yes	Global	No
innodb_autoextend_innodb [1548]	Yes	Yes	Yes	Global	Yes
innodb_autoinc_lock_mode [1548]	Yes	Yes	Yes	Global	No
innodb_buffer_pool_instances [1549]	Yes	Yes	Yes	Global	No
innodb_buffer_pool_size [1549]	Yes	Yes	Yes	Global	No
innodb_change_buffering [1550]	Yes	Yes	Yes	Global	Yes
innodb_checksums [1551]	Yes	Yes	Yes	Global	No
innodb_commit_concurrency [1551]	Yes	Yes	Yes	Global	Yes
innodb_concurrency_tickets [1552]	Yes	Yes	Yes	Global	Yes
innodb_data_file_path [1552]	Yes	Yes	Yes	Global	No
innodb_data_home_dir [1552]	Yes	Yes	Yes	Global	No
innodb_doublewrite [1553]	Yes	Yes	Yes	Global	No
innodb_fast_shutdown [1553]	Yes	Yes	Yes	Global	Yes
innodb_file_format [1554]	Yes	Yes	Yes	Global	Yes
innodb_file_format_check [1554]	Yes	Yes	Yes	Global	No
innodb_file_format_max [1555]	Yes	Yes	Yes	Global	Yes
innodb_file_per_table [1555]	Yes	Yes	Yes	Global	Yes
innodb_flush_log_at_trx_commit [1556]	Yes	Yes	Yes	Global	Yes
innodb_flush_method [1557]	Yes	Yes	Yes	Global	No
innodb_force_load_consistency [1558]	Yes	Yes	Yes	Global	No
innodb_force_recovery [1558]	Yes	Yes	Yes	Global	No
innodb_io_capacity [1559]	Yes	Yes	Yes	Global	Yes
innodb_large_prefix [1560]	Yes	Yes	Yes	Global	Yes
innodb_lock_wait_timeout [1560]	Yes	Yes	Yes	Both	Yes
innodb_locks_unsafe_for_binlog [1561]	Yes	Yes	Yes	Global	No
innodb_log_buffer_size [1563]	Yes	Yes	Yes	Global	No
innodb_log_file_size [1564]	Yes	Yes	Yes	Global	No
innodb_log_files_in_group [1564]	Yes	Yes	Yes	Global	No
innodb_log_group_home_dir [1564]	Yes	Yes	Yes	Global	No
innodb_max_dirty_pages_pct [1565]	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag [1565]	Yes	Yes	Yes	Global	Yes
innodb_mirrored_log_groups [1566]	Yes	Yes	Yes	Global	No
innodb_old_blocks_pct [1566]	Yes	Yes	Yes	Global	Yes
innodb_old_blocks_timeout [1566]	Yes	Yes	Yes	Global	Yes

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
innodb_open_files [1567]	Yes	Yes	Yes	Global	No
innodb_purge_batch_size [1567]	Yes	Yes	Yes	Global	No
innodb_purge_threads [1567]	Yes	Yes	Yes	Global	No
innodb_read_ahead_threshold [1568]	Yes	Yes	Yes	Global	Yes
innodb_read_io_threads [1568]	Yes	Yes	Yes	Global	No
innodb_replication_delay [1569]	Yes	Yes	Yes	Global	Yes
innodb_rollback_on_timeout [1569]	Yes	Yes	Yes	Global	No
innodb_rollback_segments [1570]	Yes	Yes	Yes	Global	Yes
innodb_spin_wait_delay [1570]	Yes	Yes	Yes	Global	Yes
innodb_stats_method [1570]	Yes	Yes	Yes	Global	Yes
innodb_stats_on_metadata [1571]	Yes	Yes	Yes	Global	Yes
innodb_stats_sample_pages [1571]	Yes	Yes	Yes	Global	Yes
innodb_strict_mode [1572]	Yes	Yes	Yes	Both	Yes
innodb_support_xa [1573]	Yes	Yes	Yes	Both	Yes
innodb_sync_spin_loops [1573]	Yes	Yes	Yes	Global	Yes
innodb_table_locks [1573]	Yes	Yes	Yes	Both	Yes
innodb_thread_concurrency [1573]	Yes	Yes	Yes	Global	Yes
innodb_thread_sleep_delay [1574]	Yes	Yes	Yes	Global	Yes
innodb_use_native_aio [1574]	Yes	Yes	Yes	Global	No
innodb_use_sys_malloc [1575]	Yes	Yes	Yes	Global	No
innodb_version [1575]			Yes	Global	No
innodb_write_io_threads [1575]	Yes	Yes	Yes	Global	No
insert_id [506]			Yes	Session	Yes
interactive_timeout [507]	Yes	Yes	Yes	Both	Yes
join_buffer_size [507]	Yes	Yes	Yes	Both	Yes
keep_files_on_create [508]	Yes	Yes	Yes	Both	Yes
key_buffer_size [508]	Yes	Yes	Yes	Global	Yes
key_cache_age_threshold [509]	Yes	Yes	Yes	Global	Yes
key_cache_block_size [509]	Yes	Yes	Yes	Global	Yes
key_cache_division_limit [510]	Yes	Yes	Yes	Global	Yes
language [446]	Yes	Yes	Yes	Global	No
large_files_support [511]			Yes	Global	No
large_page_size [511]			Yes	Global	No
large-pages [446]	Yes	Yes			No
- Variable: large_pages			Yes	Global	No
last_insert_id [511]			Yes	Session	Yes
lc-messages [447]	Yes	Yes			Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- Variable: lc_messages			Yes	Both	Yes
lc-messages-dir [447]	Yes	Yes			No
- Variable: lc_messages_dir			Yes	Global	No
lc_time_names [512]			Yes	Both	Yes
license [513]			Yes	Global	No
local_infile [513]			Yes	Global	Yes
lock_wait_timeout [513]	Yes	Yes	Yes	Both	Yes
locked_in_memory [514]			Yes	Global	No
log [447]	Yes	Yes	Yes	Global	Yes
log_bin [514]			Yes	Global	No
log-bin [1892]	Yes	Yes	Yes	Global	No
log-bin-trust-function-creators [1892]	Yes	Yes			Yes
- Variable: log_bin_trust_function_creators			Yes	Global	Yes
log-bin-trust-routine-creators	Yes	Yes			Yes
- Variable: log_bin_trust_routine_creators			Yes	Global	Yes
log_bin_use_v1_row_events [1901]	Yes	Yes	Yes	Global	No
log-bin-use-v1-row-events [1893]	Yes	Yes			No
- Variable: log_bin_use_v1_row_events			Yes	Global	No
log-error [448]	Yes	Yes			No
- Variable: log_error			Yes	Global	No
log-output [449]	Yes	Yes			Yes
- Variable: log_output			Yes	Global	Yes
log-queries-not-using-indexes [449]	Yes	Yes			Yes
- Variable: log_queries_not_using_indexes			Yes	Global	Yes
log-slave-updates [1870]	Yes	Yes			No
- Variable: log_slave_updates			Yes	Global	No
log_slave_updates [1870]	Yes	Yes	Yes	Global	No
log-slow-queries [450]	Yes	Yes			Yes

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- Variable: log_slow_queries			Yes	Global	Yes
log-warnings [451]	Yes	Yes			Yes
- Variable: log_warnings			Yes	Both	Yes
long_query_time [517]	Yes	Yes	Yes	Both	Yes
low-priority-updates [452]	Yes	Yes			Yes
- Variable: low_priority_updates			Yes	Both	Yes
lower_case_file_system [518]	Yes	Yes	Yes	Global	No
lower_case_table_names [518]	Yes	Yes	Yes	Global	No
max_allowed_packet [519]	Yes	Yes	Yes	Global	Yes
max_binlog_cache_size [520]	Yes	Yes	Yes	Global	Yes
max_binlog_size [1903]	Yes	Yes	Yes	Global	Yes
max_binlog_stmt_cache_size [1903]	Yes	Yes	Yes	Global	Yes
max_connect_errors [521]	Yes	Yes	Yes	Global	Yes
max_connections [519]	Yes	Yes	Yes	Global	Yes
max_delayed_threads [520]	Yes	Yes	Yes	Both	Yes
max_error_count [520]	Yes	Yes	Yes	Both	Yes
max_heap_table_size [521]	Yes	Yes	Yes	Both	Yes
max_insert_delayed_threads [521]			Yes	Both	Yes
max_join_size [522]	Yes	Yes	Yes	Both	Yes
max_length_for_sort_files [522]	Yes	Yes	Yes	Both	Yes
max_long_data_size [523]	Yes	Yes	Yes	Global	No
max_prepared_stmt_count [523]	Yes	Yes	Yes	Global	Yes
max_relay_log_size [524]	Yes	Yes	Yes	Global	Yes
max_seeks_for_key [524]	Yes	Yes	Yes	Both	Yes
max_sort_length [524]	Yes	Yes	Yes	Both	Yes
max_sp_recursion_depth [525]	Yes	Yes	Yes	Both	Yes
max_user_connections [525]	Yes	Yes	Yes	Both	Yes
max_write_lock_count [526]	Yes	Yes	Yes	Global	Yes
memlock [453]	Yes	Yes	Yes	Global	No
metadata_locks_cache_size [526]			Yes	Global	No
min-examined-row-limit [452]	Yes	Yes	Yes	Both	Yes
myisam_data_pointer_size [527]	Yes	Yes	Yes	Global	Yes
myisam_max_sort_file_size [528]	Yes	Yes	Yes	Global	Yes
myisam_mmap_size [528]	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
myisam_recover_options [529]			Yes	Global	No
myisam_repair_threads [529]	Yes	Yes	Yes	Both	Yes
myisam_sort_buffer_size [530]	Yes	Yes	Yes	Both	Yes
myisam_stats_method [530]	Yes	Yes	Yes	Both	Yes
myisam_use_mmap [531]	Yes	Yes	Yes	Global	Yes
named_pipe [531]			Yes	Global	No
ndb_autoincrement_prefix_sz [2148]	Yes	Yes	Yes	Both	Yes
ndb-batch-size [2138]	Yes	Yes	Yes	Global	No
ndb-blob-read-batch-bytes [2140]	Yes	Yes	Yes	Both	Yes
ndb-blob-write-batch-bytes [2140]	Yes	Yes	Yes	Both	Yes
ndb_cache_check_time [2149]	Yes	Yes	Yes	Global	Yes
ndb-cluster-connection-pool [2139]	Yes	Yes	Yes	Global	No
ndb-deferred-constraints [2141]	Yes	Yes			Yes
- Variable: ndb_deferred_constraints			Yes	Both	Yes
ndb_deferred_constraints [2149]	Yes	Yes	Yes	Both	Yes
ndb_distribution [2150]	Yes	Yes	Yes	Both	Yes
ndb-distribution [2141]	Yes	Yes			Yes
- Variable: ndb_distribution			Yes	Both	Yes
ndb_extra_logging [2150]	Yes	Yes	Yes	Global	Yes
ndb_force_send [2150]	Yes	Yes	Yes	Both	Yes
ndb_join_pushdown [2151]			Yes	Global	No
ndb-log-apply-status [2142]	Yes	Yes			No
- Variable: ndb_log_apply_status			Yes	Global	No
ndb_log_bin [2153]	Yes		Yes	Both	Yes
ndb_log_binlog_index [2153]	Yes		Yes	Global	Yes
ndb_log_empty_epochs [2160]	Yes	Yes	Yes	Global	Yes
ndb-log-empty-epochs [2144]	Yes	Yes	Yes	Global	Yes
ndb_log_orig			Yes	Global	No

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
ndb-log-transaction-id [2143]	Yes	Yes			No
- Variable: ndb_log_transaction_id			Yes	Global	No
ndb_log_transaction_id [2154]			Yes	Global	No
ndb-log-update-as-write [2373]	Yes	Yes	Yes	Global	Yes
ndb_log_updated_only [2157]	Yes	Yes	Yes	Global	Yes
ndb_optimization_delay [2147]			Yes	Global	Yes
ndb_table_no_logging [2156]			Yes	Session	Yes
ndb_table_temporary [2157]			Yes	Session	Yes
ndb_use_copying_alter_table [2157]			Yes	Both	No
ndb_use_exact_count [2157]			Yes	Both	Yes
ndb_use_transactions [2157]	Yes	Yes	Yes	Both	Yes
ndb-wait-connected [2146]	Yes	Yes	Yes	Global	No
ndb-wait-setup [2146]	Yes	Yes	Yes	Global	No
ndbinfo_database [2158]			Yes	Global	No
ndbinfo_max_bytes [2159]	Yes		Yes	Both	Yes
ndbinfo_max_rows [2159]	Yes		Yes	Both	Yes
ndbinfo_show_hidden [2159]	Yes		Yes	Both	Yes
ndbinfo_table_prefix [2160]	Yes		Yes	Both	Yes
ndbinfo_version [2160]			Yes	Global	No
net_buffer_length [531]	Yes	Yes	Yes	Both	Yes
net_read_timeout [532]	Yes	Yes	Yes	Both	Yes
net_retry_count [532]	Yes	Yes	Yes	Both	Yes
net_write_timeout [533]	Yes	Yes	Yes	Both	Yes
new [533]	Yes	Yes	Yes	Both	Yes
old [534]	Yes	Yes	Yes	Global	No
old-alter-table [455]	Yes	Yes			Yes
- Variable: old_alter_table			Yes	Both	Yes
old_passwords [534]			Yes	Both	Yes
open-files-limit [456]	Yes	Yes			No
- Variable: open_files_limit			Yes	Global	No
optimizer_prune_level [535]	Yes	Yes	Yes	Both	Yes
optimizer_search_depth [536]	Yes	Yes	Yes	Both	Yes

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
optimizer_switch [536]	Yes	Yes	Yes	Both	Yes
partition [456]	Yes	Yes			No
- Variable: have_partitioning			Yes	Global	No
performance_schema [2554]	Yes	Yes	Yes	Global	No
performance_schema_events_waits_history_long_size [2554]	Yes	Yes	Yes	Global	No
performance_schema_events_waits_history_size [2554]	Yes	Yes	Yes	Global	No
performance_schema_index_cond_classes [2554]	Yes	Yes	Yes	Global	No
performance_schema_index_cond_instances [2555]	Yes	Yes	Yes	Global	No
performance_schema_index_file_classes [2555]	Yes	Yes	Yes	Global	No
performance_schema_index_file_handlers [2555]	Yes	Yes	Yes	Global	No
performance_schema_index_file_instances [2556]	Yes	Yes	Yes	Global	No
performance_schema_index_mutex_classes [2556]	Yes	Yes	Yes	Global	No
performance_schema_index_mutex_instances [2556]	Yes	Yes	Yes	Global	No
performance_schema_index_rwlock_classes [2557]	Yes	Yes	Yes	Global	No
performance_schema_index_rwlock_instances [2557]	Yes	Yes	Yes	Global	No
performance_schema_index_table_handlers [2557]	Yes	Yes	Yes	Global	No
performance_schema_index_table_instances [2558]	Yes	Yes	Yes	Global	No
performance_schema_index_thread_classes [2558]	Yes	Yes	Yes	Global	No
performance_schema_index_thread_instances [2558]	Yes	Yes	Yes	Global	No
pid-file [456]	Yes	Yes			No
- Variable: pid_file			Yes	Global	No
plugin_dir [537]	Yes	Yes	Yes	Global	No
port [458]	Yes	Yes	Yes	Global	No
preload_buffer_size [538]	Yes	Yes	Yes	Both	Yes
profiling [539]			Yes	Both	Yes
profiling_history_size [539]	Yes	Yes	Yes	Both	Yes
protocol_version [539]			Yes	Global	No
proxy_user [539]			Yes	Session	No
pseudo_thread_id [539]			Yes	Session	Yes
query_alloc_block_size [539]	Yes	Yes	Yes	Both	Yes
query_cache_limit [540]	Yes	Yes	Yes	Global	Yes
query_cache_min_res_unit [541]	Yes	Yes	Yes	Global	Yes
query_cache_size [541]	Yes	Yes	Yes	Global	Yes
query_cache_type [542]	Yes	Yes	Yes	Both	Yes
query_cache_wlock_invalidate [542]	Yes	Yes	Yes	Both	Yes
query_prealloc_size [542]	Yes	Yes	Yes	Both	Yes
rand_seed1 [543]			Yes	Session	Yes

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
rand_seed2 [543]			Yes	Session	Yes
range_alloc_block_size [544]	Yes	Yes	Yes	Both	Yes
read_buffer_size [544]	Yes	Yes	Yes	Both	Yes
read_only [544]	Yes	Yes	Yes	Global	Yes
read_rnd_buffer_size [545]	Yes	Yes	Yes	Both	Yes
relay-log [1873]	Yes	Yes			No
- Variable: relay_log			Yes	Global	No
relay-log-index [1874]	Yes	Yes			No
- Variable: relay_log_index			Yes	Global	No
relay_log_index [1885]	Yes	Yes	Yes	Global	No
relay_log_info_file [1886]	Yes	Yes	Yes	Global	No
relay_log_purge [546]	Yes	Yes	Yes	Global	Yes
relay_log_recovery [1886]	Yes	Yes	Yes	Global	No
relay_log_space_limit [546]	Yes	Yes	Yes	Global	No
report-host [1880]	Yes	Yes			No
- Variable: report_host			Yes	Global	No
report-password [1880]	Yes	Yes			No
- Variable: report_password			Yes	Global	No
report-port [1881]	Yes	Yes			No
- Variable: report_port			Yes	Global	No
report-user [1881]	Yes	Yes			No
- Variable: report_user			Yes	Global	No
rpl_recovery_rank [1887]			Yes	Global	Yes
rpl_semi_sync_master_enabled [548]			Yes	Global	Yes
rpl_semi_sync_master_timeout [548]			Yes	Global	Yes
rpl_semi_sync_master_trace_level [549]			Yes	Global	Yes
rpl_semi_sync_master_wait_no_slave [549]			Yes	Global	Yes
rpl_semi_sync_slave_enabled [550]			Yes	Global	Yes
rpl_semi_sync_slave_trace_level [550]			Yes	Global	Yes
safe-show-database [459]	Yes	Yes	Yes	Global	Yes
secure-auth [460]	Yes	Yes			Yes

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- Variable: secure_auth			Yes	Global	Yes
secure-file-priv [460]	Yes	Yes			No
- Variable: secure_file_priv			Yes	Global	No
server-id [1858]	Yes	Yes			Yes
- Variable: server_id			Yes	Global	Yes
server-id-bits [2144]	Yes	Yes			No
- Variable: server_id_bits			Yes	Global	No
shared_memory [551]			Yes	Global	No
shared_memory_base_name [551]			Yes	Global	No
skip-external-locking [460]	Yes	Yes			No
- Variable: skip_external_locking			Yes	Global	No
skip-name-resolve [461]	Yes	Yes			No
- Variable: skip_name_resolve			Yes	Global	No
skip-networking [461]	Yes	Yes			No
- Variable: skip_networking			Yes	Global	No
skip-show-database [463]	Yes	Yes			No
- Variable: skip_show_database			Yes	Global	No
slave_allow_batching [1887]	Yes	Yes	Yes	Global	Yes
slave_compressed_protocol [1887]	Yes	Yes	Yes	Global	Yes
slave_exec_mode [1887]	Yes	Yes	Yes	Global	Yes
slave-load-tmpdir [1882]	Yes	Yes			No
- Variable: slave_load_tmpdir			Yes	Global	No
slave_max_allowed_packet [1869]			Yes	Global	Yes
slave-net-timeout [1883]	Yes	Yes			Yes
- Variable: slave_net_timeout			Yes	Global	Yes
slave-skip-errors [1883]	Yes	Yes			No

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- Variable: slave_skip_errors			Yes	Global	No
slave_transaction_retries [1889]	Yes	Yes	Yes	Global	Yes
slave_type_conversions [1889]	Yes	Yes	Yes	Global	No
slow_launch_time [552]	Yes	Yes	Yes	Global	Yes
slow-query-log [463]	Yes	Yes			Yes
- Variable: slow_query_log			Yes	Global	Yes
slow_query_log_file [552]	Yes	Yes	Yes	Global	Yes
socket [464]	Yes	Yes	Yes	Global	No
sort_buffer_size [554]	Yes	Yes	Yes	Both	Yes
sql_auto_is_null [554]			Yes	Both	Yes
sql_big_selects [555]			Yes	Both	Yes
sql_big_tables [435]			Yes	Both	Yes
sql_buffer_result [555]			Yes	Both	Yes
sql_log_bin [555]			Yes	Both	Yes
sql_log_off [556]			Yes	Both	Yes
sql_log_update [556]			Yes	Session	Yes
sql_low_priority_updates [517]			Yes	Both	Yes
sql_max_join_size [522]			Yes	Both	Yes
sql-mode [464]	Yes	Yes			Yes
- Variable: sql_mode			Yes	Both	Yes
sql_notes [556]			Yes	Both	Yes
sql_quote_show_create [556]			Yes	Both	Yes
sql_safe_updates [557]			Yes	Both	Yes
sql_select_limit [557]			Yes	Both	Yes
sql_slave_skip_counter [1890]			Yes	Global	Yes
sql_warnings [557]			Yes	Both	Yes
ssl-ca [750]	Yes	Yes			No
- Variable: ssl_ca			Yes	Global	No
ssl-capath [750]	Yes	Yes			No
- Variable: ssl_capath			Yes	Global	No
ssl-cert [750]	Yes	Yes			No
- Variable: ssl_cert			Yes	Global	No
ssl-cipher [750]	Yes	Yes			No
- Variable: ssl_cipher			Yes	Global	No
ssl-key [751]	Yes	Yes			No

Server System Variables

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- Variable: ssl_key			Yes	Global	No
storage_engine [559]			Yes	Both	Yes
stored_program_cache [559]	Yes	Yes	Yes	Global	Yes
sync_binlog [1904]	Yes	Yes	Yes	Global	Yes
sync_frm [559]	Yes	Yes	Yes	Global	Yes
sync_master_info [1890]	Yes	Yes	Yes	Global	Yes
sync_relay_log [1890]	Yes	Yes	Yes	Global	Yes
sync_relay_log_info [1891]	Yes	Yes	Yes	Global	Yes
system_time_zone [560]			Yes	Global	No
table_definition_cache [560]	Yes	Yes	Yes	Global	Yes
table_lock_wait_timeout [561]	Yes	Yes	Yes	Global	Yes
table_open_cache [561]	Yes	Yes	Yes	Global	Yes
table_type [559]			Yes	Both	Yes
thread_cache_size [561]	Yes	Yes	Yes	Global	Yes
thread_concurrency [562]	Yes	Yes	Yes	Global	No
thread_handling [562]	Yes	Yes	Yes	Global	No
thread_pool_algorithm [563]	Yes	Yes	Yes	Global	No
thread_pool_high_priority_connection [563]	Yes	Yes	Yes	Both	Yes
thread_pool_max_unused_threads [564]	Yes	Yes	Yes	Global	Yes
thread_pool_prio_kickup_timer [565]	Yes	Yes	Yes	Both	Yes
thread_pool_size [565]	Yes	Yes	Yes	Global	No
thread_pool_stall_limit [566]	Yes	Yes	Yes	Global	Yes
thread_stack [566]	Yes	Yes	Yes	Global	No
time_format [567]			Yes	Global	No
time_zone [567]	Yes	Yes	Yes	Both	Yes
timed_mutexes [567]	Yes	Yes	Yes	Global	Yes
timestamp [567]			Yes	Session	Yes
tmp_table_size [568]	Yes	Yes	Yes	Both	Yes
tmpdir [466]	Yes	Yes	Yes	Global	No
transaction_alloc_block_size [569]	Yes	Yes	Yes	Both	Yes
transaction_prealloc_size [569]	Yes	Yes	Yes	Both	Yes
tx_isolation [570]			Yes	Both	Yes
unique_checks [571]			Yes	Both	Yes
updatable_views_with_limit [571]	Yes	Yes	Yes	Both	Yes
version [572]			Yes	Global	No
version_comment [572]			Yes	Global	No
version_compile_machine [572]			Yes	Global	No
version_compile_os [572]			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
wait_timeout [572]	Yes	Yes	Yes	Both	Yes
warning_count [573]			Yes	Session	No

^aThis option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

^bThis option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

For additional system variable information, see these sections:

- [Section 5.1.5, “Using System Variables”](#), discusses the syntax for setting and displaying system variable values.
- [Section 5.1.5.2, “Dynamic System Variables”](#), lists the variables that can be set at runtime.
- Information on tuning system variables can be found in [Section 8.11.2, “Tuning Server Parameters”](#).
- [Section 14.3.4, “InnoDB Startup Options and System Variables”](#), lists `InnoDB` system variables.
- [Section 17.3.4.3, “MySQL Cluster System Variables”](#), lists system variables which are specific to MySQL Cluster.
- For information on server system variables specific to replication, see [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).



Note

Some of the following variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, before MySQL 5.5.10, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1 [494]` works but `--delay_key_write=ON [494]` does not. As of MySQL 5.5.10, boolean variables can be set at startup to the values `ON`, `TRUE`, `OFF`, and `FALSE` (not case sensitive). See [Section 4.2.3.2, “Program Option Modifiers”](#).

Some system variables control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to a system variable that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to a variable for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some system variables take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued variable is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

- [authentication_windows_log_level \[482\]](#)

Version Introduced	5.5.16
Command-Line Format	<code>--authentication_windows_log_level</code>
Option-File Format	<code>authentication_windows_log_level</code>

Option Sets Variable	Yes, <code>authentication_windows_log_level</code> [482]	
	Permitted Values	
Type	<code>numeric</code>	
Default	<code>0</code>	
Range	<code>0 .. 4</code>	

This variable is available only if the `authentication_windows` Windows authentication plugin is enabled and debugging code is enabled. See [Section 6.3.6.3, “The Windows Native Authentication Plugin”](#).

This variable sets the logging level for the Windows authentication plugin. The following table shows the permitted values.

Value	Description
0	No logging
1	Log only error messages
2	Log level 1 messages and warning messages
3	Log level 2 messages and information notes
4	Log level 3 messages and debug messages

This variable was added in MySQL 5.5.16.

- `authentication_windows_use_principal_name` [483]

Version Introduced	5.5.16	
Command-Line Format	<code>--authentication_windows_use_principal_name</code>	
Option-File Format	<code>authentication_windows_use_principal_name</code>	
Option Sets Variable	Yes, <code>authentication_windows_use_principal_name</code> [483]	
	Permitted Values	
Type	<code>boolean</code>	
Default	<code>ON</code>	

This variable is available only if the `authentication_windows` Windows authentication plugin is enabled. See [Section 6.3.6.3, “The Windows Native Authentication Plugin”](#).

A client that authenticates using the `InitSecurityContext()` function should provide a string identifying the service to which it connects (`targetName`). MySQL uses the principal name (UPN) of the account under which the server is running. The UPN has the form `user_id@computer_name` and need not be registered anywhere to be used. This UPN is sent by the server at the beginning of authentication handshake.

This variable controls whether the server sends the UPN in the initial challenge. By default, the variable is enabled. For security reasons, it can be disabled to avoid sending the server's account name to a client in clear text. If the variable is disabled, the server always sends a `0x00` byte in the first challenge, the client does not specify `targetName`, and as a result, NTLM authentication is used.

If the server fails to obtain its UPN (which will happen primarily in environments that do not support Kerberos authentication), the UPN is not sent by the server and NTLM authentication is used.

This variable was added in MySQL 5.5.16.

- [autocommit \[484\]](#)

Command-Line Format	<code>--autocommit[=#]</code>	
Option-File Format	<code>autocommit</code>	
Option Sets Variable	Yes, autocommit [484]	
Variable Name	<code>autocommit</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>

The autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0, you must use `COMMIT` to accept a transaction or `ROLLBACK` to cancel it. If [autocommit \[484\]](#) is 0 and you change it to 1, MySQL performs an automatic `COMMIT` of any open transaction. Another way to begin a transaction is to use a `START TRANSACTION` or `BEGIN` statement. See [Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

By default, client connections begin with [autocommit \[484\]](#) set to 1. To cause clients to begin with a default of 0, set the global [autocommit \[484\]](#) value by starting the server with the `--autocommit=0 [484]` option. To set the variable using an option file, include these lines:

```
[mysqld]
autocommit=0
```

Before MySQL 5.5.8, the global [autocommit \[484\]](#) value cannot be set at startup. As a workaround, set the [init_connect \[505\]](#) system variable:

```
SET GLOBAL init_connect='SET autocommit=0';
```

The [init_connect \[505\]](#) variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

```
[mysqld]
init_connect='SET autocommit=0'
```

The content of [init_connect \[505\]](#) is not executed for users that have the [SUPER \[697\]](#) privilege (unlike the effect of setting the global [autocommit \[484\]](#) value at startup).

- [automatic_sp_privileges \[484\]](#)

Variable Name	<code>automatic_sp_privileges</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>TRUE</code>

When this variable has a value of 1 (the default), the server automatically grants the [EXECUTE \[696\]](#) and [ALTER ROUTINE \[695\]](#) privileges to the creator of a stored routine, if the user cannot already

execute and alter or drop the routine. (The `ALTER ROUTINE` [695] privilege is required to drop the routine.) The server also automatically drops those privileges from the creator when the routine is dropped. If `automatic_sp_privileges` [484] is 0, the server does not automatically add or drop these privileges.

The creator of a routine is the account used to execute the `CREATE` statement for it. This might not be the same as the account named as the `DEFINER` in the routine definition.

See also [Section 19.2.2, “Stored Routines and MySQL Privileges”](#).

- [back_log](#) [485]

Command-Line Format	<code>--back_log=#</code>
Option-File Format	<code>back_log</code>
Option Sets Variable	Yes, back_log [485]
Variable Name	<code>back_log</code>
Variable Scope	Global
Dynamic Variable	No

The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The [back_log](#) [485] value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix `listen()` system call should have more details. Check your OS documentation for the maximum value for this variable. [back_log](#) [485] cannot be set higher than your operating system limit.

- [basedir](#) [485]

Command-Line Format	<code>--basedir=path</code>	
	<code>-b</code>	
Option-File Format	<code>basedir</code>	
Option Sets Variable	Yes, basedir [485]	
Variable Name	<code>basedir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The MySQL installation base directory. This variable can be set with the `--basedir` [435] option. Relative path names for other variables usually are resolved relative to the base directory.

- [big_tables](#) [485]

If set to 1, all temporary tables are stored on disk rather than in memory. This is a little slower, but the error `The table tbi_name is full` does not occur for `SELECT` operations that require a

large temporary table. The default value for a new connection is 0 (use in-memory temporary tables). Normally, you should never need to set this variable, because in-memory tables are automatically converted to disk-based tables as required.

**Note**

This variable was formerly named `sql_big_tables`.

- `bulk_insert_buffer_size` [486]

Command-Line Format	<code>--bulk_insert_buffer_size=#</code>	
Option-File Format	<code>bulk_insert_buffer_size</code>	
Option Sets Variable	Yes, <code>bulk_insert_buffer_size</code> [486]	
Variable Name	<code>bulk_insert_buffer_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	<code>numeric</code>
	Default	8388608
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	<code>numeric</code>
	Default	8388608
	Range	0 .. 18446744073709547520

MyISAM uses a special tree-like cache to make bulk inserts faster for `INSERT ... SELECT`, `INSERT ... VALUES (...), (...), ...`, and `LOAD DATA INFILE` when adding data to nonempty tables. This variable limits the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. The default value is 8MB.

- `character_set_client` [486]

Variable Name	<code>character_set_client</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The character set for statements that arrive from the client. The session value of this variable is set using the character set requested by the client when the client connects to the server. (Many clients support a `--default-character-set` option to enable this character set to be specified explicitly. See also Section 10.1.4, “Connection Character Sets and Collations”.) The global value of the variable is used to

set the session value in cases when the client-requested value is unknown or not available, or the server is configured to ignore client requests:

- The client is from a version of MySQL older than MySQL 4.1, and thus does not request a character set.
- The client requests a character set not known to the server. For example, a Japanese-enabled client requests `sjis` when connecting to a server not configured with `sjis` support.
- `mysqld` was started with the `--skip-character-set-client-handshake` [437] option, which causes it to ignore client character set configuration. This reproduces MySQL 4.0 behavior and is useful should you wish to upgrade the server without upgrading all the clients.

`ucs2`, `utf16`, and `utf32` cannot be used as a client character set, which means that they also do not work for `SET NAMES` or `SET CHARACTER SET`.

- `character_set_connection` [487]

Variable Name	<code>character_set_connection</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The character set used for literals that do not have a character set introducer and for number-to-string conversion.

- `character_set_database` [487]

Variable Name	<code>character_set_database</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Footnote	This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.	
	Permitted Values	
	Type	<code>string</code>

The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `character_set_server` [488].

- `character_set_filesystem` [487]

Command-Line Format	<code>--character-set-filesystem=name</code>
Option-File Format	<code>character-set-filesystem</code>
Option Sets Variable	Yes, <code>character_set_filesystem</code> [487]
Variable Name	<code>character_set_filesystem</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

	Permitted Values	
	Type	<code>string</code>

The file system character set. This variable is used to interpret string literals that refer to file names, such as in the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` [1105] function. Such file names are converted from `character_set_client` [486] to `character_set_filesystem` [487] before the file opening attempt occurs. The default value is `binary`, which means that no conversion occurs. For systems on which multi-byte file names are permitted, a different value may be more appropriate. For example, if the system represents file names using UTF-8, set `character_set_filesystem` [487] to `'utf8'`.

- `character_set_results` [488]

Variable Name	<code>character_set_results</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The character set used for returning query results such as result sets or error messages to the client.

- `character_set_server` [488]

Command-Line Format	<code>--character-set-server</code>	
Option-File Format	<code>character-set-server</code>	
Option Sets Variable	Yes, <code>character_set_server</code> [488]	
Variable Name	<code>character_set_server</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The server's default character set.

- `character_set_system` [488]

Variable Name	<code>character_set_system</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

The character set used by the server for storing identifiers. The value is always `utf8`.

- `character_sets_dir` [488]

Command-Line Format	<code>--character-sets-dir=path</code>
Option-File Format	<code>character-sets-dir=path</code>

Option Sets Variable	Yes, character_sets_dir [488]	
Variable Name	<code>character-sets-dir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>directory name</code>

The directory where character sets are installed.

- [collation_connection \[489\]](#)

Variable Name	<code>collation_connection</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The collation of the connection character set.

- [collation_database \[489\]](#)

Variable Name	<code>collation_database</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Footnote	This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.	
	Permitted Values	
	Type	<code>string</code>

The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as [collation_server \[489\]](#).

- [collation_server \[489\]](#)

Command-Line Format	<code>--collation-server</code>	
Option-File Format	<code>collation-server</code>	
Option Sets Variable	Yes, collation_server [489]	
Variable Name	<code>collation_server</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The server's default collation.

- [completion_type \[489\]](#)

Command-Line Format	<code>--completion_type=#</code>	
Option-File Format	<code>completion_type</code>	
Option Sets Variable	Yes, completion_type [489]	
Variable Name	<code>completion_type</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (<= 5.5.2)	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Valid Values	<code>0</code> <code>1</code> <code>2</code>
	Permitted Values (>= 5.5.3)	
	Type	<code>enumeration</code>
	Default	<code>NO_CHAIN</code>
	Valid Values	<code>NO_CHAIN</code> <code>CHAIN</code> <code>RELEASE</code> <code>0</code> <code>1</code> <code>2</code>

The transaction completion type. This variable can take the values shown in the following table. As of MySQL 5.5.3, the variable can be assigned using either the name values or corresponding integer values. Before 5.5.3, only the integer values can be used.

Value	Description
<code>NO_CHAIN</code> (or 0)	<code>COMMIT</code> and <code>ROLLBACK</code> are unaffected. This is the default value.
<code>CHAIN</code> (or 1)	<code>COMMIT</code> and <code>ROLLBACK</code> are equivalent to <code>COMMIT AND CHAIN</code> and <code>ROLLBACK AND CHAIN</code> , respectively. (A new transaction starts immediately with the same isolation level as the just-terminated transaction.)
<code>RELEASE</code> (or 2)	<code>COMMIT</code> and <code>ROLLBACK</code> are equivalent to <code>COMMIT RELEASE</code> and <code>ROLLBACK RELEASE</code> , respectively. (The server disconnects after terminating the transaction.)

`completion_type` [489] affects transactions that begin with `START TRANSACTION` or `BEGIN` and end with `COMMIT` or `ROLLBACK`. It does not apply to implicit commits resulting from execution of the statements listed in [Section 13.3.3, “Statements That Cause an Implicit Commit”](#). It also does not apply for `XA COMMIT`, `XA ROLLBACK`, or when `autocommit=1` [484].

- `concurrent_insert` [490]

Command-Line Format	<code>--concurrent_insert[=#]</code>
Option-File Format	<code>concurrent_insert</code>
Option Sets Variable	Yes, <code>concurrent_insert</code> [490]
Variable Name	<code>concurrent_insert</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values (<= 5.5.2)
Type	<code>numeric</code>
Default	1
Valid Values	0 1 2
	Permitted Values (>= 5.5.3)
Type	<code>enumeration</code>
Default	<code>AUTO</code>
Valid Values	<code>NEVER</code> <code>AUTO</code> <code>ALWAYS</code> 0 1 2

If `AUTO` (the default), MySQL permits `INSERT` and `SELECT` statements to run concurrently for `MyISAM` tables that have no free blocks in the middle of the data file. If you start `mysqld` with `--skip-new` [533], this variable is set to `NEVER`.

This variable can take the values shown in the following table. As of MySQL 5.5.3, the variable can be assigned using either the name values or corresponding integer values. Before 5.5.3, only the integer values can be used.

Value	Description
<code>NEVER</code> (or 0)	Disables concurrent inserts
<code>AUTO</code> (or 1)	(Default) Enables concurrent insert for <code>MyISAM</code> tables that do not have holes
<code>ALWAYS</code> (or 2)	Enables concurrent inserts for all <code>MyISAM</code> tables, even those that have holes. For a table with a hole, new rows are inserted at the end of the table if it is in use by another thread. Otherwise, MySQL acquires a normal write lock and inserts the row into the hole.

See also [Section 8.10.3, “Concurrent Inserts”](#).

- [connect_timeout \[492\]](#)

Command-Line Format	<code>--connect_timeout=#</code>	
Option-File Format	<code>connect_timeout</code>	
Option Sets Variable	Yes, connect_timeout [492]	
Variable Name	<code>connect_timeout</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>10</code>

The number of seconds that the `mysqld` server waits for a connect packet before responding with `Bad handshake`. The default value is 10 seconds.

Increasing the [connect_timeout \[492\]](#) value might help if clients frequently encounter errors of the form `Lost connection to MySQL server at 'XXX', system error: errno`.

- [datadir \[492\]](#)

Command-Line Format	<code>--datadir=path</code>	
	<code>-h</code>	
Option-File Format	<code>datadir</code>	
Option Sets Variable	Yes, datadir [492]	
Variable Name	<code>datadir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The MySQL data directory. This variable can be set with the `--datadir [439]` option.

- [date_format \[492\]](#)

This variable is unused.

- [datetime_format \[492\]](#)

This variable is unused.

- [debug \[492\]](#)

Command-Line Format	<code>--debug[=debug_options]</code>	
Option-File Format	<code>debug</code>	
Variable Name	<code>debug</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	492

		Permitted Values
Type		string
Default		'd:t:o,/tmp/mysqlld.trace'

This variable indicates the current debugging settings. It is available only for servers built with debugging support. The initial value comes from the value of instances of the `--debug` [439] option given at server startup. The global and session values may be set at runtime; the `SUPER` [697] privilege is required, even for the session value.

Assigning a value that begins with `+` or `-` cause the value to added to or subtracted from the current value:

```
mysql> SET debug = 'T';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| T       |
+-----+

mysql> SET debug = '+P';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| P:T     |
+-----+

mysql> SET debug = '-P';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| T       |
+-----+
```

- `debug_sync` [493]

Variable Name	<code>debug_sync</code>	
Variable Scope	Session	
Dynamic Variable	Yes	
Permitted Values		
Type		string

This variable is the user interface to the Debug Sync facility. Use of Debug Sync requires that MySQL be configured with the `-DENABLE_DEBUG_SYNC=1` [154] option (see Section 2.10.4, “MySQL Source-Configuration Options”). If Debug Sync is not compiled in, this system variable is not available.

The global variable value is read only and indicates whether the facility is enabled. By default, Debug Sync is disabled and the value of `debug_sync` [493] is `OFF`. If the server is started with `--debug-sync-timeout=N` [440], where `N` is a timeout value greater than 0, Debug Sync is enabled and the value of `debug_sync` [493] is `ON - current signal` followed by the signal name. Also, `N` becomes the default timeout for individual synchronization points.

The session value can be read by any user and will have the same value as the global variable. The session value can be set by users that have the `SUPER` [697] privilege to control synchronization points.

For a description of the Debug Sync facility and how to use synchronization points, see [MySQL Internals: Test Synchronization](#).

- [default_storage_engine \[494\]](#)

Command-Line Format	<code>--default-storage-engine=name</code>	
Option-File Format	<code>default-storage-engine</code>	
Option Sets Variable	Yes, default_storage_engine [494]	
Variable Name	<code>default_storage_engine</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (<= 5.5.4)	
	Type	<code>enumeration</code>
	Default	<code>MyISAM</code>
	Permitted Values (>= 5.5.5)	
	Type	<code>enumeration</code>
	Default	<code>InnoDB</code>

The default storage engine. To set the storage engine at server startup, use the `--default-storage-engine [441]` option. See [Section 5.1.3, “Server Command Options”](#).

If you disable the default storage engine at server startup, you must set the default engine to a different engine or the server will not start.

This variable was added in MySQL 5.5.3 to be used in preference to [storage_engine \[559\]](#), which is now deprecated.

- [default_week_format \[494\]](#)

Command-Line Format	<code>--default_week_format=#</code>	
Option-File Format	<code>default_week_format</code>	
Option Sets Variable	Yes, default_week_format [494]	
Variable Name	<code>default_week_format</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 7</code>

The default mode value to use for the `WEEK()` [\[1151\]](#) function. See [Section 12.7, “Date and Time Functions”](#).

- [delay_key_write \[494\]](#)

Command-Line Format	<code>--delay-key-write[=name]</code>
----------------------------	---------------------------------------

Option-File Format	delay-key-write	
Option Sets Variable	Yes, delay_key_write [494]	
Variable Name	delay-key-write	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	ON
	Valid Values	ON OFF ALL

This option applies only to [MyISAM](#) tables. It can have one of the following values to affect handling of the [DELAY_KEY_WRITE](#) table option that can be used in [CREATE TABLE](#) statements.

Option	Description
OFF	DELAY_KEY_WRITE is ignored.
ON	MySQL honors any DELAY_KEY_WRITE option specified in CREATE TABLE statements. This is the default value.
ALL	All new opened tables are treated as if they were created with the DELAY_KEY_WRITE option enabled.

If [DELAY_KEY_WRITE](#) is enabled for a table, the key buffer is not flushed for the table on every index update, but only when the table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all [MyISAM](#) tables by starting the server with the [--myisam-recover-options](#) [454] option (for example, [--myisam-recover-options=BACKUP,FORCE](#) [454]). See [Section 5.1.3, “Server Command Options”](#), and [Section 14.5.1, “MyISAM Startup Options”](#).



Warning

If you enable external locking with [--external-locking](#) [444], there is no protection against index corruption for tables that use delayed key writes.

- [delayed_insert_limit](#) [495]

Command-Line Format	--delayed_insert_limit=#	
Option-File Format	delayed_insert_limit	
Option Sets Variable	Yes, delayed_insert_limit [495]	
Variable Name	delayed_insert_limit	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32

	Type	numeric
	Default	100
	Range	1 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	100
	Range	1 .. 18446744073709547520

After inserting `delayed_insert_limit` [495] delayed rows, the `INSERT DELAYED` handler thread checks whether there are any `SELECT` statements pending. If so, it permits them to execute before continuing to insert delayed rows.

- `delayed_insert_timeout` [496]

Command-Line Format	<code>--delayed_insert_timeout=#</code>	
Option-File Format	<code>delayed_insert_timeout</code>	
Option Sets Variable	Yes, <code>delayed_insert_timeout</code> [496]	
Variable Name	<code>delayed_insert_timeout</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	300

How many seconds an `INSERT DELAYED` handler thread should wait for `INSERT` statements before terminating.

- `delayed_queue_size` [496]

Command-Line Format	<code>--delayed_queue_size=#</code>	
Option-File Format	<code>delayed_queue_size</code>	
Option Sets Variable	Yes, <code>delayed_queue_size</code> [496]	
Variable Name	<code>delayed_queue_size</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	1000
	Range	1 .. 4294967295
	Permitted Values	

Platform Bit Size	64
Type	numeric
Default	1000
Range	1 .. 18446744073709547520

This is a per-table limit on the number of rows to queue when handling `INSERT DELAYED` statements. If the queue becomes full, any client that issues an `INSERT DELAYED` statement waits until there is room in the queue again.

- `div_precision_increment` [497]

Command-Line Format	<code>--div_precision_increment=#</code>
Option-File Format	<code>div_precision_increment</code>
Option Sets Variable	Yes, <code>div_precision_increment</code> [497]
Variable Name	<code>div_precision_increment</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type numeric
	Default 4
	Range 0 .. 30

This variable indicates the number of digits by which to increase the scale of the result of division operations performed with the `/` [1122] operator. The default value is 4. The minimum and maximum values are 0 and 30, respectively. The following example illustrates the effect of increasing the default value.

```
mysql> SELECT 1/7;
+-----+
| 1/7   |
+-----+
| 0.1429 |
+-----+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+-----+
| 1/7   |
+-----+
| 0.142857142857 |
+-----+
```

- `engine_condition_pushdown` [497]

Version Deprecated	5.5.3
Command-Line Format	<code>--engine-condition-pushdown</code>
Option-File Format	<code>engine-condition-pushdown</code>
Option Sets Variable	Yes, <code>engine_condition_pushdown</code> [497]
Variable Name	<code>engine_condition_pushdown</code>
Variable Scope	Global, Session

Dynamic Variable	Yes
Deprecated	5.5.3, by optimizer_switch
	Permitted Values
Type	boolean
Default	ON

The engine condition pushdown optimization enables processing for certain comparisons to be “pushed down” to the storage engine level for more efficient execution. For more information, see [Section 8.13.3, “Engine Condition Pushdown Optimization”](#).

Engine condition pushdown is used only by the `NDBCLUSTER` storage engine. Enabling this optimization on a MySQL Server acting as a MySQL Cluster SQL node causes `WHERE` conditions on unindexed columns to be evaluated on the cluster’s data nodes and only the rows that match to be sent back to the SQL node that issued the query. This greatly reduces the amount of cluster data that must be sent over the network, increasing the efficiency with which results are returned.

The `engine_condition_pushdown` [\[497\]](#) variable controls whether engine condition pushdown is enabled. By default, this variable is `ON` (1). Setting it to `OFF` (0) disables pushdown.

This variable is deprecated as of MySQL 5.5.3 and is removed in MySQL 5.6. Use the `engine_condition_pushdown` flag of the `optimizer_switch` [\[536\]](#) variable instead. See [Section 8.8.5.2, “Controlling Switchable Optimizations”](#).

- `error_count` [\[498\]](#)

The number of errors that resulted from the last statement that generated messages. This variable is read only. See [Section 13.7.5.18, “SHOW ERRORS Syntax”](#).

- `event_scheduler` [\[498\]](#)

Command-Line Format	<code>--event-scheduler[=value]</code>
Option-File Format	<code>event-scheduler</code>
Option Sets Variable	Yes, <code>event_scheduler</code> [498]
Variable Name	<code>event_scheduler</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	enumeration
Default	OFF
Valid Values	ON OFF DISABLED

This variable indicates the status of the Event Scheduler; possible values are `ON`, `OFF`, and `DISABLED`, with the default being `OFF`. This variable and its effects on the Event Scheduler’s operation are discussed in greater detail in the [Overview section of the Events chapter \[2450\]](#).

- `expire_logs_days` [\[498\]](#)

Command-Line Format	<code>--expire_logs_days=#</code>
Option-File Format	<code>expire_logs_days</code>
Option Sets Variable	Yes, <code>expire_logs_days</code> [498]
Variable Name	<code>expire_logs_days</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>0</code>
	Range <code>0 .. 99</code>

The number of days for automatic binary log file removal. The default is 0, which means “no automatic removal.” Possible removals happen at startup and when the binary log is flushed. Log flushing occurs as indicated in [Section 5.2, “MySQL Server Logs”](#).

To remove binary log files manually, use the `PURGE BINARY LOGS` statement. See [Section 13.4.1.1, “PURGE BINARY LOGS Syntax”](#).

- [external_user](#) [499]

Version Introduced	5.5.7
Variable Name	<code>external_user</code>
Variable Scope	Session
Dynamic Variable	No
	Permitted Values
	Type <code>string</code>

The external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this variable is `NULL`. See [Section 6.3.7, “Proxy Users”](#).

This variable was added in MySQL 5.5.7.

- [flush](#) [499]

Command-Line Format	<code>--flush</code>
Option-File Format	<code>flush</code>
Variable Name	<code>flush</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>boolean</code>
	Default <code>OFF</code>

If `ON`, the server flushes (synchronizes) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle

the synchronizing to disk. See [Section C.5.4.2, “What to Do If MySQL Keeps Crashing”](#). This variable is set to `ON` if you start `mysqld` with the `--flush` [\[444\]](#) option.

- [flush_time](#) [\[500\]](#)

Command-Line Format	<code>--flush_time=#</code>
Option-File Format	<code>flush_time</code>
Option Sets Variable	Yes, flush_time [500]
Variable Name	<code>flush_time</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>0</code>
	Min Value <code>0</code>

If this is set to a nonzero value, all tables are closed every [flush_time](#) [\[500\]](#) seconds to free up resources and synchronize unflushed data to disk. This option is best used only on systems with minimal resources.

- [foreign_key_checks](#) [\[500\]](#)

If set to 1 (the default), foreign key constraints for `InnoDB` tables are checked. If set to 0, they are ignored. Disabling foreign key checking can be useful for reloading `InnoDB` tables in an order different from that required by their parent/child relationships. See [Section 14.3.5.4, “FOREIGN KEY Constraints”](#).

Setting [foreign_key_checks](#) [\[500\]](#) to 0 also affects data definition statements: `DROP SCHEMA` drops a schema even if it contains tables that have foreign keys that are referred to by tables outside the schema, and `DROP TABLE` drops tables that have foreign keys that are referred to by other tables.



Note

Setting [foreign_key_checks](#) [\[500\]](#) to 1 does not trigger a scan of the existing table data. Therefore, rows added to the table while `foreign_key_checks = 0` [\[500\]](#) will not be verified for consistency.

- [ft_boolean_syntax](#) [\[500\]](#)

Command-Line Format	<code>--ft_boolean_syntax=name</code>
Option-File Format	<code>ft_boolean_syntax</code>
Variable Name	<code>ft_boolean_syntax</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>string</code>
	Default <code>+-><()~*:""&</code>

The list of operators supported by boolean full-text searches performed using `IN BOOLEAN MODE`. See [Section 12.9.2, “Boolean Full-Text Searches”](#).

The default variable value is `'+ -><()~*:""&|'`. The rules for changing the value are as follows:

- Operator function is determined by position within the string.
- The replacement value must be 14 characters.
- Each character must be an ASCII nonalphanumeric character.
- Either the first or second character must be a space.
- No duplicates are permitted except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
- Positions 10, 13, and 14 (which by default are set to “:”, “&”, and “|”) are reserved for future extensions.
- `ft_max_word_len` [\[501\]](#)

Command-Line Format	<code>--ft_max_word_len=#</code>	
Option-File Format	<code>ft_max_word_len</code>	
Option Sets Variable	Yes, <code>ft_max_word_len</code> [501]	
Variable Name	<code>ft_max_word_len</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Min Value	<code>10</code>

The maximum length of the word to be included in a `FULLTEXT` index.



Note

`FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_min_word_len` [\[501\]](#)

Command-Line Format	<code>--ft_min_word_len=#</code>	
Option-File Format	<code>ft_min_word_len</code>	
Option Sets Variable	Yes, <code>ft_min_word_len</code> [501]	
Variable Name	<code>ft_min_word_len</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>

	Default	4
	Min Value	1

The minimum length of the word to be included in a `FULLTEXT` index.



Note

`FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_query_expansion_limit` [502]

Command-Line Format	<code>--ft_query_expansion_limit=#</code>	
Option-File Format	<code>ft_query_expansion_limit</code>	
Option Sets Variable	Yes, <code>ft_query_expansion_limit</code> [502]	
Variable Name	<code>ft_query_expansion_limit</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	20
	Range	0 .. 1000

The number of top matches to use for full-text searches performed using `WITH QUERY EXPANSION`.

- `ft_stopword_file` [502]

Command-Line Format	<code>--ft_stopword_file=file_name</code>	
Option-File Format	<code>ft_stopword_file=file_name</code>	
Option Sets Variable	Yes, <code>ft_stopword_file</code> [502]	
Variable Name	<code>ft_stopword_file</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The file from which to read the list of stopwords for full-text searches. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the `storage/myisam/ft_static.c` file). Setting this variable to the empty string (`'`) disables stopword filtering. See also [Section 12.9.4, “Full-Text Stopwords”](#).



Note

`FULLTEXT` indexes must be rebuilt after changing this variable or the contents of the stopword file. Use `REPAIR TABLE tbl_name QUICK`.

- `general_log` [502]

Command-Line Format	<code>--general-log</code>
Option-File Format	<code>general-log</code>
Option Sets Variable	Yes, <code>general_log</code> [502]
Variable Name	<code>general_log</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>boolean</code>
	Default <code>OFF</code>

Whether the general query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The default value depends on whether the `--general_log` [502] option is given. The destination for log output is controlled by the `log_output` [515] system variable; if that value is `NONE`, no log entries are written even if the log is enabled.

- `general_log_file` [503]

Command-Line Format	<code>--general-log-file=file_name</code>
Option-File Format	<code>general_log_file</code>
Option Sets Variable	Yes, <code>general_log_file</code> [503]
Variable Name	<code>general_log_file</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>file name</code>
	Default <code>host_name.log</code>

The name of the general query log file. The default value is `host_name.log`, but the initial value can be changed with the `--general_log_file` [503] option.

- `group_concat_max_len` [503]

Command-Line Format	<code>--group_concat_max_len=#</code>
Option-File Format	<code>group_concat_max_len</code>
Option Sets Variable	Yes, <code>group_concat_max_len</code> [503]
Variable Name	<code>group_concat_max_len</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Platform Bit Size <code>32</code>
	Type <code>numeric</code>
	Default <code>1024</code>

	Range	4 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	1024
	Range	4 .. 18446744073709547520

The maximum permitted result length in bytes for the `GROUP_CONCAT()` [1205] function. The default is 1024.

- `have_compress` [504]

YES if the `zlib` compression library is available to the server, **NO** if not. If not, the `COMPRESS()` [1185] and `UNCOMPRESS()` [1189] functions cannot be used.

- `have_crypt` [504]

YES if the `crypt()` system call is available to the server, **NO** if not. If not, the `ENCRYPT()` [1187] function cannot be used.

- `have_csv` [504]

YES if `mysqld` supports `CSV` tables, **NO** if not.

This variable is deprecated and is removed in MySQL 5.6. Use `SHOW ENGINES` instead.

- `have_dynamic_loading` [504]

YES if `mysqld` supports dynamic loading of plugins, **NO** if not.

- `have_geometry` [504]

YES if the server supports spatial data types, **NO** if not.

- `have_innodb` [504]

YES if `mysqld` supports `InnoDB` tables. **DISABLED** if `--skip-innodb` [1546] is used.

This variable is deprecated and is removed in MySQL 5.6. Use `SHOW ENGINES` instead.

- `have_openssl` [504]

This variable is an alias for `have_ssl` [505].

- `have_partitioning` [504]

YES if `mysqld` supports partitioning.

This variable is deprecated and is removed in MySQL 5.6. Use `SHOW ENGINES` instead.

- `have_profiling` [504]

YES if statement profiling capability is present, **NO** if not. If present, the `profiling` system variable controls whether this capability is enabled or disabled. See [Section 13.7.5.32, “SHOW PROFILES Syntax”](#).

- [have_query_cache \[505\]](#)

YES if `mysqld` supports the query cache, NO if not.

- [have_rtree_keys \[505\]](#)

YES if `RTREE` indexes are available, NO if not. (These are used for spatial indexes in `MyISAM` tables.)

- [have_ssl \[505\]](#)

YES if `mysqld` supports SSL connections, NO if not. `DISABLED` indicates that the server was compiled with SSL support, but was not started with the appropriate `--ssl-xxx` options. For more information, see [Section 6.3.8.2, “Configuring MySQL for SSL”](#).

- [have_symlink \[505\]](#)

YES if symbolic link support is enabled, NO if not. This is required on Unix for support of the `DATA DIRECTORY` and `INDEX DIRECTORY` table options, and on Windows for support of data directory symlinks. If the server is started with the `--skip-symbolic-links [462]` option, the value is `DISABLED`.

- [hostname \[505\]](#)

Variable Name	<code>hostname</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

The server sets this variable to the server host name at startup.

- [identity \[505\]](#)

This variable is a synonym for the `last_insert_id [511]` variable. It exists for compatibility with other database systems. You can read its value with `SELECT @@identity`, and set it using `SET identity`.

- [init_connect \[505\]](#)

Command-Line Format	<code>--init-connect=name</code>	
Option-File Format	<code>init_connect</code>	
Option Sets Variable	Yes, <code>init_connect [505]</code>	
Variable Name	<code>init_connect</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

A string to be executed by the server for each client that connects. The string consists of one or more SQL statements, separated by semicolon characters. For example, each client session begins by default with `autocommit` mode enabled. For older servers (before MySQL 5.5.8), there is no global

`autocommit` [484] system variable to specify that autocommit should be disabled by default, but as a workaround `init_connect` [505] can be used to achieve the same effect:

```
SET GLOBAL init_connect='SET autocommit=0';
```

The `init_connect` [505] variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

```
[mysqld]
init_connect='SET autocommit=0'
```

The content of `init_connect` [505] is not executed for users that have the `SUPER` [697] privilege. This is done so that an erroneous value for `init_connect` [505] does not prevent all clients from connecting. For example, the value might contain a statement that has a syntax error, thus causing client connections to fail. Not executing `init_connect` [505] for users that have the `SUPER` [697] privilege enables them to open a connection and fix the `init_connect` [505] value.

- `init_file` [506]

Command-Line Format	<code>--init-file=file_name</code>
Option-File Format	<code>init-file=file_name</code>
Option Sets Variable	Yes, <code>init_file</code> [506]
Variable Name	<code>init_file</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>file name</code>

The name of the file specified with the `--init-file` [445] option when you start the server. This should be a file containing SQL statements that you want the server to execute when it starts. Each statement must be on a single line and should not include comments. No statement terminator such as `;`, `\g`, or `\G` should be given at the end of each statement.

Note that the `--init-file` [445] option is unavailable if MySQL was configured with the `DISABLE_GRANT_OPTIONS` [156] compiler flag. See Section 2.10.4, “MySQL Source-Configuration Options”.

- `innodb_xxx`

InnoDB system variables are listed in Section 14.3.4, “InnoDB Startup Options and System Variables”. These variables control many aspects of storage, memory use, and I/O patterns for InnoDB tables, and are especially important now that InnoDB is the default storage engine.

- `insert_id` [506]

The value to be used by the following `INSERT` or `ALTER TABLE` statement when inserting an `AUTO_INCREMENT` value. This is mainly used with the binary log.

- `interactive_timeout` [506]

Command-Line Format	<code>--interactive_timeout=#</code>
Option-File Format	<code>interactive_timeout</code>
Option Sets Variable	Yes, <code>interactive_timeout</code> [506]

Variable Name	<code>interactive_timeout</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Type	<code>numeric</code>
Default	<code>28800</code>
Min Value	<code>1</code>

The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the `CLIENT_INTERACTIVE` option to `mysql_real_connect()`. See also `wait_timeout` [572].

- `join_buffer_size` [507]

Command-Line Format	<code>--join_buffer_size=#</code>
Option-File Format	<code>join_buffer_size</code>
Option Sets Variable	Yes, <code>join_buffer_size</code> [507]
Variable Name	<code>join_buffer_size</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

The minimum size of the buffer that is used for plain index scans, range index scans, and joins that do not use indexes and thus perform full table scans. Normally, the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` [507] to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary. There is no gain from setting the buffer larger than required to hold each matching row, and all joins allocate at least the minimum size, so use caution in setting this variable to a large value globally. It is better to keep the global setting small and change to a larger setting only in sessions that are doing large joins. Memory allocation time can cause substantial performance drops if the global size is larger than needed by most queries that use it.

The maximum permissible setting for `join_buffer_size` [507] is 4GB. Values larger than 4GB are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB with a warning).

- `keep_files_on_create` [507]

Command-Line Format	<code>--keep_files_on_create=#</code>
Option-File Format	<code>keep_files_on_create</code>
Option Sets Variable	Yes, <code>keep_files_on_create</code> [507]
Variable Name	<code>keep_files_on_create</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Type	<code>boolean</code>

	Default	OFF
--	----------------	-----

If a `MyISAM` table is created with no `DATA DIRECTORY` option, the `.MYD` file is created in the database directory. By default, if `MyISAM` finds an existing `.MYD` file in this case, it overwrites it. The same applies to `.MYI` files for tables created with no `INDEX DIRECTORY` option. To suppress this behavior, set the `keep_files_on_create` [507] variable to `ON` (1), in which case `MyISAM` will not overwrite existing files and returns an error instead. The default value is `OFF` (0).

If a `MyISAM` table is created with a `DATA DIRECTORY` or `INDEX DIRECTORY` option and an existing `.MYD` or `.MYI` file is found, `MyISAM` always returns an error. It will not overwrite a file in the specified directory.

- `key_buffer_size` [508]

Command-Line Format	<code>--key_buffer_size=#</code>
Option-File Format	<code>key_buffer_size</code>
Option Sets Variable	Yes, <code>key_buffer_size</code> [508]
Variable Name	<code>key_buffer_size</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	32
Type	numeric
Default	8388608
Range	8 .. 4294967295

Index blocks for `MyISAM` tables are buffered and are shared by all threads. `key_buffer_size` [508] is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

The maximum permissible setting for `key_buffer_size` [508] is 4GB on 32-bit platforms. Values larger than 4GB are permitted for 64-bit platforms. The effective maximum size might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware platform. The value of this variable indicates the amount of memory requested. Internally, the server allocates as much memory as possible up to this amount, but the actual allocation might be less.

You can increase the value to get better index handling for all reads and multiple writes; on a system whose primary function is to run MySQL using the `MyISAM` storage engine, 25% of the machine's total memory is an acceptable value for this variable. However, you should be aware that, if you make the value too large (for example, more than 50% of the machine's total memory), your system might start to page and become extremely slow. This is because MySQL relies on the operating system to perform file system caching for data reads, so you must leave some room for the file system cache. You should also consider the memory requirements of any other storage engines that you may be using in addition to `MyISAM`.

For even more speed when writing many rows at the same time, use `LOCK TABLES`. See [Section 8.2.2.1, “Speed of INSERT Statements”](#).

You can check the performance of the key buffer by issuing a `SHOW STATUS` statement and examining the `Key_read_requests` [603], `Key_reads` [603], `Key_write_requests` [603], and `Key_writes` [603] status variables. (See [Section 13.7.5, “SHOW Syntax”](#).) The

`Key_reads/Key_read_requests` ratio should normally be less than 0.01. The `Key_writes/Key_write_requests` ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the `DELAY_KEY_WRITE` table option.

The fraction of the key buffer in use can be determined using `key_buffer_size` [508] in conjunction with the `Key_blocks_unused` [603] status variable and the buffer block size, which is available from the `key_cache_block_size` [509] system variable:

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

This value is an approximation because some space in the key buffer is allocated internally for administrative structures. Factors that influence the amount of overhead for these structures include block size and pointer size. As block size increases, the percentage of the key buffer lost to overhead tends to decrease. Larger blocks results in a smaller number of read operations (because more keys are obtained per read), but conversely an increase in reads of keys that are not examined (if not all keys in a block are relevant to a query).

It is possible to create multiple `MyISAM` key caches. The size limit of 4GB applies to each cache individually, not as a group. See [Section 8.9.2, “The MyISAM Key Cache”](#).

- `key_cache_age_threshold` [509]

Command-Line Format	<code>--key_cache_age_threshold=#</code>
Option-File Format	<code>key_cache_age_threshold</code>
Option Sets Variable	Yes, <code>key_cache_age_threshold</code> [509]
Variable Name	<code>key_cache_age_threshold</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Platform Bit Size 32
	Type <code>numeric</code>
	Default 300
	Range 100 .. 4294967295
	Permitted Values
	Platform Bit Size 64
	Type <code>numeric</code>
	Default 300
	Range 100 .. 18446744073709547520

This value controls the demotion of buffers from the hot sublist of a key cache to the warm sublist. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. See [Section 8.9.2, “The MyISAM Key Cache”](#).

- `key_cache_block_size` [509]

Command-Line Format	<code>--key_cache_block_size=#</code>
----------------------------	---------------------------------------

Option-File Format	<code>key_cache_block_size</code>
Option Sets Variable	Yes, key_cache_block_size [509]
Variable Name	<code>key_cache_block_size</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	<code>numeric</code>
Default	<code>1024</code>
Range	<code>512 .. 16384</code>

The size in bytes of blocks in the key cache. The default value is 1024. See [Section 8.9.2, “The MyISAM Key Cache”](#).

- [key_cache_division_limit \[510\]](#)

Command-Line Format	<code>--key_cache_division_limit=#</code>
Option-File Format	<code>key_cache_division_limit</code>
Option Sets Variable	Yes, key_cache_division_limit [510]
Variable Name	<code>key_cache_division_limit</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	<code>numeric</code>
Default	<code>100</code>
Range	<code>1 .. 100</code>

The division point between the hot and warm sublists of the key cache buffer list. The value is the percentage of the buffer list to use for the warm sublist. Permissible values range from 1 to 100. The default value is 100. See [Section 8.9.2, “The MyISAM Key Cache”](#).

- [language \[510\]](#)

Command-Line Format	<code>--language=name</code>
	<code>-L</code>
Option-File Format	<code>language</code>
Option Sets Variable	Yes, language [510]
Variable Name	<code>language</code>
Variable Scope	Global
Dynamic Variable	No
Deprecated	5.6.1
	Permitted Values
Type	<code>directory name</code>
Default	<code>/usr/local/mysql/share/mysql/english/</code>

The directory where error messages are located. See [Section 10.2, “Setting the Error Message Language”](#).

`language` [510] is removed as of MySQL 5.5.0. Similar information is available from the `lc_messages_dir` [512] and `lc_messages` [512] variables.

- `large_files_support` [511]

Variable Name	<code>large_files_support</code>
Variable Scope	Global
Dynamic Variable	No

Whether `mysqld` was compiled with options for large file support.

- `large_pages` [511]

Command-Line Format	<code>--large-pages</code>	
Option-File Format	<code>large-pages</code>	
Option Sets Variable	Yes, <code>large_pages</code> [511]	
Variable Name	<code>large_pages</code>	
Variable Scope	Global	
Dynamic Variable	No	
Platform Specific	linux	
	Permitted Values	
	Type (linux)	<code>boolean</code>
	Default	<code>FALSE</code>

Whether large page support is enabled (via the `--large-pages` [446] option). See [Section 8.11.4.2, “Enabling Large Page Support”](#).

- `large_page_size` [511]

Variable Name	<code>large_page_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type (linux)	<code>numeric</code>
	Default	<code>0</code>

If large page support is enabled, this shows the size of memory pages. Currently, large memory pages are supported only on Linux; on other platforms, the value of this variable is always 0. See [Section 8.11.4.2, “Enabling Large Page Support”](#).

- `last_insert_id` [511]

The value to be returned from `LAST_INSERT_ID()` [1194]. This is stored in the binary log when you use `LAST_INSERT_ID()` [1194] in a statement that updates a table. Setting this variable does not update the value returned by the `mysql_insert_id()` C API function.

- `lc_messages` [512]

Command-Line Format	<code>--lc-messages=name</code>	
Option-File Format	<code>lc-messages</code>	
Option Sets Variable	Yes, <code>lc_messages</code> [512]	
Variable Name	<code>lc-messages</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The locale to use for error messages. The server converts the value to a language name and combines it with the value of the `lc_messages_dir` [512] to produce the location for the error message file. See Section 10.2, “Setting the Error Message Language”.

- `lc_messages_dir` [512]

Command-Line Format	<code>--lc-messages-dir=path</code>	
Option-File Format	<code>lc-messages-dir</code>	
Option Sets Variable	Yes, <code>lc_messages_dir</code> [512]	
Variable Name	<code>lc-messages-dir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>directory name</code>

The directory where error messages are located. The value is used together with the value of `lc_messages` [512] to produce the location for the error message file. See Section 10.2, “Setting the Error Message Language”.

- `lc_time_names` [512]

Variable Name	<code>lc_time_names</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

This variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()` [1138], `DAYNAME()` [1140] and `MONTHNAME()` [1143] functions. Locale names are POSIX-style values such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting. For further information, see Section 10.7, “MySQL Server Locale Support”.

- [license \[513\]](#)

Variable Name	<code>license</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>GPL</code>

The type of license the server has.

- [local_infile \[513\]](#)

Variable Name	<code>local_infile</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>

Whether `LOCAL` is supported for `LOAD DATA INFILE` statements. If this variable is disabled, clients cannot use `LOCAL` in `LOAD DATA` statements. See [Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”](#).

- [lock_wait_timeout \[513\]](#)

Version Introduced	5.5.3	
Command-Line Format	<code>--lock_wait_timeout=#</code>	
Option-File Format	<code>lock_wait_timeout</code>	
Option Sets Variable	Yes, <code>lock_wait_timeout</code> [513]	
Variable Name	<code>lock_wait_timeout</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>31536000</code>
	Range	<code>1 .. 31536000</code>

This variable specifies the timeout in seconds for attempts to acquire metadata locks. The permissible values range from 1 to 31536000 (1 year). The default is 31536000.

This timeout applies to all statements that use metadata locks. These include DML and DDL operations on tables, views, stored procedures, and stored functions, as well as `LOCK TABLES`, `FLUSH TABLES WITH READ LOCK`, and `HANDLER` statements.

This timeout does not apply to implicit accesses to system tables in the `mysql` database, such as grant tables modified by `GRANT` or `REVOKE` statements or table logging statements. The timeout does apply to system tables accessed directly, such as with `SELECT` or `UPDATE`.

The timeout value applies separately for each metadata lock attempt. A given statement can require more than one lock, so it is possible for the statement to block for longer than the `lock_wait_timeout` [513] value before reporting a timeout error. When lock timeout occurs, `ER_LOCK_WAIT_TIMEOUT` [3993] is reported.

`lock_wait_timeout` [513] does not apply to delayed inserts, which always execute with a timeout of 1 year. This is done to avoid unnecessary timeouts because a session that issues a delayed insert receives no notification of delayed insert timeouts.

This variable was added in MySQL 5.5.3.

- `locked_in_memory` [514]

Variable Name	<code>locked_in_memory</code>
Variable Scope	Global
Dynamic Variable	No

Whether `mysqld` was locked in memory with `--memlock` [453].

- `log` [514]

Whether logging of all statements to the general query log is enabled. See Section 5.2.3, “The General Query Log”.

This variable is deprecated and is removed in MySQL 5.6. Use `general_log` [502] instead.

- `log_bin` [514]

Variable Name	<code>log_bin</code>
Variable Scope	Global
Dynamic Variable	No

Whether the binary log is enabled. If the `--log-bin` [1892] option is used, then the value of this variable is `ON`; otherwise it is `OFF`. This variable reports only on the status of binary logging (enabled or disabled); it does not actually report the value to which `--log-bin` [1892] is set.

See Section 5.2.4, “The Binary Log”.

- `log_bin_trust_function_creators` [514]

Command-Line Format	<code>--log-bin-trust-function-creators</code>	
Option-File Format	<code>log-bin-trust-function-creators</code>	
Option Sets Variable	Yes, <code>log_bin_trust_function_creators</code> [514]	
Variable Name	<code>log_bin_trust_function_creators</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

This variable applies when binary logging is enabled. It controls whether stored function creators can be trusted not to create stored functions that will cause unsafe events to be written to the binary log. If set to 0 (the default), users are not permitted to create or alter stored functions unless they have the [SUPER \[697\]](#) privilege in addition to the [CREATE ROUTINE \[695\]](#) or [ALTER ROUTINE \[695\]](#) privilege. A setting of 0 also enforces the restriction that a function must be declared with the [DETERMINISTIC](#) characteristic, or with the [READS SQL DATA](#) or [NO SQL](#) characteristic. If the variable is set to 1, MySQL does not enforce these restrictions on stored function creation. This variable also applies to trigger creation. See [Section 19.7, “Binary Logging of Stored Programs”](#).

- [log_error \[515\]](#)

Command-Line Format	<code>--log-error[=name]</code>	
Option-File Format	<code>log-error</code>	
Option Sets Variable	Yes, log_error [515]	
Variable Name	<code>log_error</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The location of the error log.

- [log_output \[515\]](#)

Command-Line Format	<code>--log-output=name</code>	
Option-File Format	<code>log-output</code>	
Option Sets Variable	Yes, log_output [515]	
Variable Name	<code>log_output</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>set</code>
	Default	<code>FILE</code>
	Valid Values	<code>TABLE</code> <code>FILE</code> <code>NONE</code>

The destination for general query log and slow query log output. The value can be a comma-separated list of one or more of the words [TABLE](#) (log to tables), [FILE](#) (log to files), or [NONE](#) (do not log to tables or files). The default value is [FILE](#). [NONE](#), if present, takes precedence over any other specifiers. If the value is [NONE](#) log entries are not written even if the logs are enabled. If the logs are not enabled, no logging occurs even if the value of [log_output \[515\]](#) is not [NONE](#). For more information, see [Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”](#).

- [log_queries_not_using_indexes \[515\]](#)

Command-Line Format	<code>--log-queries-not-using-indexes</code>	
Option-File Format	<code>log-queries-not-using-indexes</code>	
Option Sets Variable	Yes, <code>log_queries_not_using_indexes</code> [515]	
Variable Name	<code>log_queries_not_using_indexes</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Whether queries that do not use indexes are logged to the slow query log. See [Section 5.2.5, “The Slow Query Log”](#).

- `log_slow_queries` [516]

Command-Line Format	<code>--log-slow-queries[=name]</code>	
Option-File Format	<code>log-slow-queries</code>	
Option Sets Variable	Yes, <code>log_slow_queries</code> [516]	
Variable Name	<code>log_slow_queries</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
Deprecated	5.1.29, by <code>slow-query-log</code>	
	Permitted Values	
	Type	<code>boolean</code>

Whether slow queries should be logged. “Slow” is determined by the value of the `long_query_time` [517] variable. See [Section 5.2.5, “The Slow Query Log”](#).

This variable is deprecated and is removed in MySQL 5.6. Use `slow_query_log` [552] instead.

- `log_warnings` [516]

Command-Line Format	<code>--log-warnings[=#]</code>	
	<code>-W [#]</code>	
Option-File Format	<code>log-warnings</code>	
Option Sets Variable	Yes, <code>log_warnings</code> [516]	
Variable Name	<code>log_warnings</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Disabled by	<code>skip-log-warnings</code>	
	Permitted Values	
	Platform Bit Size	<code>64</code>

Type	numeric
Default	1
Range	0 .. 18446744073709547520

Whether to produce additional warning messages to the error log. It is enabled (1) by default and can be disabled by setting it to 0. Aborted connections and access-denied errors for new connection attempts are logged if the value is greater than 1. The server logs messages about statements that are unsafe for statement-based logging only if the value is greater than 0.

- [long_query_time \[517\]](#)

Command-Line Format	--long_query_time=#	
Option-File Format	long_query_time	
Option Sets Variable	Yes, long_query_time [517]	
Variable Name	long_query_time	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	10
	Min Value	0

If a query takes longer than this many seconds, the server increments the [Slow_queries \[608\]](#) status variable. If the slow query log is enabled, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum and default values of [long_query_time \[517\]](#) are 0 and 10, respectively. The value can be specified to a resolution of microseconds. For logging to a file, times are written including the microseconds part. For logging to tables, only integer times are written; the microseconds part is ignored. See [Section 5.2.5, “The Slow Query Log”](#).

- [low_priority_updates \[517\]](#)

Command-Line Format	--low-priority-updates	
Option-File Format	low-priority-updates	
Option Sets Variable	Yes, low_priority_updates [517]	
Variable Name	low_priority_updates	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	FALSE

If set to 1, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. This affects only storage engines that use

only table-level locking (such as [MyISAM](#), [MEMORY](#), and [MERGE](#)). This variable previously was named [sql_low_priority_updates](#).

- [lower_case_file_system](#) [518]

Command-Line Format	<code>--lower_case_file_system[=#]</code>	
Option-File Format	<code>lower_case_file_system</code>	
Option Sets Variable	Yes, lower_case_file_system [518]	
Variable Name	<code>lower_case_file_system</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>boolean</code>

This variable describes the case sensitivity of file names on the file system where the data directory is located. `OFF` means file names are case sensitive, `ON` means they are not case sensitive. This variable is read only because it reflects a file system attribute and setting it would have no effect on the file system.

- [lower_case_table_names](#) [518]

Command-Line Format	<code>--lower_case_table_names[=#]</code>	
Option-File Format	<code>lower_case_table_names</code>	
Option Sets Variable	Yes, lower_case_table_names [518]	
Variable Name	<code>lower_case_table_names</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 2</code>

If set to 0, table names are stored as specified and comparisons are case sensitive. If set to 1, table names are stored in lowercase on disk and comparisons are not case sensitive. If set to 2, table names are stored as given but compared in lowercase. This option also applies to database names and table aliases. For additional information, see [Section 9.2.2, “Identifier Case Sensitivity”](#).

You should *not* set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or Mac OS X). If you set this variable to 0 on such a system and access [MyISAM](#) table names using different lettercases, index corruption may result. On Windows the default value is 1. On Mac OS X, the default value is 2.

If you are using [InnoDB](#) tables, you should set this variable to 1 on all platforms to force names to be converted to lowercase.

The setting of this variable has no effect on replication filtering options. This is a known issue which is fixed in MySQL 5.6. See [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#), for more information.

You should not use different settings for `lower_case_table_names` on replication masters and slaves. In particular, you should not do this when the slave uses a case-sensitive file system, as this can cause replication to fail. This is a known issue which is fixed in MySQL 5.6. For more information, see [Section 16.4.1.35, “Replication and Variables”](#).

- [max_allowed_packet \[519\]](#)

Command-Line Format	<code>--max_allowed_packet=#</code>
Option-File Format	<code>max_allowed_packet</code>
Option Sets Variable	Yes, max_allowed_packet [519]
Variable Name	<code>max_allowed_packet</code>
Variable Scope	Global
Dynamic Variable	Yes

The maximum size of one packet or any generated/intermediate string.

The packet message buffer is initialized to [net_buffer_length \[531\]](#) bytes, but can grow up to [max_allowed_packet \[519\]](#) bytes when needed. This value by default is small, to catch large (possibly incorrect) packets.

You must increase this value if you are using large `BLOB` columns or long strings. It should be as big as the largest `BLOB` you want to use. The protocol limit for [max_allowed_packet \[519\]](#) is 1GB. The value should be a multiple of 1024; nonmultiples are rounded down to the nearest multiple.

When you change the message buffer size by changing the value of the [max_allowed_packet \[519\]](#) variable, you should also change the buffer size on the client side if your client program permits it. On the client side, [max_allowed_packet \[519\]](#) has a default of 1GB. Some programs such as `mysql` and `mysqldump` enable you to change the client-side value by setting [max_allowed_packet \[519\]](#) on the command line or in an option file.

The session value of this variable is read only.

- [max_connect_errors \[519\]](#)

Command-Line Format	<code>--max_connect_errors=#</code>
Option-File Format	<code>max_connect_errors</code>
Option Sets Variable	Yes, max_connect_errors [519]
Variable Name	<code>max_connect_errors</code>
Variable Scope	Global
Dynamic Variable	Yes

If more than this many successive connection requests from a host are interrupted without a successful connection, the server blocks that host from further connections. You can unblock blocked hosts by flushing the host cache. To do so, issue a `FLUSH HOSTS` statement or execute a `mysqladmin flush-hosts` command. If a connection is established successfully within fewer than [max_connect_errors \[519\]](#) attempts after a previous connection was interrupted, the error count for the host is cleared to zero. However, once a host is blocked, flushing the host cache is the only way to unblock it.

- [max_connections \[519\]](#)

Command-Line Format	<code>--max_connections=#</code>
Option-File Format	<code>max_connections</code>
Option Sets Variable	Yes, <code>max_connections</code> [519]
Variable Name	<code>max_connections</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	numeric
Default	151
Range	1 .. 100000

The maximum permitted number of simultaneous client connections. By default, this is 151. See [Section C.5.2.7, “Too many connections”](#), for more information.

Increasing this value increases the number of file descriptors that `mysqld` requires. See [Section 8.4.3.1, “How MySQL Opens and Closes Tables”](#), for comments on file descriptor limits.

- `max_delayed_threads` [520]

Command-Line Format	<code>--max_delayed_threads=#</code>
Option-File Format	<code>max_delayed_threads</code>
Option Sets Variable	Yes, <code>max_delayed_threads</code> [520]
Variable Name	<code>max_delayed_threads</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Type	numeric
Default	20
Range	0 .. 16384

Do not start more than this number of threads to handle `INSERT DELAYED` statements. If you try to insert data into a new table after all `INSERT DELAYED` threads are in use, the row is inserted as if the `DELAYED` attribute was not specified. If you set this to 0, MySQL never creates a thread to handle `DELAYED` rows; in effect, this disables `DELAYED` entirely.

For the `SESSION` value of this variable, the only valid values are 0 or the `GLOBAL` value.

- `max_error_count` [520]

Command-Line Format	<code>--max_error_count=#</code>
Option-File Format	<code>max_error_count</code>
Option Sets Variable	Yes, <code>max_error_count</code> [520]
Variable Name	<code>max_error_count</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

	Permitted Values
Type	numeric
Default	64
Range	0 .. 65535

The maximum number of error, warning, and note messages to be stored for display by the `SHOW ERRORS` and `SHOW WARNINGS` statements.

- `max_heap_table_size` [521]

Command-Line Format	<code>--max_heap_table_size=#</code>
Option-File Format	<code>max_heap_table_size</code>
Option Sets Variable	Yes, <code>max_heap_table_size</code> [521]
Variable Name	<code>max_heap_table_size</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	32
Type	numeric
Default	16777216
Range	16384 .. 4294967295
	Permitted Values
Platform Bit Size	64
Type	numeric
Default	16777216
Range	16384 .. 1844674407370954752

This variable sets the maximum size to which user-created `MEMORY` tables are permitted to grow. The value of the variable is used to calculate `MEMORY` table `MAX_ROWS` values. Setting this variable has no effect on any existing `MEMORY` table, unless the table is re-created with a statement such as `CREATE TABLE` or altered with `ALTER TABLE` or `TRUNCATE TABLE`. A server restart also sets the maximum size of existing `MEMORY` tables to the global `max_heap_table_size` [521] value.

This variable is also used in conjunction with `tmp_table_size` [568] to limit the size of internal in-memory tables. See [Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#).

`max_heap_table_size` is not replicated. See [Section 16.4.1.20, “Replication and `MEMORY` Tables”](#), and [Section 16.4.1.35, “Replication and Variables”](#), for more information.

- `max_insert_delayed_threads` [521]

Variable Name	<code>max_insert_delayed_threads</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

	Permitted Values	
	Type	numeric

This variable is a synonym for `max_delayed_threads` [520].

- `max_join_size` [522]

Command-Line Format	<code>--max_join_size=#</code>	
Option-File Format	<code>max_join_size</code>	
Option Sets Variable	Yes, <code>max_join_size</code> [522]	
Variable Name	<code>max_join_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (>= 5.5.0)	
	Type	numeric
	Default	18446744073709551615
	Range	1 .. 18446744073709551615

Do not permit `SELECT` statements that probably need to examine more than `max_join_size` [522] rows (for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than `max_join_size` [522] disk seeks. By setting this value, you can catch `SELECT` statements where keys are not used properly and that would probably take a long time. Set it if your users tend to perform joins that lack a `WHERE` clause, that take a long time, or that return millions of rows.

Setting this variable to a value other than `DEFAULT` resets the value of `sql_big_selects` [555] to 0. If you set the `sql_big_selects` [555] value again, the `max_join_size` [522] variable is ignored.

If a query result is in the query cache, no result size check is performed, because the result has previously been computed and it does not burden the server to send it to the client.

This variable previously was named `sql_max_join_size`.

- `max_length_for_sort_data` [522]

Command-Line Format	<code>--max_length_for_sort_data=#</code>	
Option-File Format	<code>max_length_for_sort_data</code>	
Option Sets Variable	Yes, <code>max_length_for_sort_data</code> [522]	
Variable Name	<code>max_length_for_sort_data</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1024
	Range	4 .. 8388608

The cutoff on the size of index values that determines which `filesort` algorithm to use. See [Section 8.13.9, “ORDER BY Optimization”](#).

- [max_long_data_size \[523\]](#)

Version Introduced	5.5.11	
Version Deprecated	5.5.11	
Command-Line Format	<code>--max_long_data_size=#</code>	
Option-File Format	<code>max_long_data_size</code>	
Option Sets Variable	Yes, max_long_data_size [523]	
Variable Name	<code>max_long_data_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
Deprecated	5.5.11	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>1048576</code>
	Range	<code>1024 .. 4294967295</code>

The maximum size of parameter values that can be sent with the `mysql_stmt_send_long_data()` C API function. If not set at server startup, the default is the value of the [max_allowed_packet \[519\]](#) system variable. This variable is deprecated. In MySQL 5.6, it is removed and the maximum parameter size is controlled by [max_allowed_packet \[519\]](#).

- [max_prepared_stmt_count \[523\]](#)

Command-Line Format	<code>--max_prepared_stmt_count=#</code>	
Option-File Format	<code>max_prepared_stmt_count</code>	
Option Sets Variable	Yes, max_prepared_stmt_count [523]	
Variable Name	<code>max_prepared_stmt_count</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>16382</code>
	Range	<code>0 .. 1048576</code>

This variable limits the total number of prepared statements in the server. (The sum of the number of prepared statements across all sessions.) It can be used in environments where there is the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. If the value is set lower than the current number of prepared statements, existing statements are not affected and can be used, but no new statements can be prepared until the current number drops below the limit. The default value is 16,382. The permissible range of values is from 0 to 1 million. Setting the value to 0 disables prepared statements.

- [max_relay_log_size \[523\]](#)

Command-Line Format	<code>--max_relay_log_size=#</code>
Option-File Format	<code>max_relay_log_size</code>

Option Sets Variable	Yes, max_relay_log_size [523]	
Variable Name	max_relay_log_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 1073741824

If a write by a replication slave to its relay log causes the current log file size to exceed the value of this variable, the slave rotates the relay logs (closes the current file and opens the next one). If [max_relay_log_size \[523\]](#) is 0, the server uses [max_binlog_size \[1903\]](#) for both the binary log and the relay log. If [max_relay_log_size \[523\]](#) is greater than 0, it constrains the size of the relay log, which enables you to have different sizes for the two logs. You must set [max_relay_log_size \[523\]](#) to between 4096 bytes and 1GB (inclusive), or to 0. The default value is 0. See [Section 16.2.1, “Replication Implementation Details”](#).

- [max_seeks_for_key \[524\]](#)

Command-Line Format	<code>--max_seeks_for_key=#</code>	
Option-File Format	max_seeks_for_key	
Option Sets Variable	Yes, max_seeks_for_key [524]	
Variable Name	max_seeks_for_key	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	4294967295
	Range	1 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	18446744073709547520
	Range	1 .. 18446744073709547520

Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index (see [Section 13.7.5.23, “SHOW INDEX Syntax”](#)). By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

- [max_sort_length \[524\]](#)

Command-Line Format	<code>--max_sort_length=#</code>
Option-File Format	<code>max_sort_length</code>
Option Sets Variable	Yes, <code>max_sort_length</code> [524]
Variable Name	<code>max_sort_length</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>1024</code>
	Range <code>4 .. 8388608</code>

The number of bytes to use when sorting data values. Only the first `max_sort_length` [524] bytes of each value are used; the rest are ignored.

- `max_sp_recursion_depth` [525]

Command-Line Format	<code>--max_sp_recursion_depth[=#]</code>
Option-File Format	<code>max_sp_recursion_depth</code>
Option Sets Variable	Yes, <code>max_sp_recursion_depth</code> [525]
Variable Name	<code>max_sp_recursion_depth</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>0</code>
	Max Value <code>255</code>

The number of times that any given stored procedure may be called recursively. The default value for this option is 0, which completely disables recursion in stored procedures. The maximum value is 255.

Stored procedure recursion increases the demand on thread stack space. If you increase the value of `max_sp_recursion_depth` [525], it may be necessary to increase thread stack size by increasing the value of `thread_stack` [566] at server startup.

- `max_tmp_tables` [525]

This variable is unused.

- `max_user_connections` [525]

Command-Line Format	<code>--max_user_connections=#</code>
Option-File Format	<code>max_user_connections</code>
Option Sets Variable	Yes, <code>max_user_connections</code> [525]
Variable Name	<code>max_user_connections</code>

Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 4294967295

The maximum number of simultaneous connections permitted to any given MySQL user account. A value of 0 (the default) means “no limit.”

This variable has a global value that can be set at server startup or runtime. It also has a read-only session value that indicates the effective simultaneous-connection limit that applies to the account associated with the current session. The session value is initialized as follows:

- If the user account has a nonzero `MAX_USER_CONNECTIONS` resource limit, the session `max_user_connections` [525] value is set to that limit.
- Otherwise, the session `max_user_connections` [525] value is set to the global value.

Account resource limits are specified using the `GRANT` statement. See [Section 6.3.4, “Setting Account Resource Limits”](#), and [Section 13.7.1.3, “GRANT Syntax”](#).

- `max_write_lock_count` [526]

Command-Line Format	<code>--max_write_lock_count=#</code>	
Option-File Format	<code>max_write_lock_count</code>	
Option Sets Variable	Yes, <code>max_write_lock_count</code> [526]	
Variable Name	<code>max_write_lock_count</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	4294967295
	Range	1 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	18446744073709547520
	Range	1 .. 18446744073709547520

After this many write locks, permit some pending read lock requests to be processed in between.

- `metadata_locks_cache_size` [526]

Version Introduced	5.5.19	
Variable Name	<code>metadata_locks_cache_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>1024</code>
	Range	<code>1 .. 1048576</code>

The size of the metadata locks cache. The server uses this cache to avoid creation and destruction of synchronization objects. This is particularly helpful on systems where such operations are expensive, such as Windows XP. This variable was added in MySQL 5.5.19.

- `min_examined_row_limit` [527]

Command-Line Format	<code>--min-examined-row-limit=#</code>	
Option-File Format	<code>min-examined-row-limit</code>	
Variable Name	<code>min_examined_row_limit</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	<code>32</code>
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 4294967295</code>
	Permitted Values	
	Platform Bit Size	<code>64</code>
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 18446744073709547520</code>

Queries that examine fewer than this number of rows are not logged to the slow query log.

- `myisam_data_pointer_size` [527]

Command-Line Format	<code>--myisam_data_pointer_size=#</code>	
Option-File Format	<code>myisam_data_pointer_size</code>	
Option Sets Variable	Yes, <code>myisam_data_pointer_size</code> [527]	
Variable Name	<code>myisam_data_pointer_size</code>	
Variable Scope	Global	
Dynamic Variable	Yes	

	Permitted Values	
Type	numeric	
Default	6	
Range	2 .. 7	

The default pointer size in bytes, to be used by `CREATE TABLE` for `MyISAM` tables when no `MAX_ROWS` option is specified. This variable cannot be less than 2 or larger than 7. The default value is 6. See [Section C.5.2.12, “The table is full”](#).

- `myisam_max_sort_file_size` [528]

Command-Line Format	<code>--myisam_max_sort_file_size=#</code>	
Option-File Format	<code>myisam_max_sort_file_size</code>	
Option Sets Variable	Yes, <code>myisam_max_sort_file_size</code> [528]	
Variable Name	<code>myisam_max_sort_file_size</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
Type	numeric	
Default	2147483648	

The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

The default value is 2GB. If `MyISAM` index files exceed this size and disk space is available, increasing the value may help performance. The space must be available in the file system containing the directory where the original index file is located.

- `myisam_mmap_size` [528]

Version Introduced	5.5.1	
Command-Line Format	<code>--myisam_mmap_size=#</code>	
Option-File Format	<code>myisam_mmap_size</code>	
Option Sets Variable	Yes, <code>myisam_mmap_size</code> [528]	
Variable Name	<code>myisam_mmap_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
Platform Bit Size	32	
Type	numeric	
Default	4294967295	
Range	7 .. 4294967295	

	Permitted Values
Platform Bit Size	64
Type	numeric
Default	18446744073709547520
Range	7 .. 18446744073709547520

The maximum amount of memory to use for memory mapping compressed `MyISAM` files. If many compressed `MyISAM` tables are used, the value can be decreased to reduce the likelihood of memory-swapping problems. This variable was added in MySQL 5.5.1.

- `myisam_recover_options` [529]

Variable Name	<code>myisam_recover_options</code>
Variable Scope	Global
Dynamic Variable	No

The value of the `--myisam-recover-options` [454] option. See [Section 5.1.3, “Server Command Options”](#).

- `myisam_repair_threads` [529]

Command-Line Format	<code>--myisam_repair_threads=#</code>
Option-File Format	<code>myisam_repair_threads</code>
Option Sets Variable	Yes, <code>myisam_repair_threads</code> [529]
Variable Name	<code>myisam_repair_threads</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	32
Type	numeric
Default	1
Range	1 .. 4294967295
	Permitted Values
Platform Bit Size	64
Type	numeric
Default	1
Range	1 .. 18446744073709547520

If this value is greater than 1, `MyISAM` table indexes are created in parallel (each index in its own thread) during the `Repair by sorting` process. The default value is 1.

**Note**Multi-threaded repair is still *beta-quality* code.

- `myisam_sort_buffer_size` [530]

Command-Line Format	<code>--myisam_sort_buffer_size=#</code>
Option-File Format	<code>myisam_sort_buffer_size</code>
Option Sets Variable	Yes, <code>myisam_sort_buffer_size</code> [530]
Variable Name	<code>myisam_sort_buffer_size</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	32
Type	numeric
Default	8388608
Range	4 .. 4294967295
	Permitted Values
Platform Bit Size	64
Type	numeric
Default	8388608
Range	4 .. 18446744073709547520

The size of the buffer that is allocated when sorting MyISAM indexes during a `REPAIR TABLE` or when creating indexes with `CREATE INDEX` or `ALTER TABLE`.

The maximum permissible setting for `myisam_sort_buffer_size` [530] is 4GB. Values larger than 4GB are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB with a warning).

- `myisam_stats_method` [530]

Command-Line Format	<code>--myisam_stats_method=name</code>
Option-File Format	<code>myisam_stats_method</code>
Option Sets Variable	Yes, <code>myisam_stats_method</code> [530]
Variable Name	<code>myisam_stats_method</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Type	enumeration
Valid Values	<code>nulls_equal</code> <code>nulls_unequal</code>

	<code>nulls_ignored</code>
--	----------------------------

How the server treats `NULL` values when collecting statistics about the distribution of index values for `MyISAM` tables. This variable has three possible values, `nulls_equal`, `nulls_unequal`, and `nulls_ignored`. For `nulls_equal`, all `NULL` index values are considered equal and form a single value group that has a size equal to the number of `NULL` values. For `nulls_unequal`, `NULL` values are considered unequal, and each `NULL` forms a distinct value group of size 1. For `nulls_ignored`, `NULL` values are ignored.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in [Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”](#).

- `myisam_use_mmap` [\[531\]](#)

Command-Line Format	<code>--myisam_use_mmap</code>
Option-File Format	<code>myisam_use_mmap</code>
Option Sets Variable	Yes, <code>myisam_use_mmap</code> [531]
Variable Name	<code>myisam_use_mmap</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>boolean</code>
	Default <code>OFF</code>

Use memory mapping for reading and writing `MyISAM` tables.

- `named_pipe` [\[531\]](#)

Variable Name	<code>named_pipe</code>
Variable Scope	Global
Dynamic Variable	No
Platform Specific	windows
	Permitted Values
	Type <code>boolean</code> (windows)
	Default <code>OFF</code>

(Windows only.) Indicates whether the server supports connections over named pipes.

- `net_buffer_length` [\[531\]](#)

Command-Line Format	<code>--net_buffer_length=#</code>
Option-File Format	<code>net_buffer_length</code>
Option Sets Variable	Yes, <code>net_buffer_length</code> [531]
Variable Name	<code>net_buffer_length</code>
Variable Scope	Global, Session

Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	16384
	Range	1024 .. 1048576

Each client thread is associated with a connection buffer and result buffer. Both begin with a size given by `net_buffer_length` [531] but are dynamically enlarged up to `max_allowed_packet` [519] bytes as needed. The result buffer shrinks to `net_buffer_length` [531] after each SQL statement.

This variable should not normally be changed, but if you have very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the connection buffer is automatically enlarged. The maximum value to which `net_buffer_length` [531] can be set is 1MB.

The session value of this variable is read only.

- `net_read_timeout` [532]

Command-Line Format	<code>--net_read_timeout=#</code>	
Option-File Format	<code>net_read_timeout</code>	
Option Sets Variable	Yes, <code>net_read_timeout</code> [532]	
Variable Name	<code>net_read_timeout</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	30
	Min Value	1

The number of seconds to wait for more data from a connection before aborting the read. When the server is reading from the client, `net_read_timeout` [532] is the timeout value controlling when to abort. When the server is writing to the client, `net_write_timeout` [533] is the timeout value controlling when to abort. See also `slave_net_timeout` [1888].

- `net_retry_count` [532]

Command-Line Format	<code>--net_retry_count=#</code>	
Option-File Format	<code>net_retry_count</code>	
Option Sets Variable	Yes, <code>net_retry_count</code> [532]	
Variable Name	<code>net_retry_count</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32

	Type	numeric
	Default	10
	Range	1 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	10
	Range	1 .. 18446744073709547520

If a read or write on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads.

- [net_write_timeout \[533\]](#)

Command-Line Format	--net_write_timeout=#	
Option-File Format	net_write_timeout	
Option Sets Variable	Yes, net_write_timeout [533]	
Variable Name	net_write_timeout	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	60
	Min Value	1

The number of seconds to wait for a block to be written to a connection before aborting the write. See also [net_read_timeout \[532\]](#).

- [new \[533\]](#)

Command-Line Format	--new	
	-n	
Option-File Format	new	
Option Sets Variable	Yes, new [533]	
Variable Name	new	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Disabled by	skip-new	
	Permitted Values	
	Type	boolean
	Default	FALSE

This variable was used in MySQL 4.0 to turn on some 4.1 behaviors, and is retained for backward compatibility. In MySQL 5.5, its value is always `OFF`.

- `old` [534]

Command-Line Format	<code>--old</code>
Option-File Format	<code>old</code>
Variable Name	<code>old</code>
Variable Scope	Global
Dynamic Variable	No

`old` [534] is a compatibility variable. It is disabled by default, but can be enabled at startup to revert the server to behaviors present in older versions.

Currently, when `old` [534] is enabled, it changes the default scope of index hints to that used prior to MySQL 5.1.17. That is, index hints with no `FOR` clause apply only to how indexes are used for row retrieval and not to resolution of `ORDER BY` or `GROUP BY` clauses. (See [Section 13.2.9.3, “Index Hint Syntax”](#).) Take care about enabling this in a replication setup. With statement-based binary logging, having different modes for the master and slaves might lead to replication errors.

- `old_alter_table` [534]

Command-Line Format	<code>--old-alter-table</code>
Option-File Format	<code>old-alter-table</code>
Option Sets Variable	Yes, <code>old_alter_table</code> [534]
Variable Name	<code>old_alter_table</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type <code>boolean</code>
	Default <code>OFF</code>

When this variable is enabled, the server does not use the optimized method of processing an `ALTER TABLE` operation. It reverts to using a temporary table, copying over the data, and then renaming the temporary table to the original, as used by MySQL 5.0 and earlier. For more information on the operation of `ALTER TABLE`, see [Section 13.1.7, “ALTER TABLE Syntax”](#).

- `old_passwords` [534]

Variable Name	<code>old_passwords</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

This variable determines the type of password hashing performed by the `PASSWORD()` [1188] function and statements such as `CREATE USER` and `SET PASSWORD`. The following table shows the permitted values of `old_passwords` [534], as well as the type of password hashing each value produces, and the plugins that use each hashing type.

Value	Password Hashing Format	Intended Use
0 or OFF	MySQL 4.1 native hashing	Accounts that authenticate with the <code>mysql_native_password</code> plugin
1 or ON	Pre-4.1 (“old”) native hashing	Accounts that authenticate with the <code>mysql_old_password</code> plugin

If `old_passwords=1`, `PASSWORD('str')` [1188] returns the same value as `OLD_PASSWORD('str')` [1187].

For information about authentication plugins and hashing formats, see [Section 6.3.6, “Pluggable Authentication”](#), and [Section 6.1.2.4, “Password Hashing in MySQL”](#).

- `one_shot` [535]

This is not a variable, but it can be used when setting some variables. It is described in [Section 13.7.4, “SET Syntax”](#).

- `open_files_limit` [535]

Command-Line Format	<code>--open-files-limit=#</code>
Option-File Format	<code>open-files-limit</code>
Option Sets Variable	Yes, <code>open_files_limit</code> [535]
Variable Name	<code>open_files_limit</code>
Variable Scope	Global
Dynamic Variable	No

The number of files that the operating system permits `mysqld` to open. This is the real value permitted by the system and might be different from the value you gave using the `--open-files-limit` [456] option to `mysqld` or `mysqld_safe`. The value is 0 on systems where MySQL cannot change the number of open files.

- `optimizer_prune_level` [535]

Command-Line Format	<code>--optimizer_prune_level[=#]</code>
Option-File Format	<code>optimizer_prune_level</code>
Option Sets Variable	Yes, <code>optimizer_prune_level</code> [535]
Variable Name	<code>optimizer_prune_level</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type <code>boolean</code>
	Default <code>1</code>

Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space. A value of 0 disables heuristics so that the optimizer performs an exhaustive search. A value of 1 causes the optimizer to prune plans based on the number of rows retrieved by intermediate plans.

- [optimizer_search_depth \[536\]](#)

Command-Line Format	<code>--optimizer_search_depth[=#]</code>	
Option-File Format	<code>optimizer_search_depth</code>	
Option Sets Variable	Yes, optimizer_search_depth [536]	
Variable Name	<code>optimizer_search_depth</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (<= 5.5.99)	
	Type	<code>numeric</code>
	Default	<code>62</code>
	Range	<code>0 .. 63</code>

The maximum depth of search performed by the query optimizer. Values larger than the number of relations in a query result in better query plans, but take longer to generate an execution plan for a query. Values smaller than the number of relations in a query return an execution plan quicker, but the resulting plan may be far from being optimal. If set to 0, the system automatically picks a reasonable value. If set to 63, the optimizer switches to the algorithm used in MySQL 5.0.0 (and previous versions) for performing searches. The value of 63 is deprecated and will be treated as invalid in a future MySQL release.

- [optimizer_switch \[536\]](#)

Command-Line Format	<code>--optimizer_switch=value</code>	
Option-File Format	<code>optimizer_switch</code>	
	<code>optimizer_switch</code>	
	<code>optimizer_switch</code>	
Option Sets Variable	Yes, optimizer_switch [536]	
Variable Name	<code>optimizer_switch</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (<= 5.5.2)	
	Type	<code>set</code>
	Valid Values	<code>index_merge={on off}</code> <code>index_merge_intersection={on off}</code> <code>index_merge_sort_union={on off}</code> <code>index_merge_union={on off}</code>
	Permitted Values (>= 5.5.3)	
	Type	<code>set</code>
	Valid Values	<code>engine_condition_pushdown={on off}</code> <code>index_merge={on off}</code>

	<pre>index_merge_intersection={on off} index_merge_sort_union={on off} index_merge_union={on off}</pre>
--	---

The `optimizer_switch` [536] system variable enables control over optimizer behavior. The value of this variable is a set of flags, each of which has a value of `on` or `off` to indicate whether the corresponding optimizer behavior is enabled or disabled. This variable has global and session values and can be changed at runtime. The global default can be set at server startup.

To see the current set of optimizer flags, select the variable value:

```
mysql> SELECT @@optimizer_switch\G
***** 1. row *****
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on
```

For more information about the syntax of this variable and the optimizer behaviors that it controls, see [Section 8.8.5.2, “Controlling Switchable Optimizations”](#).

- `performance_schema_XXX`

Performance Schema system variables are listed in [Section 21.8, “Performance Schema System Variables”](#).

- `pid_file` [537]

Command-Line Format	<code>--pid-file=file_name</code>	
Option-File Format	<code>pid-file=file_name</code>	
Option Sets Variable	Yes, <code>pid_file</code> [537]	
Variable Name	<code>pid_file</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The path name of the process ID (PID) file. This variable can be set with the `--pid-file` [456] option.

- `plugin_dir` [537]

Command-Line Format	<code>--plugin_dir=path</code>	
Option-File Format	<code>plugin_dir</code>	
Option Sets Variable	Yes, <code>plugin_dir</code> [537]	
Variable Name	<code>plugin_dir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values (>= 5.5.0, <= 5.5.4)	

	Type (other)	directory name
	Default	BASEDIR/lib/mysql/plugin
	Permitted Values (>= 5.5.0, <= 5.5.4)	
	Type (windows)	directory name
	Default	BASEDIR/lib/plugin
	Permitted Values (>= 5.5.5)	
	Type	directory name
	Default	BASEDIR/lib/plugin

The path name of the plugin directory.

If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` [537] read only to the server or by setting `--secure-file-priv` [460] to a directory where `SELECT` writes can be made safely.

- `port` [538]

Command-Line Format	<code>--port=#</code>	
	<code>-P</code>	
Option-File Format	<code>port</code>	
Option Sets Variable	Yes, <code>port</code> [538]	
Variable Name	<code>port</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	3306

The number of the port on which the server listens for TCP/IP connections. This variable can be set with the `--port` [458] option.

- `preload_buffer_size` [538]

Command-Line Format	<code>--preload_buffer_size=#</code>	
Option-File Format	<code>preload_buffer_size</code>	
Option Sets Variable	Yes, <code>preload_buffer_size</code> [538]	
Variable Name	<code>preload_buffer_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	32768

	Range	1024 .. 1073741824
--	--------------	--------------------

The size of the buffer that is allocated when preloading indexes.

- [profiling \[539\]](#)

If set to 0 or `OFF` (the default), statement profiling is disabled. If set to 1 or `ON`, statement profiling is enabled and the `SHOW PROFILE` and `SHOW PROFILES` statements provide access to profiling information. See [Section 13.7.5.32, “SHOW PROFILES Syntax”](#).

- [profiling_history_size \[539\]](#)

The number of statements for which to maintain profiling information if [profiling \[539\]](#) is enabled. The default value is 15. The maximum value is 100. Setting the value to 0 effectively disables profiling. See [Section 13.7.5.32, “SHOW PROFILES Syntax”](#).

- [protocol_version \[539\]](#)

Variable Name	<code>protocol_version</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>

The version of the client/server protocol used by the MySQL server.

- [proxy_user \[539\]](#)

Version Introduced	5.5.7	
Variable Name	<code>proxy_user</code>	
Variable Scope	Session	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

If the current client is a proxy for another user, this variable is the proxy user account name. Otherwise, this variable is `NULL`. See [Section 6.3.7, “Proxy Users”](#).

This variable was added in MySQL 5.5.7.

- [pseudo_thread_id \[539\]](#)

Variable Name	<code>pseudo_thread_id</code>	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>

This variable is for internal server use.

- [query_alloc_block_size \[539\]](#)

Command-Line Format	<code>--query_alloc_block_size=#</code>
Option-File Format	<code>query_alloc_block_size</code>
Option Sets Variable	Yes, <code>query_alloc_block_size</code> [539]
Variable Name	<code>query_alloc_block_size</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	32
Type	numeric
Default	8192
Range	1024 .. 4294967295
Block Size	1024
	Permitted Values
Platform Bit Size	64
Type	numeric
Default	8192
Range	1024 .. 18446744073709547520
Block Size	1024

The allocation size of memory blocks that are allocated for objects created during statement parsing and execution. If you have problems with memory fragmentation, it might help to increase this parameter.

- `query_cache_limit` [540]

Command-Line Format	<code>--query_cache_limit=#</code>
Option-File Format	<code>query_cache_limit</code>
Option Sets Variable	Yes, <code>query_cache_limit</code> [540]
Variable Name	<code>query_cache_limit</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	32
Type	numeric
Default	1048576
Range	0 .. 4294967295
	Permitted Values

Platform Bit Size	64
Type	numeric
Default	1048576
Range	0 .. 18446744073709547520

Do not cache results that are larger than this number of bytes. The default value is 1MB.

- [query_cache_min_res_unit \[541\]](#)

Command-Line Format	<code>--query_cache_min_res_unit=#</code>
Option-File Format	<code>query_cache_min_res_unit</code>
Option Sets Variable	Yes, query_cache_min_res_unit [541]
Variable Name	<code>query_cache_min_res_unit</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Platform Bit Size 32
	Type numeric
	Default 4096
	Range 512 .. 4294967295
	Permitted Values
	Platform Bit Size 64
	Type numeric
	Default 4096
	Range 512 .. 18446744073709547520

The minimum size (in bytes) for blocks allocated by the query cache. The default value is 4096 (4KB). Tuning information for this variable is given in [Section 8.9.3.3, “Query Cache Configuration”](#).

- [query_cache_size \[541\]](#)

Command-Line Format	<code>--query_cache_size=#</code>
Option-File Format	<code>query_cache_size</code>
Option Sets Variable	Yes, query_cache_size [541]
Variable Name	<code>query_cache_size</code>
Variable Scope	Global
Dynamic Variable	Yes

The amount of memory allocated for caching query results. The default value is 0, which disables the query cache. To reduce overhead significantly, you should also start the server with [query_cache_type=0 \[542\]](#) if you will not be using the query cache. The permissible values are multiples of 1024; other values are rounded down to the nearest multiple. Note that

`query_cache_size` [541] bytes of memory are allocated even if `query_cache_type` [542] is set to 0. See Section 8.9.3.3, “Query Cache Configuration”, for more information.

The query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value of `query_cache_size` [541] too small, a warning will occur, as described in Section 8.9.3.3, “Query Cache Configuration”.

- `query_cache_type` [542]

Command-Line Format	<code>--query_cache_type=#</code>
Option-File Format	<code>query_cache_type</code>
Option Sets Variable	Yes, <code>query_cache_type</code> [542]
Variable Name	<code>query_cache_type</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

Set the query cache type. Setting the `GLOBAL` value sets the type for all clients that connect thereafter. Individual clients can set the `SESSION` value to affect their own use of the query cache. Possible values are shown in the following table.

Option	Description
0 or <code>OFF</code>	Do not cache results in or retrieve results from the query cache. Note that this does not deallocate the query cache buffer. To do that, you should set <code>query_cache_size</code> [541] to 0.
1 or <code>ON</code>	Cache all cacheable query results except for those that begin with <code>SELECT SQL_NO_CACHE</code> .
2 or <code>DEMAND</code>	Cache results only for cacheable queries that begin with <code>SELECT SQL_CACHE</code> .

This variable defaults to `ON`.

If the server is started with `query_cache_type` set to 0, it does not acquire the query cache mutex at all, which means that the query cache cannot be enabled at runtime and there is reduced overhead in query execution.

- `query_cache_wlock_invalidate` [542]

Command-Line Format	<code>--query_cache_wlock_invalidate</code>	
Option-File Format	<code>query_cache_wlock_invalidate</code>	
Option Sets Variable	Yes, <code>query_cache_wlock_invalidate</code> [542]	
Variable Name	<code>query_cache_wlock_invalidate</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Normally, when one client acquires a `WRITE` lock on a `MyISAM` table, other clients are not blocked from issuing statements that read from the table if the query results are present in the query cache. Setting

this variable to 1 causes acquisition of a [WRITE](#) lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect.

- [query_prealloc_size](#) [543]

Command-Line Format	<code>--query_prealloc_size=#</code>	
Option-File Format	<code>query_prealloc_size</code>	
Option Sets Variable	Yes, query_prealloc_size [543]	
Variable Name	<code>query_prealloc_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	8192
	Range	8192 .. 4294967295
	Block Size	1024
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	8192
	Range	8192 .. 18446744073709547520
	Block Size	1024

The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger [query_prealloc_size](#) [543] value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

- [rand_seed1](#) [543]

The [rand_seed1](#) [543] and [rand_seed2](#) [543] variables exist as session variables only, and can be set but not read. The variables—but not their values—are shown in the output of `SHOW VARIABLES`.

The purpose of these variables is to support replication of the `RAND()` [1128] function. For statements that invoke `RAND()` [1128], the master passes two values to the slave, where they are used to seed the random number generator. The slave uses these values to set the session variables [rand_seed1](#) [543] and [rand_seed2](#) [543] so that `RAND()` [1128] on the slave generates the same value as on the master.

- [rand_seed2](#) [543]

See the description for [rand_seed1](#) [543].

- [range_alloc_block_size \[544\]](#)

Command-Line Format	<code>--range_alloc_block_size=#</code>	
Option-File Format	<code>range_alloc_block_size</code>	
Option Sets Variable	Yes, range_alloc_block_size [544]	
Variable Name	<code>range_alloc_block_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	4096
	Range	4096 .. 4294967295
	Block Size	1024

The size of blocks that are allocated when doing range optimization.

- [read_buffer_size \[544\]](#)

Command-Line Format	<code>--read_buffer_size=#</code>	
Option-File Format	<code>read_buffer_size</code>	
Option Sets Variable	Yes, read_buffer_size [544]	
Variable Name	<code>read_buffer_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	131072
	Range	8200 .. 2147479552

Each thread that does a sequential scan for a [MyISAM](#) table allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you might want to increase this value, which defaults to 131072. The value of this variable should be a multiple of 4KB. If it is set to a value that is not a multiple of 4KB, its value will be rounded down to the nearest multiple of 4KB.

The maximum permissible setting for [read_buffer_size \[544\]](#) is 2GB.

For more information about memory use during different operations, see [Section 8.11.4.1, “How MySQL Uses Memory”](#).

- [read_only \[544\]](#)

Command-Line Format	<code>--read-only</code>	
Option-File Format	<code>read_only</code>	544

Option Sets Variable	Yes, <code>read_only</code> [544]	
Variable Name	<code>read_only</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>false</code>

This variable is off by default. When it is enabled, the server permits no updates except from users that have the `SUPER` [697] privilege or (on a slave server) from updates performed by slave threads. In replication setups, it can be useful to enable `read_only` [544] on slave servers to ensure that slaves accept updates only from the master server and not from clients.

`read_only` [544] does not apply to `TEMPORARY` tables, nor does it prevent the server from inserting rows into the log tables (see Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”). This variable does not prevent the use of `ANALYZE TABLE` or `OPTIMIZE TABLE` statements because its purpose is to prevent changes to table structure or contents. Analysis and optimization do not qualify as such changes. This means, for example, that consistency checks on read-only slaves can be performed with `mysqlcheck --all-databases --analyze`.

`read_only` [544] exists only as a `GLOBAL` variable, so changes to its value require the `SUPER` [697] privilege. Changes to `read_only` [544] on a master server are not replicated to slave servers. The value can be set on a slave server independent of the setting on the master.



Important

In MySQL 5.5, enabling `read_only` prevents the use of the `SET PASSWORD` statement by any user not having the `SUPER` [697] privilege. This is not necessarily the case for all MySQL release series. When replicating from one MySQL release series to another (for example, from a MySQL 5.0 master to a MySQL 5.1 or later slave), you should check the documentation for the versions running on both master and slave to determine whether the behavior of `read_only` in this regard is or is not the same, and, if it is different, whether this has an impact on your applications.

The following conditions apply:

- If you attempt to enable `read_only` [544] while you have any explicit locks (acquired with `LOCK TABLES`) or have a pending transaction, an error occurs.
- If you attempt to enable `read_only` [544] while other clients hold explicit table locks or have pending transactions, the attempt blocks until the locks are released and the transactions end. While the attempt to enable `read_only` [544] is pending, requests by other clients for table locks or to begin transactions also block until `read_only` [544] has been set.
- `read_only` [544] can be enabled while you hold a global read lock (acquired with `FLUSH TABLES WITH READ LOCK`) because that does not involve table locks.

As of MySQL 5.5.3, attempts to set `read_only` [544] block for active transactions that hold metadata locks until those transactions end.

- `read_rnd_buffer_size` [545]

Command-Line Format	<code>--read_rnd_buffer_size=#</code>
Option-File Format	<code>read_rnd_buffer_size</code>
Option Sets Variable	Yes, <code>read_rnd_buffer_size</code> [545]
Variable Name	<code>read_rnd_buffer_size</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type numeric
	Default 262144
	Range 8200 .. 4294967295

When reading rows from a `MyISAM` table in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks. See [Section 8.13.9, “ORDER BY Optimization”](#). Setting the variable to a large value can improve `ORDER BY` performance by a lot. However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries.

The maximum permissible setting for `read_rnd_buffer_size` [545] is 2GB.

For more information about memory use during different operations, see [Section 8.11.4.1, “How MySQL Uses Memory”](#).

- `relay_log_purge` [546]

Command-Line Format	<code>--relay_log_purge</code>
Option-File Format	<code>relay_log_purge</code>
Option Sets Variable	Yes, <code>relay_log_purge</code> [546]
Variable Name	<code>relay_log_purge</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type boolean
	Default TRUE

Disables or enables automatic purging of relay log files as soon as they are not needed any more. The default value is 1 (`ON`).

- `relay_log_space_limit` [546]

Command-Line Format	<code>--relay_log_space_limit=#</code>
Option-File Format	<code>relay_log_space_limit</code>
Option Sets Variable	Yes, <code>relay_log_space_limit</code> [546]
Variable Name	<code>relay_log_space_limit</code>
Variable Scope	Global
Dynamic Variable	No

	Permitted Values
Platform Bit Size	32
Type	numeric
Default	0
Range	0 .. 4294967295
	Permitted Values
Platform Bit Size	64
Type	numeric
Default	0
Range	0 .. 18446744073709547520

The maximum amount of space to use for all relay logs.

- [report_host \[547\]](#)

Command-Line Format	<code>--report-host=host_name</code>
Option-File Format	<code>report-host</code>
Option Sets Variable	Yes, report_host [547]
Variable Name	<code>report-host</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
Type	string

The value of the `--report-host [1880]` option.

- [report_password \[547\]](#)

Command-Line Format	<code>--report-password=name</code>
Option-File Format	<code>report-password</code>
Option Sets Variable	Yes, report_password [547]
Variable Name	<code>report-password</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
Type	string

The value of the `--report-password [1880]` option. Not the same as the password used for the MySQL replication user account.

- [report_port \[547\]](#)

Command-Line Format	<code>--report-port=#</code>
----------------------------	------------------------------

Option-File Format	<code>report-port</code>
Option Sets Variable	Yes, <code>report_port</code> [547]
Variable Name	<code>report-port</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values (>= 5.5.23)
Type	<code>numeric</code>
Default	<code>0</code>

The value of the `--report-port` [1881] option.

- `report_user` [548]

Command-Line Format	<code>--report-user=name</code>
Option-File Format	<code>report-user</code>
Option Sets Variable	Yes, <code>report_user</code> [548]
Variable Name	<code>report-user</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
Type	<code>string</code>

The value of the `--report-user` [1881] option. Not the same as the name for the MySQL replication user account.

- `rpl_semi_sync_master_enabled` [548]

Variable Name	<code>rpl_semi_sync_master_enabled</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	<code>boolean</code>
Default	<code>OFF</code>

Controls whether semisynchronous replication is enabled on the master. To enable or disable the plugin, set this variable to `ON` or `OFF` (or 1 or 0), respectively. The default is `OFF`.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_timeout` [548]

Variable Name	<code>rpl_semi_sync_master_timeout</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values (>= 5.5.0)
Type	<code>numeric</code>

	Default	10000
--	----------------	-------

A value in milliseconds that controls how long the master waits on a commit for acknowledgment from a slave before timing out and reverting to asynchronous replication. The default value is 10000 (10 seconds).

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [rpl_semi_sync_master_trace_level \[549\]](#)

Variable Name	rpl_semi_sync_master_trace_level	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	32

The semisynchronous replication debug trace level on the master. Currently, four levels are defined:

- 1 = general level (for example, time function failures)
- 16 = detail level (more verbose information)
- 32 = net wait level (more information about network waits)
- 64 = function level (information about function entry and exit)

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [rpl_semi_sync_master_wait_no_slave \[549\]](#)

Variable Name	rpl_semi_sync_master_wait_no_slave	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

With semisynchronous replication, for each transaction, the master waits until timeout for acknowledgment of receipt from some semisynchronous slave. If no response occurs during this period, the master reverts to normal replication. This variable controls whether the master waits for the timeout to expire before reverting to normal replication even if the slave count drops to zero during the timeout period.

If the value is `ON` (the default), it is permissible for the slave count to drop to zero during the timeout period (for example, if slaves disconnect). The master still waits for the timeout, so as long as some slave reconnects and acknowledges the transaction within the timeout interval, semisynchronous replication continues.

If the value is `OFF`, the master reverts to normal replication if the slave count drops to zero during the timeout period.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [rpl_semi_sync_slave_enabled \[550\]](#)

Variable Name	<code>rpl_semi_sync_slave_enabled</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Controls whether semisynchronous replication is enabled on the slave. To enable or disable the plugin, set this variable to `ON` or `OFF` (or 1 or 0), respectively. The default is `OFF`.

This variable is available only if the slave-side semisynchronous replication plugin is installed.

- [rpl_semi_sync_slave_trace_level \[550\]](#)

Variable Name	<code>rpl_semi_sync_slave_trace_level</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>32</code>

The semisynchronous replication debug trace level on the slave. See [rpl_semi_sync_master_trace_level \[549\]](#) for the permissible values.

This variable is available only if the slave-side semisynchronous replication plugin is installed.

- [secure_auth \[550\]](#)

Command-Line Format	<code>--secure-auth</code>
Option-File Format	<code>secure-auth</code>
Option Sets Variable	Yes, secure_auth [550]
Variable Name	<code>secure_auth</code>
Variable Scope	Global
Dynamic Variable	Yes

If this variable is enabled, the server blocks connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format.

Enable this variable to prevent all use of passwords employing the old format (and hence insecure communication over the network).

Server startup fails with an error if this variable is enabled and the privilege tables are in pre-4.1 format. See [Section C.5.2.4, "Client does not support authentication protocol"](#).

- [secure_file_priv \[550\]](#)

Command-Line Format	<code>--secure-file-priv=path</code>	
Option-File Format	<code>secure-file-priv=path</code>	
Option Sets Variable	Yes, <code>secure_file_priv</code> [550]	
Variable Name	<code>secure-file-priv</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

By default, this variable is empty. If set to the name of a directory, it limits the effect of the `LOAD_FILE()` [1105] function and the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements to work only with files in that directory.

- `server_id` [551]

Command-Line Format	<code>--server-id=#</code>	
Option-File Format	<code>server-id</code>	
Option Sets Variable	Yes, <code>server_id</code> [551]	
Variable Name	<code>server_id</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 4294967295</code>

The server ID, used in replication to give each master and slave a unique identity. This variable is set by the `--server-id` [1858] option. For each server participating in replication, you should pick a positive integer in the range from 1 to $2^{32} - 1$ to act as that server's ID.

- `shared_memory` [551]

Variable Name	<code>shared_memory</code>
Variable Scope	Global
Dynamic Variable	No
Platform Specific	windows

(Windows only.) Whether the server permits shared-memory connections.

- `shared_memory_base_name` [551]

Variable Name	<code>shared_memory_base_name</code>
Variable Scope	Global
Dynamic Variable	No
Platform Specific	windows

(Windows only.) The name of shared memory to use for shared-memory connections. This is useful when running multiple MySQL instances on a single physical machine. The default name is `MYSQL`. The name is case sensitive.

- `skip_external_locking` [552]

This is `OFF` if `mysqld` uses external locking, `ON` if external locking is disabled. This affects only `MyISAM` table access.

- `skip_name_resolve` [552]

This variable is set from the value of the `--skip-name-resolve` [461] option. If it is `ON`, `mysqld` resolves host names when checking client connections. If `OFF`, `mysqld` uses only IP numbers and all `Host` column values in the grant tables must be IP addresses or `localhost`. See Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”.

This variable was added in MySQL 5.5.5.

- `skip_networking` [552]

This is `ON` if the server permits only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe or shared memory. This variable can be set to `ON` with the `--skip-networking` [461] option.

- `skip_show_database` [552]

This prevents people from using the `SHOW DATABASES` statement if they do not have the `SHOW DATABASES` [697] privilege. This can improve security if you have concerns about users being able to see databases belonging to other users. Its effect depends on the `SHOW DATABASES` [697] privilege: If the variable value is `ON`, the `SHOW DATABASES` statement is permitted only to users who have the `SHOW DATABASES` [697] privilege, and the statement displays all database names. If the value is `OFF`, `SHOW DATABASES` is permitted to all users, but displays the names of only those databases for which the user has the `SHOW DATABASES` [697] or other privilege.

- `slow_launch_time` [552]

Command-Line Format	<code>--slow_launch_time=#</code>	
Option-File Format	<code>slow_launch_time</code>	
Option Sets Variable	Yes, <code>slow_launch_time</code> [552]	
Variable Name	<code>slow_launch_time</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>2</code>

If creating a thread takes longer than this many seconds, the server increments the `Slow_launch_threads` [608] status variable.

- `slow_query_log` [552]

Command-Line Format	<code>--slow-query-log</code>
----------------------------	-------------------------------

Option-File Format	slow-query-log	
Option Sets Variable	Yes, slow_query_log [552]	
Variable Name	slow_query_log	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Whether the slow query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The default value depends on whether the `--slow_query_log` [552] option is given. The destination for log output is controlled by the `log_output` [515] system variable; if that value is `NONE`, no log entries are written even if the log is enabled.

“Slow” is determined by the value of the `long_query_time` [517] variable. See [Section 5.2.5, “The Slow Query Log”](#).

- [slow_query_log_file](#) [553]

Command-Line Format	<code>--slow-query-log-file=file_name</code>	
Option-File Format	slow_query_log_file	
Option Sets Variable	Yes, slow_query_log_file [553]	
Variable Name	slow_query_log_file	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	file name

The name of the slow query log file. The default value is `host_name-slow.log`, but the initial value can be changed with the `--slow_query_log_file` option.

- [socket](#) [553]

Command-Line Format	<code>--socket=name</code>	
Option-File Format	socket	
Option Sets Variable	Yes, socket [553]	
Variable Name	socket	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name
	Default	<code>/tmp/mysql.sock</code>

On Unix platforms, this variable is the name of the socket file that is used for local client connections. The default is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On Windows, this variable is the name of the named pipe that is used for local client connections. The default value is `MySQL` (not case sensitive).

- `sort_buffer_size` [554]

Command-Line Format	<code>--sort_buffer_size=#</code>
Option-File Format	<code>sort_buffer_size</code>
Option Sets Variable	Yes, <code>sort_buffer_size</code> [554]
Variable Name	<code>sort_buffer_size</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

Each session that needs to do a sort allocates a buffer of this size. `sort_buffer_size` [554] is not specific to any storage engine and applies in a general manner for optimization. See [Section 8.13.9, “ORDER BY Optimization”](#), for example.

If you see many `Sort_merge_passes` [608] per second in `SHOW GLOBAL STATUS` output, you can consider increasing the `sort_buffer_size` [554] value to speed up `ORDER BY` or `GROUP BY` operations that cannot be improved with query optimization or improved indexing. The entire buffer is allocated even if it is not all needed, so setting it larger than required globally will slow down most queries that sort. It is best to increase it as a session setting, and only for the sessions that need a larger size. On Linux, there are thresholds of 256KB and 2MB where larger values may significantly slow down memory allocation, so you should consider staying below one of those values. Experiment to find the best value for your workload. See [Section C.5.4.4, “Where MySQL Stores Temporary Files”](#).

The maximum permissible setting for `sort_buffer_size` [554] is 4GB. Values larger than 4GB are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB with a warning).

- `sql_auto_is_null` [554]

Variable Name	<code>sql_auto_is_null</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (<= 5.5.2)	
	Type	<code>boolean</code>
	Default	<code>1</code>
	Permitted Values (>= 5.5.3)	
	Type	<code>boolean</code>
	Default	<code>0</code>

If this variable is set to 1, then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` [1194] function. For details, including the return value after a multiple-row insert,

see [Section 12.14, “Information Functions”](#). If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL` [1091] comparison is used by some ODBC programs, such as Access. See [Section 22.1.7.1.1, “Obtaining Auto-Increment Values”](#). This behavior can be disabled by setting `sql_auto_is_null` [554] to 0.

The default value of `sql_auto_is_null` [554] is 0 as of MySQL 5.5.3, and 1 for earlier versions.

- `sql_big_selects` [555]

Variable Name	<code>sql_big_selects</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>1</code>

If set to 0, MySQL aborts `SELECT` statements that are likely to take a very long time to execute (that is, statements for which the optimizer estimates that the number of examined rows exceeds the value of `max_join_size` [522]). This is useful when an inadvisable `WHERE` statement has been issued. The default value for a new connection is 1, which permits all `SELECT` statements.

If you set the `max_join_size` [522] system variable to a value other than `DEFAULT`, `sql_big_selects` [555] is set to 0.

- `sql_buffer_result` [555]

Variable Name	<code>sql_buffer_result</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>0</code>

If set to 1, `sql_buffer_result` [555] forces results from `SELECT` statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. The default value is 0.

- `sql_log_bin` [555]

Variable Name	<code>sql_log_bin</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>

This variable controls whether logging to the binary log is done. The default value is 1 (do logging). To change logging for the current session, change the session value of this variable. The session user must have the `SUPER` [697] privilege to set this variable.

Beginning with MySQL 5.5.5, it is no longer possible to set `@@session.sql_log_bin` within a transaction or subquery. (Bug #53437)

- [sql_log_off \[556\]](#)

Variable Name	<code>sql_log_off</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>0</code>

This variable controls whether logging to the general query log is done. The default value is 0 (do logging). To change logging for the current session, change the session value of this variable. The session user must have the `SUPER` [697] privilege to set this option. The default value is 0.

- [sql_log_update \[556\]](#)

Version Removed	5.5.3	
Variable Name	<code>sql_log_update</code>	
Variable Scope	Session	
Dynamic Variable	Yes	
Deprecated	5.0, by <code>sql_log_bin</code>	
	Permitted Values	
	Type	<code>boolean</code>

This variable is deprecated, and is mapped to `sql_log_bin` [555]. It was removed in MySQL 5.5.3.

- [sql_mode \[556\]](#)

Command-Line Format	<code>--sql-mode=name</code>
Option-File Format	<code>sql-mode</code>
Option Sets Variable	Yes, <code>sql_mode</code> [556]
Variable Name	<code>sql_mode</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

The current server SQL mode, which can be set dynamically. See [Section 5.1.7, “Server SQL Modes”](#).

- [sql_notes \[556\]](#)

If set to 1 (the default), warnings of `Note` level increment `warning_count` and the server records them. If set to 0, `Note` warnings do not increment `warning_count` [573] and the server does not record them. `mysqldump` includes output to set this variable to 0 so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation.

- [sql_quote_show_create \[556\]](#)

If set to 1 (the default), the server quotes identifiers for `SHOW CREATE TABLE` and `SHOW CREATE DATABASE` statements. If set to 0, quoting is disabled. This option is enabled by default so that replication works for identifiers that require quoting. See [Section 13.7.5.12, “SHOW CREATE TABLE Syntax”](#), and [Section 13.7.5.8, “SHOW CREATE DATABASE Syntax”](#).

- `sql_safe_updates` [\[557\]](#)

If set to 1, MySQL aborts `UPDATE` or `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause. This makes it possible to catch `UPDATE` or `DELETE` statements where keys are not used properly and that would probably change or delete a large number of rows. The default value is 0.

- `sql_select_limit` [\[557\]](#)

Variable Name	<code>sql_select_limit</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>

The maximum number of rows to return from `SELECT` statements. The default value for a new connection is the maximum number of rows that the server permits per table. Typical default values are $(2^{32})-1$ or $(2^{64})-1$. If you have changed the limit, the default value can be restored by assigning a value of `DEFAULT`.

If a `SELECT` has a `LIMIT` clause, the `LIMIT` takes precedence over the value of `sql_select_limit` [\[557\]](#).

`sql_select_limit` [\[557\]](#) does not apply to `SELECT` statements executed within stored routines. It also does not apply to `SELECT` statements that do not produce a result set to be returned to the client. These include `SELECT` statements in subqueries, `CREATE TABLE ... SELECT`, and `INSERT INTO ... SELECT`.

- `sql_warnings` [\[557\]](#)

This variable controls whether single-row `INSERT` statements produce an information string if warnings occur. The default is 0. Set the value to 1 to produce an information string.

- `ssl_ca` [\[557\]](#)

Command-Line Format	<code>--ssl-ca=name</code>	
Option-File Format	<code>ssl-ca</code>	
Option Sets Variable	Yes, <code>ssl_ca</code> [557]	
Variable Name	<code>ssl-ca</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The path to a file with a list of trusted SSL CAs.

- `ssl_capath` [\[557\]](#)

Command-Line Format	<code>--ssl-capath=name</code>	
Option-File Format	<code>ssl-capath</code>	
Option Sets Variable	Yes, <code>ssl_capath</code> [557]	
Variable Name	<code>ssl-capath</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>directory name</code>

The path to a directory that contains trusted SSL CA certificates in PEM format.

- `ssl_cert` [558]

Command-Line Format	<code>--ssl-cert=name</code>	
Option-File Format	<code>ssl-cert</code>	
Option Sets Variable	Yes, <code>ssl_cert</code> [558]	
Variable Name	<code>ssl-cert</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The name of the SSL certificate file to use for establishing a secure connection.

- `ssl_cipher` [558]

Command-Line Format	<code>--ssl-cipher=name</code>	
Option-File Format	<code>ssl-cipher</code>	
Option Sets Variable	Yes, <code>ssl_cipher</code> [558]	
Variable Name	<code>ssl-cipher</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

A list of permissible ciphers to use for SSL encryption.

- `ssl_key` [558]

Command-Line Format	<code>--ssl-key=name</code>	
Option-File Format	<code>ssl-key</code>	
Option Sets Variable	Yes, <code>ssl_key</code> [558]	
Variable Name	<code>ssl-key</code>	
Variable Scope	Global	

Dynamic Variable	No
	Permitted Values
Type	string

The name of the SSL key file to use for establishing a secure connection.

- [storage_engine \[559\]](#)

Variable Name	storage_engine	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (>= 5.5.3, <= 5.5.4)	
Type	enumeration	
Default	MyISAM	
	Permitted Values (>= 5.5.5)	
Type	enumeration	
Default	InnoDB	

The default storage engine (table type). To set the storage engine at server startup, use the `--default-storage-engine [441]` option. See [Section 5.1.3, “Server Command Options”](#).

This variable is deprecated as of MySQL 5.5.3. Use [default_storage_engine \[494\]](#) instead.

- [stored_program_cache \[559\]](#)

Version Introduced	5.5.21	
Command-Line Format	<code>--stored-program-cache=#</code>	
Option-File Format	stored_program_cache	
Variable Name	stored_program_cache	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
Type	numeric	
Default	256	
Range	256 .. 524288	

Sets a soft upper limit for the number of cached stored routines per connection. The value of this variable is specified in terms of the number of stored routines held in each of the two caches maintained by the MySQL Server for, respectively, stored procedures and stored functions.

Whenever a stored routine is executed this cache size is checked before the first or top-level statement in the routine is parsed; if the number of routines of the same type (stored procedures or stored functions according to which is being executed) exceeds the limit specified by this variable, the corresponding cache is flushed and memory previously allocated for cached objects is freed. This allows the cache to be flushed safely, even when there are dependencies between stored routines.

- [sync_frm \[559\]](#)

Command-Line Format	<code>--sync_frm</code>	
Option-File Format	<code>sync_frm</code>	
Option Sets Variable	Yes, <code>sync_frm</code> [559]	
Variable Name	<code>sync_frm</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>TRUE</code>

If this variable is set to 1, when any nontemporary table is created its `.frm` file is synchronized to disk (using `fdatasync()`). This is slower but safer in case of a crash. The default is 1.

- `system_time_zone` [560]

Variable Name	<code>system_time_zone</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set `system_time_zone` [560]. Typically the time zone is specified by the `TZ` environment variable. It also can be specified using the `--timezone` [258] option of the `mysqld_safe` script.

The `system_time_zone` [560] variable differs from `time_zone` [567]. Although they might have the same value, the latter variable is used to initialize the time zone for each client that connects. See Section 10.6, “MySQL Server Time Zone Support”.

- `table_definition_cache` [560]

Command-Line Format	<code>--table_definition_cache=#</code>	
Option-File Format	<code>table_definition_cache</code>	
Option Sets Variable	Yes, <code>table_definition_cache</code> [560]	
Variable Name	<code>table_definition_cache</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>256</code>
	Range	<code>256 .. 524288</code>

The number of table definitions (from `.frm` files) that can be stored in the definition cache. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache. The minimum and default values are both 400.

- `table_lock_wait_timeout` [561]

Version Removed	5.5.3	
Command-Line Format	<code>--table_lock_wait_timeout=#</code>	
Option-File Format	<code>table_lock_wait_timeout</code>	
Option Sets Variable	Yes, <code>table_lock_wait_timeout</code> [561]	
Variable Name	<code>table_lock_wait_timeout</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	50
	Range	1 .. 1073741824

This variable is unused. It was removed in 5.5.3.

- `table_open_cache` [561]

Command-Line Format	<code>--table-open-cache=#</code>	
Option-File Format	<code>table_open_cache</code>	
Variable Name	<code>table_open_cache</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	400
	Range	64 .. 524288

The number of open tables for all threads. Increasing this value increases the number of file descriptors that `mysqld` requires. You can check whether you need to increase the table cache by checking the `Opened_tables` [604] status variable. See Section 5.1.6, “Server Status Variables”. If the value of `Opened_tables` [604] is large and you do not use `FLUSH TABLES` often (which just forces all tables to be closed and reopened), then you should increase the value of the `table_open_cache` [561] variable. For more information about the table cache, see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

- `table_type` [561]

This variable was removed in MySQL 5.5.3. Use `storage_engine` [559] instead.

- `thread_cache_size` [561]

Command-Line Format	<code>--thread_cache_size=#</code>
Option-File Format	<code>thread_cache_size</code>
Option Sets Variable	Yes, <code>thread_cache_size</code> [561]
Variable Name	<code>thread_cache_size</code>
Variable Scope	Global
Dynamic Variable	Yes

How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than `thread_cache_size` [561] threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. Normally, this does not provide a notable performance improvement if you have a good thread implementation. However, if your server sees hundreds of connections per second you should normally set `thread_cache_size` [561] high enough so that most new connections use cached threads. By examining the difference between the `Connections` [598] and `Threads_created` [610] status variables, you can see how efficient the thread cache is. For details, see [Section 5.1.6, "Server Status Variables"](#).

- `thread_concurrency` [562]

Command-Line Format	<code>--thread_concurrency=#</code>
Option-File Format	<code>thread_concurrency</code>
Option Sets Variable	Yes, <code>thread_concurrency</code> [562]
Variable Name	<code>thread_concurrency</code>
Variable Scope	Global
Dynamic Variable	No
Deprecated	5.6.1
	Permitted Values
	Type <code>numeric</code>
	Default <code>10</code>
	Range <code>1 .. 512</code>

This variable is specific to Solaris systems, for which `mysqld` invokes the `thr_setconcurrency()` with the variable value. This function enables applications to give the threads system a hint about the desired number of threads that should be run at the same time.

- `thread_handling` [562]

Command-Line Format	<code>--thread_handling=name</code>
Option-File Format	<code>thread_handling=name</code>
Option Sets Variable	Yes, <code>thread_handling</code> [562]
Variable Name	<code>thread_handling</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values (<= 5.5.13)

	Type	enumeration
	Valid Values	no-threads one-thread-per-connection
	Permitted Values (>= 5.5.16)	
	Type	enumeration
	Valid Values	no-threads one-thread-per-connection dynamically-loaded

The thread-handling model used by the server for connection threads. The permissible user-settable values are `no-threads` (the server uses a single thread) and `one-thread-per-connection` (the server uses one thread to handle each client connection; this is the default). `no-threads` is useful for debugging under Linux; see [MySQL Internals: Porting to Other Systems](#).

If the thread pool plugin is enabled, the server sets the `thread_handling` [562] value to `dynamically-loaded`. See [Section 8.11.6.1, “Thread Pool Components and Installation”](#).

- `thread_pool_algorithm` [563]

Version Introduced	5.5.16	
Command-Line Format	<code>--thread_pool_algorithm=#</code>	
Option-File Format	<code>thread_pool_algorithm=#</code>	
Option Sets Variable	Yes, <code>thread_pool_algorithm</code> [563]	
Variable Name	<code>thread_pool_algorithm</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 1

This variable controls which algorithm the thread pool plugin uses:

- A value of 0 (the default) uses a conservative low-concurrency algorithm which is most well tested and is known to produce very good results.
- A value of 1 increases the concurrency and uses a more aggressive algorithm which at times has been known to perform 5–10% better on optimal thread counts, but has degrading performance as the number of connections increases. Its use should be considered as experimental and not supported.

This variable was added in MySQL 5.5.16. It is available only if the thread pool plugin is enabled. See [Section 8.11.6, “The Thread Pool Plugin”](#)

- `thread_pool_high_priority_connection` [563]

Version Introduced	5.5.16
---------------------------	--------

Command-Line Format	<code>--thread_pool_high_priority_connection=#</code>
Option-File Format	<code>thread_pool_high_priority_connection=#</code>
Option Sets Variable	Yes, <code>thread_pool_high_priority_connection</code> [563]
Variable Name	<code>thread_pool_high_priority_connection</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
Type	<code>numeric</code>
Default	<code>0</code>
Range	<code>0 .. 1</code>

This variable affects queuing of new statements prior to execution. If the value is 0 (false, the default), statement queuing uses both the low-priority and high-priority queues. If the value is 1 (true), queued statements always go to the high-priority queue.

This variable was added in MySQL 5.5.16. It is available only if the thread pool plugin is enabled. See [Section 8.11.6, “The Thread Pool Plugin”](#)

- `thread_pool_max_unused_threads` [564]

Version Introduced	5.5.16
Command-Line Format	<code>--thread_pool_max_unused_threads=#</code>
Option-File Format	<code>thread_pool_max_unused_threads=#</code>
Option Sets Variable	Yes, <code>thread_pool_max_unused_threads</code> [564]
Variable Name	<code>thread_pool_max_unused_threads</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	<code>numeric</code>
Default	<code>0</code>
Range	<code>0 .. 4096</code>

The maximum permitted number of unused threads in the thread pool. This variable makes it possible to limit the amount of memory used by sleeping threads.

A value of 0 (the default) means no limit on the number of sleeping threads. A value of N where N is greater than 0 means 1 consumer thread and $N-1$ reserve threads. In this case, if a thread is ready to sleep but the number of sleeping threads is already at the maximum, the thread exits rather than going to sleep.

A sleeping thread is either sleeping as a consumer thread or a reserve thread. The thread pool permits one thread to be the consumer thread when sleeping. If a thread goes to sleep and there is no existing consumer thread, it will sleep as a consumer thread. When a thread must be woken up, a consumer thread is selected if there is one. A reserve thread is selected only when there is no consumer thread to wake up.

This variable was added in MySQL 5.5.16. It is available only if the thread pool plugin is enabled. See [Section 8.11.6, “The Thread Pool Plugin”](#)

- `thread_pool_prio_kickup_timer` [\[565\]](#)

Version Introduced	5.5.16	
Command-Line Format	<code>--thread_pool_prio_kickup_timer=#</code>	
Option-File Format	<code>thread_pool_prio_kickup_timer=#</code>	
Option Sets Variable	Yes, <code>thread_pool_prio_kickup_timer</code> [565]	
Variable Name	<code>thread_pool_prio_kickup_timer</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>1000</code>
	Range	<code>0 .. 4294967294</code>

This variable affects statements waiting for execution in the low-priority queue. The value is the number of milliseconds before a waiting statement is moved to the high-priority queue. The default is 1000 (1 second). The range of values is 0 to $2^{32} - 2$.

This variable was added in MySQL 5.5.16. It is available only if the thread pool plugin is enabled. See [Section 8.11.6, “The Thread Pool Plugin”](#)

- `thread_pool_size` [\[565\]](#)

Version Introduced	5.5.16	
Command-Line Format	<code>--thread_pool_size=#</code>	
Option-File Format	<code>thread_pool_size=#</code>	
Option Sets Variable	Yes, <code>thread_pool_size</code> [565]	
Variable Name	<code>thread_pool_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>16</code>
	Range	<code>1 .. 64</code>

The number of thread groups in the thread pool. This is the most important parameter controlling thread pool performance. It affects how many statements can execute simultaneously. The default value is 16, with a range from 1 to 64 of permissible values. If a value outside this range is specified, the thread pool plugin does not load and the server writes a message to the error log.

This variable was added in MySQL 5.5.16. It is available only if the thread pool plugin is enabled. See [Section 8.11.6, “The Thread Pool Plugin”](#)

- `thread_pool_stall_limit` [566]

Version Introduced	5.5.16	
Command-Line Format	<code>--thread_pool_stall_limit=#</code>	
Option-File Format	<code>thread_pool_stall_limit=#</code>	
Option Sets Variable	Yes, <code>thread_pool_stall_limit</code> [566]	
Variable Name	<code>thread_pool_stall_limit</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>6</code>
	Range	<code>4 .. 600</code>

This variable affects executing statements. The value is the amount of time a statement has to finish after starting to execute before it becomes defined as stalled, at which point the thread pool permits the thread group to begin executing another statement. The value is measured in 10 millisecond units, so a value of 6 (the default) means 60ms. The range of values is 4 to 600 (40ms to 6s). Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.

This variable was added in MySQL 5.5.16. It is available only if the thread pool plugin is enabled. See [Section 8.11.6, “The Thread Pool Plugin”](#)

- `thread_stack` [566]

Command-Line Format	<code>--thread_stack=#</code>	
Option-File Format	<code>thread_stack</code>	
Option Sets Variable	Yes, <code>thread_stack</code> [566]	
Variable Name	<code>thread_stack</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Platform Bit Size	<code>32</code>
	Type	<code>numeric</code>
	Default	<code>196608</code>
	Range	<code>131072 .. 4294967295</code>
	Block Size	<code>1024</code>
	Permitted Values	
	Platform Bit Size	<code>64</code>
	Type	<code>numeric</code>

Default	262144
Range	131072 .. 18446744073709547520
Block Size	1024

The stack size for each thread. Many of the limits detected by the `crash-me` test are dependent on this value. See [Section 8.12.2, “The MySQL Benchmark Suite”](#). The default of 192KB (256KB for 64-bit systems) is large enough for normal operation. If the thread stack size is too small, it limits the complexity of the SQL statements that the server can handle, the recursion depth of stored procedures, and other memory-consuming actions.

- `time_format` [\[567\]](#)

This variable is unused.

- `time_zone` [\[567\]](#)

Command-Line Format	<code>--default_time_zone=string</code>	
Option-File Format	<code>default_time_zone</code>	
Variable Name	<code>time_zone</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>

The current time zone. This variable is used to initialize the time zone for each client that connects. By default, the initial value of this is 'SYSTEM' (which means, “use the value of `system_time_zone` [\[560\]](#)”). The value can be specified explicitly at server startup with the `--default-time-zone` [\[441\]](#) option. See [Section 10.6, “MySQL Server Time Zone Support”](#).

- `timed_mutexes` [\[567\]](#)

Command-Line Format	<code>--timed_mutexes</code>	
Option-File Format	<code>timed_mutexes</code>	
Option Sets Variable	Yes, <code>timed_mutexes</code> [567]	
Variable Name	<code>timed_mutexes</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

This variable controls whether InnoDB mutexes are timed. If this variable is set to 0 or `OFF` (the default), mutex timing is disabled. If the variable is set to 1 or `ON`, mutex timing is enabled. With timing enabled, the `os_wait_times` value in the output from `SHOW ENGINE INNODB MUTEX` indicates the amount of time (in ms) spent in operating system waits. Otherwise, the value is 0.

- `timestamp = {timestamp_value | DEFAULT}` [\[567\]](#)

Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. `timestamp_value` should be a Unix epoch timestamp, not a MySQL timestamp.

`SET timestamp` affects the value returned by `NOW()` [1143] but not by `SYSDATE()` [1146]. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()` [1146]. The server can be started with the `--sysdate-is-now` [465] option to cause `SYSDATE()` [1146] to be an alias for `NOW()` [1143], in which case `SET timestamp` affects both functions.

- `tmp_table_size` [568]

Command-Line Format	<code>--tmp_table_size=#</code>	
Option-File Format	<code>tmp_table_size</code>	
Option Sets Variable	Yes, <code>tmp_table_size</code> [568]	
Variable Name	<code>tmp_table_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>system dependent</code>
	Range	<code>1024 .. 4294967295</code>

The maximum size of internal in-memory temporary tables. (The actual limit is determined as the minimum of `tmp_table_size` [568] and `max_heap_table_size` [521].) If an in-memory temporary table exceeds the limit, MySQL automatically converts it to an on-disk `MyISAM` table. Increase the value of `tmp_table_size` [568] (and `max_heap_table_size` [521] if necessary) if you do many advanced `GROUP BY` queries and you have lots of memory. This variable does not apply to user-created `MEMORY` tables.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` [598] and `Created_tmp_tables` [598] variables.

See also [Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#).

- `tmpdir` [568]

Command-Line Format	<code>--tmpdir=path</code>	
	<code>-t</code>	
Option-File Format	<code>tmpdir</code>	
Option Sets Variable	Yes, <code>tmpdir</code> [568]	
Variable Name	<code>tmpdir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The directory used for temporary files and temporary tables. This variable can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (":") on Unix and semicolon characters (";") on Windows.

The multiple-directory feature can be used to spread the load between several physical disks. If the MySQL server is acting as a replication slave, you should not set `tmpdir` [568] to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails. You can set the slave's temporary directory using the `slave_load_tmpdir` [1887] variable. In that case, the slave will not use the general `tmpdir` [568] value and you can set `tmpdir` [568] to a nonpermanent location.

- `transaction_alloc_block_size` [569]

Command-Line Format	<code>--transaction_alloc_block_size=#</code>	
Option-File Format	<code>transaction_alloc_block_size</code>	
Option Sets Variable	Yes, <code>transaction_alloc_block_size</code> [569]	
Variable Name	<code>transaction_alloc_block_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	8192
	Range	1024 .. 4294967295
	Block Size	1024
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	8192
	Range	1024 .. 18446744073709547520
	Block Size	1024

The amount in bytes by which to increase a per-transaction memory pool which needs memory. See the description of `transaction_prealloc_size` [569].

- `transaction_prealloc_size` [569]

Command-Line Format	<code>--transaction_prealloc_size=#</code>
Option-File Format	<code>transaction_prealloc_size</code>
Option Sets Variable	Yes, <code>transaction_prealloc_size</code> [569]

Variable Name	<code>transaction_prealloc_size</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	<code>numeric</code>
	Default	4096
	Range	1024 .. 4294967295
	Block Size	1024
	Permitted Values	
	Platform Bit Size	64
	Type	<code>numeric</code>
	Default	4096
	Range	1024 .. 18446744073709547520
	Block Size	1024

There is a per-transaction memory pool from which various transaction-related allocations take memory. The initial size of the pool in bytes is `transaction_prealloc_size` [569]. For every allocation that cannot be satisfied from the pool because it has insufficient memory available, the pool is increased by `transaction_alloc_block_size` [569] bytes. When the transaction ends, the pool is truncated to `transaction_prealloc_size` [569] bytes.

By making `transaction_prealloc_size` [569] sufficiently large to contain all statements within a single transaction, you can avoid many `malloc()` calls.

- `tx_isolation` [570]

Variable Name	<code>tx_isolation</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>enumeration</code>
	Default	<code>REPEATABLE-READ</code>
	Valid Values	<code>READ-UNCOMMITTED</code> <code>READ-COMMITTED</code> <code>REPEATABLE-READ</code> <code>SERIALIZABLE</code>

The default transaction isolation level. Defaults to `REPEATABLE-READ` [1401].

This variable can be set directly, or indirectly using the `SET TRANSACTION` statement. See [Section 13.3.6, “SET TRANSACTION Syntax”](#). If you set `tx_isolation` [570] directly to an isolation level name that contains a space, the name should be enclosed within quotation marks, with the space replaced by a dash. For example:

```
SET tx_isolation = 'READ-COMMITTED';
```

Any unique prefix of a valid value may be used to set the value of this variable.

The default transaction isolation level can also be set at startup using the `--transaction-isolation` [465] server option.

- `unique_checks` [571]

Variable Name	<code>unique_checks</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>1</code>

If set to 1 (the default), uniqueness checks for secondary indexes in `InnoDB` tables are performed. If set to 0, storage engines are permitted to assume that duplicate keys are not present in input data. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to `InnoDB`.

Note that setting this variable to 0 does not *require* storage engines to ignore duplicate keys. An engine is still permitted to check for them and issue duplicate-key errors if it detects them.

- `updatable_views_with_limit` [571]

Command-Line Format	<code>--updatable_views_with_limit=#</code>	
Option-File Format	<code>updatable_views_with_limit</code>	
Option Sets Variable	Yes, <code>updatable_views_with_limit</code> [571]	
Variable Name	<code>updatable_views_with_limit</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>1</code>

This variable controls whether updates to a view can be made when the view does not contain all columns of the primary key defined in the underlying table, if the update statement contains a `LIMIT` clause. (Such updates often are generated by GUI tools.) An update is an `UPDATE` or `DELETE` statement. Primary key here means a `PRIMARY KEY`, or a `UNIQUE` index in which no column can contain `NULL`.

The variable can have two values:

- `1` or `YES`: Issue a warning only (not an error message). This is the default value.

- 0 or NO: Prohibit the update.
- `version` [572]

The version number for the server. The value might also include a suffix indicating server build or configuration information. `-log` indicates that one or more of the general log, slow query log, or binary log are enabled. `-debug` indicates that the server was built with debugging support enabled.

- `version_comment` [572]

Variable Name	<code>version_comment</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

The CMake configuration program has a `COMPILATION_COMMENT` [154] option that permits a comment to be specified when building MySQL. This variable contains the value of that comment. See Section 2.10.4, “MySQL Source-Configuration Options”.

- `version_compile_machine` [572]

Variable Name	<code>version_compile_machine</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

The type of machine or architecture on which MySQL was built.

- `version_compile_os` [572]

Variable Name	<code>version_compile_os</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

The type of operating system on which MySQL was built.

- `wait_timeout` [572]

Command-Line Format	<code>--wait_timeout=#</code>
Option-File Format	<code>wait_timeout</code>
Option Sets Variable	Yes, <code>wait_timeout</code> [572]
Variable Name	<code>wait_timeout</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

	Permitted Values	
Type	numeric	
Default	28800	
Range	1 .. 31536000	
	Permitted Values	
Type (windows)	numeric	
Default	28800	
Range	1 .. 2147483	

The number of seconds the server waits for activity on a noninteractive connection before closing it.

On thread startup, the session `wait_timeout` [572] value is initialized from the global `wait_timeout` [572] value or from the global `interactive_timeout` [506] value, depending on the type of client (as defined by the `CLIENT_INTERACTIVE` connect option to `mysql_real_connect()`). See also `interactive_timeout` [506].

- `warning_count` [573]

The number of errors, warnings, and notes that resulted from the last statement that generated messages. This variable is read only. See [Section 13.7.5.41, “SHOW WARNINGS Syntax”](#).

5.1.5. Using System Variables

The MySQL server maintains many system variables that indicate how it is configured. [Section 5.1.4, “Server System Variables”](#), describes the meaning of these variables. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

The server maintains two kinds of system variables. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections. A given system variable can have both a global and a session value. Global and session system variables are related as follows:

- When the server starts, it initializes all global variables to their default values. These defaults can be changed by options specified on the command line or in an option file. (See [Section 4.2.3, “Specifying Program Options”](#).)
- The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For example, the client's SQL mode is controlled by the session `sql_mode` [556] value, which is initialized when the client connects to the value of the global `sql_mode` [556] value.

System variable values can be set globally at server startup by using options on the command line or in an option file. When you use a startup option to set a variable that takes a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024 , 1024^2 or 1024^3 ; that is, units of kilobytes, megabytes, or gigabytes, respectively. Thus, the following command starts the server with a query cache size of 16 megabytes and a maximum packet size of one gigabyte:

```
mysqld --query_cache_size=16M --max_allowed_packet=1G
```

Within an option file, those variables are set like this:

```
[mysqld]
query_cache_size=16M
max_allowed_packet=1G
```

The lettercase of suffix letters does not matter; `16M` and `16m` are equivalent, as are `1G` and `1g`.

If you want to restrict the maximum value to which a system variable can be set at runtime with the `SET` statement, you can specify this maximum by using an option of the form `--maximum-var_name=value` at server startup. For example, to prevent the value of `query_cache_size` [541] from being increased to more than 32MB at runtime, use the option `--maximum-query_cache_size=32M`.

Many system variables are dynamic and can be changed while the server runs by using the `SET` statement. For a list, see [Section 5.1.5.2, “Dynamic System Variables”](#). To change a system variable with `SET`, refer to it as `var_name`, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by `GLOBAL` or `@@global..`. The `SUPER` [697] privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by `SESSION`, `@@session..`, or `@@`. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- `LOCAL` and `@@local.` are synonyms for `SESSION` and `@@session..`
- If no modifier is present, `SET` changes the session variable.

A `SET` statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent `GLOBAL` or `SESSION` modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

The `@@var_name` syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the `SET GLOBAL` statement).

To prevent incorrect usage, MySQL produces an error if you use `SET GLOBAL` with a variable that can only be used with `SET SESSION` or if you do not specify `GLOBAL` (or `@@global.`) when setting a global variable.

To set a `SESSION` variable to the `GLOBAL` value or a `GLOBAL` value to the compiled-in MySQL default value, use the `DEFAULT` keyword. For example, the following two statements are identical in setting the session value of `max_join_size` [522] to the global value:

```
SET max_join_size=DEFAULT;
```

```
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to `DEFAULT`. In such cases, use of `DEFAULT` results in an error.

You can refer to the values of specific global or session system variables in expressions by using one of the `@@`-modifiers. For example, you can retrieve values in a `SELECT` statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as `@@var_name` (that is, when you do not specify `@@global.` or `@@session.`), MySQL returns the session value if it exists and the global value otherwise. (This differs from `SET @@var_name = value`, which always refers to the session value.)



Note

Some variables displayed by `SHOW VARIABLES` may not be available using `SELECT @@var_name` syntax; an `Unknown system variable` occurs. As a workaround in such cases, you can use `SHOW VARIABLES LIKE 'var_name'`.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```



Note

Some system variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1` [494] works but `--delay_key_write=ON` [494] does not.

To display system variable names and values, use the `SHOW VARIABLES` statement:

```
mysql> SHOW VARIABLES;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| automatic_sp_privileges | ON |
| back_log | 50 |
| basedir | /home/mysql/ |
| binlog_cache_size | 32768 |
| bulk_insert_buffer_size | 8388608 |
| character_set_client | latin1 |
| character_set_connection | latin1 |
| character_set_database | latin1 |
| character_set_results | latin1 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /home/mysql/share/mysql/charsets/ |
| collation_connection | latin1_swedish_ci |
| collation_database | latin1_swedish_ci |
+-----+-----+
```

collation_server	latin1_swedish_ci
...	
innodb_additional_mem_pool_size	1048576
innodb_autoextend_increment	8
innodb_buffer_pool_size	8388608
innodb_checksums	ON
innodb_commit_concurrency	0
innodb_concurrency_tickets	500
innodb_data_file_path	ibdata1:10M:autoextend
innodb_data_home_dir	
...	
version	5.1.6-alpha-log
version_comment	Source distribution
version_compile_machine	i686
version_compile_os	suse-linux
wait_timeout	28800

With a [LIKE \[1111\]](#) clause, the statement displays only those variables that match the pattern. To obtain a specific variable name, use a [LIKE \[1111\]](#) clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the “%” wildcard character in a [LIKE \[1111\]](#) clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because “_” is a wildcard that matches any single character, you should escape it as “_” to match it literally. In practice, this is rarely necessary.

For [SHOW VARIABLES](#), if you specify neither [GLOBAL](#) nor [SESSION](#), MySQL returns [SESSION](#) values.

The reason for requiring the [GLOBAL](#) keyword when setting [GLOBAL](#)-only variables but not when retrieving them is to prevent problems in the future. If we were to remove a [SESSION](#) variable that has the same name as a [GLOBAL](#) variable, a client with the [SUPER \[697\]](#) privilege might accidentally change the [GLOBAL](#) variable rather than just the [SESSION](#) variable for its own connection. If we add a [SESSION](#) variable with the same name as a [GLOBAL](#) variable, a client that intends to change the [GLOBAL](#) variable might find only its own [SESSION](#) variable changed.

5.1.5.1. Structured System Variables

A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.
- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

MySQL 5.5 supports one structured variable type, which specifies parameters governing the operation of key caches. A key cache structured variable has these components:

- [key_buffer_size \[508\]](#)
- [key_cache_block_size \[509\]](#)
- [key_cache_division_limit \[510\]](#)
- [key_cache_age_threshold \[509\]](#)

This section describes the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in [Section 8.9.2, “The MyISAM Key Cache”](#).

To refer to a component of a structured variable instance, you can use a compound name in `instance_name.component_name` format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of `default` is always predefined. If you refer to a component of a structured variable without any instance name, the `default` instance is used. Thus, `default.key_buffer_size` and `key_buffer_size` [508] both refer to the same system variable.

Structured variable instances and components follow these naming rules:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named `default`, so `default` is not unique across variable types.
- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.
- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, `hot-cache` is not legal, but ``hot-cache`` is.
- `global`, `session`, and `local` are not legal instance names. This avoids a conflict with notation such as `@@global.var_name` for referring to nonstructured system variables.

Currently, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules will assume greater significance if some other type of structured variable is created in the future.

With one exception, you can refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

In an option file, use this syntax:

```
[mysqld]
hot_cache.key_buffer_size=64K
```

If you start the server with this option, it creates a key cache named `hot_cache` with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysqld --key_buffer_size=256K \
  --extra_cache.key_buffer_size=128K \
  --extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written `--default.key_buffer_size=256K`.) In addition, the server creates a second key cache named

`extra_cache` that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysqld --key_buffer_size=6M \
        --hot_cache.key_buffer_size=2M \
        --cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named `hot_cache` to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a `LIKE [1111]` pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a simple variable name may occur.

5.1.5.2. Dynamic System Variables

Many server system variables are dynamic and can be set at runtime using `SET GLOBAL` or `SET SESSION`. You can also obtain their values using `SELECT`. See [Section 5.1.5, “Using System Variables”](#).

The following table shows the full list of all dynamic system variables. The last column indicates for each variable whether `GLOBAL` or `SESSION` (or both) apply. The table also lists session options that can be set with the `SET` statement. [Section 5.1.4, “Server System Variables”](#), discusses these options.

Variables that have a type of “string” take a string value. Variables that have a type of “numeric” take a numeric value. Variables that have a type of “boolean” can be set to 0, 1, `ON` or `OFF`. (If you set them on the command line or in an option file, use the numeric values.) Variables that are marked as “enumeration” normally should be set to one of the available values for the variable, but can also be set to the number that corresponds to the desired enumeration value. For enumerated system variables, the first enumeration value corresponds to 0. This differs from `ENUM` columns, for which the first enumeration value corresponds to 1.

Table 5.3. Dynamic Variable Summary

Variable Name	Variable Type	Variable Scope
<code>audit_log_flush</code> [765]	boolean	GLOBAL
<code>audit_log_policy</code> [766]	enumeration	GLOBAL
<code>audit_log_rotate_on_size</code> [766]	numeric	GLOBAL
<code>auto_increment_increment</code> [1865]	numeric	GLOBAL SESSION
<code>auto_increment_offset</code> [1868]	numeric	GLOBAL SESSION
<code>autocommit</code> [484]	boolean	GLOBAL SESSION
<code>automatic_sp_privileges</code> [484]	boolean	GLOBAL
<code>big_tables</code> [435]	boolean	GLOBAL SESSION
<code>binlog_cache_size</code> [1898]	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
binlog_direct_non_transactional_updates [1898]	boolean	GLOBAL SESSION
binlog_format [436]	enumeration	GLOBAL SESSION
binlog_stmt_cache_size [1901]	numeric	GLOBAL
bulk_insert_buffer_size [486]	numeric	GLOBAL SESSION
character_set_client [486]	string	GLOBAL SESSION
character_set_connection [487]	string	GLOBAL SESSION
character_set_database [487]	string	GLOBAL SESSION
character_set_filesystem [437]	string	GLOBAL SESSION
character_set_results [488]	string	GLOBAL SESSION
character_set_server [438]	string	GLOBAL SESSION
collation_connection [489]	string	GLOBAL SESSION
collation_database [489]	string	GLOBAL SESSION
collation_server [438]	string	GLOBAL SESSION
completion_type [489]	numeric	GLOBAL SESSION
concurrent_insert [490]	boolean	GLOBAL
connect_timeout [492]	numeric	GLOBAL
debug [439]	string	GLOBAL SESSION
debug_sync [493]	string	SESSION
default_storage_engine [441]	enumeration	GLOBAL SESSION
default_week_format [494]	numeric	GLOBAL SESSION
delay_key_write [441]	enumeration	GLOBAL
delayed_insert_limit [495]	numeric	GLOBAL
delayed_insert_timeout [496]	numeric	GLOBAL
delayed_queue_size [496]	numeric	GLOBAL
div_precision_increment [497]	numeric	GLOBAL SESSION
engine_condition_pushdown [443]	boolean	GLOBAL SESSION
event_scheduler [443]	enumeration	GLOBAL
expire_logs_days [498]	numeric	GLOBAL
flush [499]	boolean	GLOBAL
flush_time [500]	numeric	GLOBAL
foreign_key_checks [500]	boolean	GLOBAL SESSION
ft_boolean_syntax [500]	string	GLOBAL
general_log [445]	boolean	GLOBAL
general_log_file [503]	filename	GLOBAL
group_concat_max_len [503]	numeric	GLOBAL SESSION
identity [505]	numeric	SESSION
init_connect [505]	string	GLOBAL
init_slave [1885]	string	GLOBAL

Variable Name	Variable Type	Variable Scope
<code>innodb_adaptive_flushing</code> [1547]	boolean	GLOBAL
<code>innodb_adaptive_hash_index</code> [1547]	boolean	GLOBAL
<code>innodb_autoextend_increment</code> [1548]	numeric	GLOBAL
<code>innodb_change_buffering</code> [1550]	enumeration	GLOBAL
<code>innodb_commit_concurrency</code> [1551]	numeric	GLOBAL
<code>innodb_concurrency_tickets</code> [1552]	numeric	GLOBAL
<code>innodb_fast_shutdown</code> [1553]	numeric	GLOBAL
<code>innodb_file_format</code> [1554]	string	GLOBAL
<code>innodb_file_format_max</code> [1555]	string	GLOBAL
<code>innodb_file_per_table</code> [1555]	boolean	GLOBAL
<code>innodb_flush_log_at_trx_commit</code> [1556]	enumeration	GLOBAL
<code>innodb_io_capacity</code> [1559]	numeric	GLOBAL
<code>innodb_large_prefix</code> [1560]	boolean	GLOBAL
<code>innodb_lock_wait_timeout</code> [1560]	numeric	GLOBAL SESSION
<code>innodb_max_dirty_pages_pct</code> [1565]	numeric	GLOBAL
<code>innodb_max_purge_lag</code> [1565]	numeric	GLOBAL
<code>innodb_old_blocks_pct</code> [1566]	numeric	GLOBAL
<code>innodb_old_blocks_time</code> [1566]	numeric	GLOBAL
<code>innodb_read_ahead_threshold</code> [1568]	numeric	GLOBAL
<code>innodb_replication_delay</code> [1569]	numeric	GLOBAL
<code>innodb_rollback_segments</code> [1570]	numeric	GLOBAL
<code>innodb_spin_wait_delay</code> [1570]	numeric	GLOBAL
<code>innodb_stats_method</code> [1570]	enumeration	GLOBAL
<code>innodb_stats_on_metadata</code> [1571]	boolean	GLOBAL
<code>innodb_stats_sample_pages</code> [1571]	numeric	GLOBAL
<code>innodb_strict_mode</code> [1572]	boolean	GLOBAL SESSION
<code>innodb_support_xa</code> [1572]	boolean	GLOBAL SESSION
<code>innodb_sync_spin_loops</code> [1573]	numeric	GLOBAL
<code>innodb_table_locks</code> [1573]	boolean	GLOBAL SESSION
<code>innodb_thread_concurrency</code> [1573]	numeric	GLOBAL
<code>innodb_thread_sleep_delay</code> [1574]	numeric	GLOBAL
<code>insert_id</code> [506]	numeric	SESSION
<code>interactive_timeout</code> [506]	numeric	GLOBAL SESSION
<code>join_buffer_size</code> [507]	numeric	GLOBAL SESSION
<code>keep_files_on_create</code> [507]	boolean	GLOBAL SESSION
<code>key_buffer_size</code> [508]	numeric	GLOBAL
<code>key_cache_age_threshold</code> [509]	numeric	GLOBAL
<code>key_cache_block_size</code> [509]	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
key_cache_division_limit [510]	numeric	GLOBAL
last_insert_id [511]	numeric	SESSION
lc_messages [447]	string	GLOBAL SESSION
lc_time_names [512]	string	GLOBAL SESSION
local_infile [513]	boolean	GLOBAL
lock_wait_timeout [513]	numeric	GLOBAL SESSION
log [447]	string	GLOBAL
log_bin_trust_function_creators [1892]	boolean	GLOBAL
log_bin_trust_routine_creators	boolean	GLOBAL
log_output [449]	set	GLOBAL
log_queries_not_using_indexes [449]	boolean	GLOBAL
log_slow_queries [450]	boolean	GLOBAL
log_warnings [451]	numeric	GLOBAL SESSION
long_query_time [517]	numeric	GLOBAL SESSION
low_priority_updates [452]	boolean	GLOBAL SESSION
max_allowed_packet [519]	numeric	GLOBAL
max_binlog_cache_size [1902]	numeric	GLOBAL
max_binlog_size [1903]	numeric	GLOBAL
max_binlog_stmt_cache_size [1903]	numeric	GLOBAL
max_connect_errors [519]	numeric	GLOBAL
max_connections [519]	numeric	GLOBAL
max_delayed_threads [520]	numeric	GLOBAL SESSION
max_error_count [520]	numeric	GLOBAL SESSION
max_heap_table_size [521]	numeric	GLOBAL SESSION
max_insert_delayed_threads [521]	numeric	GLOBAL SESSION
max_join_size [522]	numeric	GLOBAL SESSION
max_length_for_sort_data [522]	numeric	GLOBAL SESSION
max_prepared_stmt_count [523]	numeric	GLOBAL
max_relay_log_size [523]	numeric	GLOBAL
max_seeks_for_key [524]	numeric	GLOBAL SESSION
max_sort_length [524]	numeric	GLOBAL SESSION
max_sp_recursion_depth [525]	numeric	GLOBAL SESSION
max_user_connections [525]	numeric	GLOBAL SESSION
max_write_lock_count [526]	numeric	GLOBAL
min_examined_row_limit [452]	numeric	GLOBAL SESSION
myisam_data_pointer_size [527]	numeric	GLOBAL
myisam_max_sort_file_size [528]	numeric	GLOBAL
myisam_repair_threads [529]	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
mysam_sort_buffer_size [530]	numeric	GLOBAL SESSION
mysam_stats_method [530]	enumeration	GLOBAL SESSION
mysam_use_mmap [531]	boolean	GLOBAL
ndb_autoincrement_prefetch_sz [2148]	numeric	GLOBAL SESSION
ndb_blob_read_batch_bytes [2140]	numeric	GLOBAL SESSION
ndb_blob_write_batch_bytes [2140]	numeric	GLOBAL SESSION
ndb_cache_check_time [2149]	numeric	GLOBAL
ndb_deferred_constraints [2141]	boolean	GLOBAL SESSION
ndb_deferred_constraints [2149]	boolean	GLOBAL SESSION
ndb_distribution={KEYHASH LINHASH} [2150]	enumeration	GLOBAL SESSION
ndb_distribution [2141]	enumeration	GLOBAL SESSION
ndb_extra_logging [2150]	numeric	GLOBAL
ndb_force_send [2150]	boolean	GLOBAL SESSION
ndb_log_bin [2153]	boolean	GLOBAL SESSION
ndb_log_binlog_index [2153]	boolean	GLOBAL
ndb_log_empty_epochs [2160]	boolean	GLOBAL
ndb_log_empty_epochs [2144]	boolean	GLOBAL
ndb_log_update_as_write [2373]	boolean	GLOBAL
ndb_log_updated_only [2372]	boolean	GLOBAL
ndb_optimization_delay [2147]	numeric	GLOBAL
ndb_table_no_logging [2156]	boolean	SESSION
ndb_table_temporary [2157]	boolean	SESSION
ndb_use_exact_count [2157]	boolean	GLOBAL SESSION
ndb_use_transactions [2157]	boolean	GLOBAL SESSION
ndbinfo_max_bytes [2159]	numeric	GLOBAL SESSION
ndbinfo_max_rows [2159]	numeric	GLOBAL SESSION
ndbinfo_show_hidden [2159]	boolean	GLOBAL SESSION
ndbinfo_table_prefix [2160]	string	GLOBAL SESSION
net_buffer_length [531]	numeric	GLOBAL SESSION
net_read_timeout [532]	numeric	GLOBAL SESSION
net_retry_count [532]	numeric	GLOBAL SESSION
net_write_timeout [533]	numeric	GLOBAL SESSION
new [533]	boolean	GLOBAL SESSION
old_alter_table [455]	boolean	GLOBAL SESSION
old_passwords [534]	boolean	GLOBAL SESSION
optimizer_prune_level [535]	boolean	GLOBAL SESSION
optimizer_search_depth [536]	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
<code>optimizer_switch</code> [536]	set	GLOBAL SESSION
<code>preload_buffer_size</code> [538]	numeric	GLOBAL SESSION
<code>profiling</code> [539]	boolean	GLOBAL SESSION
<code>profiling_history_size</code> [539]	numeric	GLOBAL SESSION
<code>pseudo_thread_id</code> [539]	numeric	SESSION
<code>query_alloc_block_size</code> [539]	numeric	GLOBAL SESSION
<code>query_cache_limit</code> [540]	numeric	GLOBAL
<code>query_cache_min_res_unit</code> [541]	numeric	GLOBAL
<code>query_cache_size</code> [541]	numeric	GLOBAL
<code>query_cache_type</code> [542]	enumeration	GLOBAL SESSION
<code>query_cache_wlock_invalidate</code> [542]	boolean	GLOBAL SESSION
<code>query_prealloc_size</code> [543]	numeric	GLOBAL SESSION
<code>rand_seed1</code> [543]	numeric	SESSION
<code>rand_seed2</code> [543]	numeric	SESSION
<code>range_alloc_block_size</code> [544]	numeric	GLOBAL SESSION
<code>read_buffer_size</code> [544]	numeric	GLOBAL SESSION
<code>read_only</code> [544]	boolean	GLOBAL
<code>read_rnd_buffer_size</code> [545]	numeric	GLOBAL SESSION
<code>relay_log_purge</code> [546]	boolean	GLOBAL
<code>rpl_recovery_rank</code> [1887]	numeric	GLOBAL
<code>rpl_semi_sync_master_enabled</code> [548]	boolean	GLOBAL
<code>rpl_semi_sync_master_timeout</code> [548]	numeric	GLOBAL
<code>rpl_semi_sync_master_trace_level</code> [549]	numeric	GLOBAL
<code>rpl_semi_sync_master_wait_no_slave</code> [549]	boolean	GLOBAL
<code>rpl_semi_sync_slave_enabled</code> [550]	boolean	GLOBAL
<code>rpl_semi_sync_slave_trace_level</code> [550]	numeric	GLOBAL
<code>safe_show_database</code> [459]	boolean	GLOBAL
<code>secure_auth</code> [460]	boolean	GLOBAL
<code>server_id</code> [1858]	numeric	GLOBAL
<code>slave_allow_batching</code> [2155]	boolean	GLOBAL
<code>slave_compressed_protocol</code> [1887]	boolean	GLOBAL
<code>slave_exec_mode</code> [1887]	enumeration	GLOBAL
<code>slave_max_allowed_packet</code> [1869]	numeric	GLOBAL
<code>slave_net_timeout</code> [1883]	numeric	GLOBAL
<code>slave_transaction_retries</code> [1889]	numeric	GLOBAL
<code>slow_launch_time</code> [552]	numeric	GLOBAL
<code>slow_query_log</code> [463]	boolean	GLOBAL
<code>slow_query_log_file</code> [553]	filename	GLOBAL

Variable Name	Variable Type	Variable Scope
<code>sort_buffer_size</code> [554]	numeric	GLOBAL SESSION
<code>sql_auto_is_null</code> [554]	boolean	GLOBAL SESSION
<code>sql_big_selects</code> [555]	boolean	GLOBAL SESSION
<code>sql_big_tables</code> [435]	boolean	GLOBAL SESSION
<code>sql_buffer_result</code> [555]	boolean	GLOBAL SESSION
<code>sql_log_bin</code> [555]	boolean	GLOBAL SESSION
<code>sql_log_off</code> [556]	boolean	GLOBAL SESSION
<code>sql_log_update</code> [556]	boolean	SESSION
<code>sql_low_priority_updates</code> [517]	boolean	GLOBAL SESSION
<code>sql_max_join_size</code> [522]	numeric	GLOBAL SESSION
<code>sql_mode</code> [464]	set	GLOBAL SESSION
<code>sql_notes</code> [556]	boolean	GLOBAL SESSION
<code>sql_quote_show_create</code> [556]	boolean	GLOBAL SESSION
<code>sql_safe_updates</code> [557]	boolean	GLOBAL SESSION
<code>sql_select_limit</code> [557]	numeric	GLOBAL SESSION
<code>sql_slave_skip_counter</code> [1890]	numeric	GLOBAL
<code>sql_warnings</code> [557]	boolean	GLOBAL SESSION
<code>storage_engine</code> [559]	enumeration	GLOBAL SESSION
<code>stored_program_cache</code> [559]	numeric	GLOBAL
<code>sync_binlog</code> [1904]	numeric	GLOBAL
<code>sync_frm</code> [559]	boolean	GLOBAL
<code>sync_master_info</code> [1890]	numeric	GLOBAL
<code>sync_relay_log</code> [1890]	numeric	GLOBAL
<code>sync_relay_log_info</code> [1891]	numeric	GLOBAL
<code>table_definition_cache</code> [560]	numeric	GLOBAL
<code>table_lock_wait_timeout</code> [561]	numeric	GLOBAL
<code>table_open_cache</code> [561]	numeric	GLOBAL
<code>table_type</code> [559]	enumeration	GLOBAL SESSION
<code>thread_cache_size</code> [561]	numeric	GLOBAL
<code>thread_pool_high_priority_connection</code> [563]	numeric	GLOBAL SESSION
<code>thread_pool_max_unused_threads</code> [564]	numeric	GLOBAL
<code>thread_pool_prio_kickup_timer</code> [565]	numeric	GLOBAL SESSION
<code>thread_pool_stall_limit</code> [566]	numeric	GLOBAL
<code>time_zone</code> [567]	string	GLOBAL SESSION
<code>timed_mutexes</code> [567]	boolean	GLOBAL
<code>timestamp</code> [567]	numeric	SESSION
<code>tmp_table_size</code> [568]	numeric	GLOBAL SESSION
<code>transaction_alloc_block_size</code> [569]	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
<code>transaction_prealloc_size</code> [569]	numeric	GLOBAL SESSION
<code>tx_isolation</code> [570]	enumeration	GLOBAL SESSION
<code>unique_checks</code> [571]	boolean	GLOBAL SESSION
<code>updatable_views_with_limit</code> [571]	boolean	GLOBAL SESSION
<code>wait_timeout</code> [572]	numeric	GLOBAL SESSION

5.1.6. Server Status Variables

The server maintains many status variables that provide information about its operation. You can view these variables and their values by using the `SHOW [GLOBAL | SESSION] STATUS` statement (see Section 13.7.5.36, “`SHOW STATUS Syntax`”). The optional `GLOBAL` keyword aggregates the values over all connections, and `SESSION` shows the values for the current connection.

```
mysql> SHOW GLOBAL STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Aborted_clients | 0 |
| Aborted_connects | 0 |
| Bytes_received | 155372598 |
| Bytes_sent | 1176560426 |
| ... | ... |
| Connections | 30023 |
| Created_tmp_disk_tables | 0 |
| Created_tmp_files | 3 |
| Created_tmp_tables | 2 |
| ... | ... |
| Threads_created | 217 |
| Threads_running | 88 |
| Uptime | 1389872 |
+-----+-----+
```

Many status variables are reset to 0 by the `FLUSH STATUS` statement.

The following table lists all available server status variables:

Table 5.4. Status Variable Summary

Variable Name	Variable Type	Variable Scope
<code>Aborted_clients</code> [596]	numeric	GLOBAL
<code>Aborted_connects</code> [596]	numeric	GLOBAL
<code>Binlog_cache_disk_use</code> [596]	numeric	GLOBAL
<code>Binlog_cache_use</code> [596]	numeric	GLOBAL
<code>Binlog_stmt_cache_disk_use</code> [596]	numeric	GLOBAL
<code>Binlog_stmt_cache_use</code> [597]	numeric	GLOBAL
<code>Bytes_received</code> [597]	numeric	GLOBAL SESSION
<code>Bytes_sent</code> [597]	numeric	GLOBAL SESSION
<code>Com_admin_commands</code> [597]	numeric	GLOBAL SESSION
<code>Com_alter_db</code> [597]	numeric	GLOBAL SESSION
<code>Com_alter_db_upgrade</code> [597]	numeric	GLOBAL SESSION
<code>Com_alter_event</code> [597]	numeric	GLOBAL SESSION

Server Status Variables

Variable Name	Variable Type	Variable Scope
Com_alter_function [597]	numeric	GLOBAL SESSION
Com_alter_procedure [597]	numeric	GLOBAL SESSION
Com_alter_server [597]	numeric	GLOBAL SESSION
Com_alter_table [597]	numeric	GLOBAL SESSION
Com_alter_tablespace [597]	numeric	GLOBAL SESSION
Com_analyze [597]	numeric	GLOBAL SESSION
Com_assign_to_keycache [597]	numeric	GLOBAL SESSION
Com_begin [597]	numeric	GLOBAL SESSION
Com_binlog [597]	numeric	GLOBAL SESSION
Com_call_procedure [597]	numeric	GLOBAL SESSION
Com_change_db [597]	numeric	GLOBAL SESSION
Com_change_master [597]	numeric	GLOBAL SESSION
Com_check [597]	numeric	GLOBAL SESSION
Com_checksum [597]	numeric	GLOBAL SESSION
Com_commit [597]	numeric	GLOBAL SESSION
Com_create_db [597]	numeric	GLOBAL SESSION
Com_create_event [597]	numeric	GLOBAL SESSION
Com_create_function [597]	numeric	GLOBAL SESSION
Com_create_index [597]	numeric	GLOBAL SESSION
Com_create_procedure [597]	numeric	GLOBAL SESSION
Com_create_server [597]	numeric	GLOBAL SESSION
Com_create_table [597]	numeric	GLOBAL SESSION
Com_create_trigger [597]	numeric	GLOBAL SESSION
Com_create_udf [597]	numeric	GLOBAL SESSION
Com_create_user [597]	numeric	GLOBAL SESSION
Com_create_view [597]	numeric	GLOBAL SESSION
Com_dealloc_sql [597]	numeric	GLOBAL SESSION
Com_delete [597]	numeric	GLOBAL SESSION
Com_delete_multi [597]	numeric	GLOBAL SESSION
Com_do [597]	numeric	GLOBAL SESSION
Com_drop_db [597]	numeric	GLOBAL SESSION
Com_drop_event [597]	numeric	GLOBAL SESSION
Com_drop_function [597]	numeric	GLOBAL SESSION
Com_drop_index [597]	numeric	GLOBAL SESSION
Com_drop_procedure [597]	numeric	GLOBAL SESSION
Com_drop_server [597]	numeric	GLOBAL SESSION
Com_drop_table [597]	numeric	GLOBAL SESSION
Com_drop_trigger [597]	numeric	GLOBAL SESSION

Server Status Variables

Variable Name	Variable Type	Variable Scope
Com_drop_user [597]	numeric	GLOBAL SESSION
Com_drop_view [597]	numeric	GLOBAL SESSION
Com_empty_query [597]	numeric	GLOBAL SESSION
Com_execute_sql [597]	numeric	GLOBAL SESSION
Com_flush [597]	numeric	GLOBAL SESSION
Com_grant [597]	numeric	GLOBAL SESSION
Com_ha_close [597]	numeric	GLOBAL SESSION
Com_ha_open [597]	numeric	GLOBAL SESSION
Com_ha_read [597]	numeric	GLOBAL SESSION
Com_help [597]	numeric	GLOBAL SESSION
Com_insert [597]	numeric	GLOBAL SESSION
Com_insert_select [597]	numeric	GLOBAL SESSION
Com_install_plugin [597]	numeric	GLOBAL SESSION
Com_kill [597]	numeric	GLOBAL SESSION
Com_load [597]	numeric	GLOBAL SESSION
Com_lock_tables [597]	numeric	GLOBAL SESSION
Com_optimize [597]	numeric	GLOBAL SESSION
Com_preload_keys [597]	numeric	GLOBAL SESSION
Com_prepare_sql [597]	numeric	GLOBAL SESSION
Com_purge [597]	numeric	GLOBAL SESSION
Com_purge_before_date [597]	numeric	GLOBAL SESSION
Com_release_savepoint [597]	numeric	GLOBAL SESSION
Com_rename_table [597]	numeric	GLOBAL SESSION
Com_rename_user [597]	numeric	GLOBAL SESSION
Com_repair [597]	numeric	GLOBAL SESSION
Com_replace [597]	numeric	GLOBAL SESSION
Com_replace_select [597]	numeric	GLOBAL SESSION
Com_reset [597]	numeric	GLOBAL SESSION
Com_resignal [597]	numeric	GLOBAL SESSION
Com_revoke [597]	numeric	GLOBAL SESSION
Com_revoke_all [597]	numeric	GLOBAL SESSION
Com_rollback [597]	numeric	GLOBAL SESSION
Com_rollback_to_savepoint [597]	numeric	GLOBAL SESSION
Com_savepoint [597]	numeric	GLOBAL SESSION
Com_select [597]	numeric	GLOBAL SESSION
Com_set_option [597]	numeric	GLOBAL SESSION
Com_show_authors [597]	numeric	GLOBAL SESSION
Com_show_binlog_events [597]	numeric	GLOBAL SESSION

Server Status Variables

Variable Name	Variable Type	Variable Scope
Com_show_binlogs [597]	numeric	GLOBAL SESSION
Com_show_charsets [597]	numeric	GLOBAL SESSION
Com_show_collations [597]	numeric	GLOBAL SESSION
Com_show_contributors [597]	numeric	GLOBAL SESSION
Com_show_create_db [597]	numeric	GLOBAL SESSION
Com_show_create_event [597]	numeric	GLOBAL SESSION
Com_show_create_func [597]	numeric	GLOBAL SESSION
Com_show_create_proc [597]	numeric	GLOBAL SESSION
Com_show_create_table [597]	numeric	GLOBAL SESSION
Com_show_create_trigger [597]	numeric	GLOBAL SESSION
Com_show_databases [597]	numeric	GLOBAL SESSION
Com_show_engine_logs [597]	numeric	GLOBAL SESSION
Com_show_engine_mutex [597]	numeric	GLOBAL SESSION
Com_show_engine_status [597]	numeric	GLOBAL SESSION
Com_show_errors [597]	numeric	GLOBAL SESSION
Com_show_events [597]	numeric	GLOBAL SESSION
Com_show_fields [597]	numeric	GLOBAL SESSION
Com_show_function_code [597]	numeric	GLOBAL SESSION
Com_show_function_status [597]	numeric	GLOBAL SESSION
Com_show_grants [597]	numeric	GLOBAL SESSION
Com_show_keys [597]	numeric	GLOBAL SESSION
Com_show_logs [597]	numeric	GLOBAL SESSION
Com_show_master_status [597]	numeric	GLOBAL SESSION
Com_show_ndb_status [597]	numeric	GLOBAL SESSION
Com_show_new_master [597]	numeric	GLOBAL SESSION
Com_show_open_tables [597]	numeric	GLOBAL SESSION
Com_show_plugins [597]	numeric	GLOBAL SESSION
Com_show_privileges [597]	numeric	GLOBAL SESSION
Com_show_procedure_code [597]	numeric	GLOBAL SESSION
Com_show_procedure_status [597]	numeric	GLOBAL SESSION
Com_show_processlist [597]	numeric	GLOBAL SESSION
Com_show_profile [597]	numeric	GLOBAL SESSION
Com_show_profiles [597]	numeric	GLOBAL SESSION
Com_show_relaylog_events [597]	numeric	GLOBAL SESSION
Com_show_slave_hosts [597]	numeric	GLOBAL SESSION
Com_show_slave_status [597]	numeric	GLOBAL SESSION
Com_show_status [597]	numeric	GLOBAL SESSION
Com_show_storage_engines [597]	numeric	GLOBAL SESSION

Server Status Variables

Variable Name	Variable Type	Variable Scope
Com_show_table_status [597]	numeric	GLOBAL SESSION
Com_show_tables [597]	numeric	GLOBAL SESSION
Com_show_triggers [597]	numeric	GLOBAL SESSION
Com_show_variables [597]	numeric	GLOBAL SESSION
Com_show_warnings [597]	numeric	GLOBAL SESSION
Com_signal [597]	numeric	GLOBAL SESSION
Com_slave_start [597]	numeric	GLOBAL SESSION
Com_slave_stop [597]	numeric	GLOBAL SESSION
Com_stmt_close [597]	numeric	GLOBAL SESSION
Com_stmt_execute [597]	numeric	GLOBAL SESSION
Com_stmt_fetch [597]	numeric	GLOBAL SESSION
Com_stmt_prepare [597]	numeric	GLOBAL SESSION
Com_stmt_reprepare [597]	numeric	GLOBAL SESSION
Com_stmt_reset [597]	numeric	GLOBAL SESSION
Com_stmt_send_long_data [597]	numeric	GLOBAL SESSION
Com_truncate [597]	numeric	GLOBAL SESSION
Com_uninstall_plugin [597]	numeric	GLOBAL SESSION
Com_unlock_tables [597]	numeric	GLOBAL SESSION
Com_update [597]	numeric	GLOBAL SESSION
Com_update_multi [597]	numeric	GLOBAL SESSION
Com_xa_commit [597]	numeric	GLOBAL SESSION
Com_xa_end [597]	numeric	GLOBAL SESSION
Com_xa_prepare [597]	numeric	GLOBAL SESSION
Com_xa_recover [597]	numeric	GLOBAL SESSION
Com_xa_rollback [597]	numeric	GLOBAL SESSION
Com_xa_start [597]	numeric	GLOBAL SESSION
Compression [597]	numeric	SESSION
Connections [598]	numeric	GLOBAL
Created_tmp_disk_tables [598]	numeric	GLOBAL SESSION
Created_tmp_files [598]	numeric	GLOBAL
Created_tmp_tables [598]	numeric	GLOBAL SESSION
Delayed_errors [598]	numeric	GLOBAL
Delayed_insert_threads [598]	numeric	GLOBAL
Delayed_writes [598]	numeric	GLOBAL
Flush_commands [598]	numeric	GLOBAL
Handler_commit [598]	numeric	GLOBAL SESSION
Handler_delete [599]	numeric	GLOBAL SESSION
Handler_discover [2161]	numeric	GLOBAL SESSION

Server Status Variables

Variable Name	Variable Type	Variable Scope
Handler_prepare [599]	numeric	GLOBAL SESSION
Handler_read_first [599]	numeric	GLOBAL SESSION
Handler_read_key [599]	numeric	GLOBAL SESSION
Handler_read_last [599]	numeric	GLOBAL SESSION
Handler_read_next [599]	numeric	GLOBAL SESSION
Handler_read_prev [599]	numeric	GLOBAL SESSION
Handler_read_rnd [599]	numeric	GLOBAL SESSION
Handler_read_rnd_next [599]	numeric	GLOBAL SESSION
Handler_rollback [599]	numeric	GLOBAL SESSION
Handler_savepoint [599]	numeric	GLOBAL SESSION
Handler_savepoint_rollback [599]	numeric	GLOBAL SESSION
Handler_update [600]	numeric	GLOBAL SESSION
Handler_write [600]	numeric	GLOBAL SESSION
Innodb_buffer_pool_pages_data [600]	numeric	GLOBAL
Innodb_buffer_pool_pages_dirty [600]	numeric	GLOBAL
Innodb_buffer_pool_pages_flushed [600]	numeric	GLOBAL
Innodb_buffer_pool_pages_free [600]	numeric	GLOBAL
Innodb_buffer_pool_pages_latched [600]	numeric	GLOBAL
Innodb_buffer_pool_pages_misc [600]	numeric	GLOBAL
Innodb_buffer_pool_pages_total [600]	numeric	GLOBAL
Innodb_buffer_pool_read_ahead [600]	numeric	GLOBAL
Innodb_buffer_pool_read_ahead_evicted [600]	numeric	GLOBAL
Innodb_buffer_pool_read_requests [600]	numeric	GLOBAL
Innodb_buffer_pool_reads [600]	numeric	GLOBAL
Innodb_buffer_pool_wait_free [601]	numeric	GLOBAL
Innodb_buffer_pool_write_requests [601]	numeric	GLOBAL
Innodb_data_fsyncs [601]	numeric	GLOBAL
Innodb_data_pending_fsyncs [601]	numeric	GLOBAL
Innodb_data_pending_reads [601]	numeric	GLOBAL
Innodb_data_pending_writes [601]	numeric	GLOBAL
Innodb_data_read [601]	numeric	GLOBAL
Innodb_data_reads [601]	numeric	GLOBAL
Innodb_data_writes [601]	numeric	GLOBAL
Innodb_data_written [601]	numeric	GLOBAL
Innodb_dblwr_pages_written [601]	numeric	GLOBAL
Innodb_dblwr_writes [601]	numeric	GLOBAL
Innodb_have_atomic_builtins [601]	numeric	GLOBAL
Innodb_log_waits [602]	numeric	GLOBAL

Server Status Variables

Variable Name	Variable Type	Variable Scope
Innodb_log_write_requests [602]	numeric	GLOBAL
Innodb_log_writes [602]	numeric	GLOBAL
Innodb_os_log_fsyncs [602]	numeric	GLOBAL
Innodb_os_log_pending_fsyncs [602]	numeric	GLOBAL
Innodb_os_log_pending_writes [602]	numeric	GLOBAL
Innodb_os_log_written [602]	numeric	GLOBAL
Innodb_page_size [602]	numeric	GLOBAL
Innodb_pages_created [602]	numeric	GLOBAL
Innodb_pages_read [602]	numeric	GLOBAL
Innodb_pages_written [602]	numeric	GLOBAL
Innodb_row_lock_current_waits [602]	numeric	GLOBAL
Innodb_row_lock_time [602]	numeric	GLOBAL
Innodb_row_lock_time_avg [602]	numeric	GLOBAL
Innodb_row_lock_time_max [602]	numeric	GLOBAL
Innodb_row_lock_waits [603]	numeric	GLOBAL
Innodb_rows_deleted [603]	numeric	GLOBAL
Innodb_rows_inserted [603]	numeric	GLOBAL
Innodb_rows_read [603]	numeric	GLOBAL
Innodb_rows_updated [603]	numeric	GLOBAL
Innodb_truncated_status_writes [603]	numeric	GLOBAL
Key_blocks_not_flushed [603]	numeric	GLOBAL
Key_blocks_unused [603]	numeric	GLOBAL
Key_blocks_used [603]	numeric	GLOBAL
Key_read_requests [603]	numeric	GLOBAL
Key_reads [603]	numeric	GLOBAL
Key_write_requests [603]	numeric	GLOBAL
Key_writes [603]	numeric	GLOBAL
Last_query_cost [603]	numeric	SESSION
Max_used_connections [604]	numeric	GLOBAL
Ndb_api_bytes_received_count [2164]	numeric	GLOBAL
Ndb_api_bytes_received_count_session [2164]	numeric	SESSION
Ndb_api_bytes_received_count_slave [2164]	numeric	GLOBAL
Ndb_api_bytes_sent_count [2164]	numeric	GLOBAL
Ndb_api_bytes_sent_count_session [2163]	numeric	SESSION
Ndb_api_bytes_sent_count_slave [2164]	numeric	GLOBAL
Ndb_api_event_bytes_count [2171]	numeric	GLOBAL
Ndb_api_event_bytes_count_injector [2171]	numeric	GLOBAL
Ndb_api_event_data_count [2171]	numeric	GLOBAL

Server Status Variables

Variable Name	Variable Type	Variable Scope
Ndb_api_event_data_count_injector [2171]	numeric	GLOBAL
Ndb_api_event_nondata_count [2171]	numeric	GLOBAL
Ndb_api_event_nondata_count_injector [2171]	numeric	GLOBAL
Ndb_api_pk_op_count [2167]	numeric	GLOBAL
Ndb_api_pk_op_count_session [2166]	numeric	SESSION
Ndb_api_pk_op_count_slave [2167]	numeric	GLOBAL
Ndb_api_pruned_scan_count [2169]	numeric	GLOBAL
Ndb_api_pruned_scan_count_session [2168]	numeric	SESSION
Ndb_api_pruned_scan_count_slave [2169]	numeric	GLOBAL
Ndb_api_range_scan_count [2168]	numeric	GLOBAL
Ndb_api_range_scan_count_session [2168]	numeric	SESSION
Ndb_api_range_scan_count_slave [2168]	numeric	GLOBAL
Ndb_api_read_row_count [2170]	numeric	GLOBAL
Ndb_api_read_row_count_session [2169]	numeric	SESSION
Ndb_api_read_row_count_slave [2170]	numeric	GLOBAL
Ndb_api_scan_batch_count [2169]	numeric	GLOBAL
Ndb_api_scan_batch_count_session [2169]	numeric	SESSION
Ndb_api_scan_batch_count_slave [2169]	numeric	GLOBAL
Ndb_api_table_scan_count [2168]	numeric	GLOBAL
Ndb_api_table_scan_count_session [2167]	numeric	SESSION
Ndb_api_table_scan_count_slave [2168]	numeric	GLOBAL
Ndb_api_trans_abort_count [2166]	numeric	GLOBAL
Ndb_api_trans_abort_count_session [2165]	numeric	SESSION
Ndb_api_trans_abort_count_slave [2165]	numeric	GLOBAL
Ndb_api_trans_close_count [2166]	numeric	GLOBAL
Ndb_api_trans_close_count_session [2166]	numeric	SESSION
Ndb_api_trans_close_count_slave [2166]	numeric	GLOBAL
Ndb_api_trans_commit_count [2165]	numeric	GLOBAL
Ndb_api_trans_commit_count_session [2165]	numeric	SESSION
Ndb_api_trans_commit_count_slave [2165]	numeric	GLOBAL
Ndb_api_trans_local_read_row_count [2170]	numeric	GLOBAL
Ndb_api_trans_local_read_row_count_session [2170]	numeric	SESSION
Ndb_api_trans_local_read_row_count_slave [2170]	numeric	GLOBAL
Ndb_api_trans_start_count [2165]	numeric	GLOBAL
Ndb_api_trans_start_count_session [2164]	numeric	SESSION
Ndb_api_trans_start_count_slave [2165]	numeric	GLOBAL
Ndb_api_uk_op_count [2167]	numeric	GLOBAL
Ndb_api_uk_op_count_session [2167]	numeric	SESSION

Server Status Variables

Variable Name	Variable Type	Variable Scope
Ndb_api_uk_op_count_slave [2167]	numeric	GLOBAL
Ndb_api_wait_exec_complete_count [2162]	numeric	GLOBAL
Ndb_api_wait_exec_complete_count_session [2162]	numeric	SESSION
Ndb_api_wait_exec_complete_count_slave [2161]	numeric	GLOBAL
Ndb_api_wait_meta_request_count [2163]	numeric	GLOBAL
Ndb_api_wait_meta_request_count_session [2162]	numeric	SESSION
Ndb_api_wait_meta_request_count_slave [2162]	numeric	GLOBAL
Ndb_api_wait_nanos_count [2163]	numeric	GLOBAL
Ndb_api_wait_nanos_count_session [2163]	numeric	SESSION
Ndb_api_wait_nanos_count_slave [2163]	numeric	GLOBAL
Ndb_api_wait_scan_result_count [2162]	numeric	GLOBAL
Ndb_api_wait_scan_result_count_session [2162]	numeric	SESSION
Ndb_api_wait_scan_result_count_slave [2162]	numeric	GLOBAL
ndb_cluster_connection_pool [2139]	numeric	GLOBAL
Ndb_cluster_node_id [2171]	numeric	GLOBAL SESSION
Ndb_config_from_host [2172]	numeric	GLOBAL SESSION
Ndb_conflict_fn_epoch [2172]	numeric	GLOBAL
Ndb_conflict_fn_epoch_trans [2172]	numeric	GLOBAL
Ndb_conflict_fn_max [2172]	numeric	GLOBAL
Ndb_conflict_fn_old [2172]	numeric	GLOBAL
Ndb_conflict_trans_conflict_commit_count [2173]	numeric	GLOBAL
Ndb_conflict_trans_detect_iter_count [2173]	numeric	GLOBAL
Ndb_conflict_trans_reject_count [2173]	numeric	GLOBAL
Ndb_conflict_trans_row_conflict_count [2173]	numeric	GLOBAL
Ndb_conflict_trans_row_reject_count [2173]	numeric	GLOBAL
ndb_execute_count [2173]	numeric	GLOBAL
ndb-nodeid [2143]	numeric	GLOBAL
Ndb_number_of_data_nodes [2173]	numeric	GLOBAL
ndb_pruned_scan_count [2174]	numeric	GLOBAL
Ndb_pushed_queries_defined [2173]	numeric	GLOBAL
Ndb_pushed_queries_dropped [2173]	numeric	GLOBAL
Ndb_pushed_queries_executed [2174]	numeric	GLOBAL
ndb_pushed_reads [2174]	numeric	GLOBAL
ndb_scan_count [2174]	numeric	GLOBAL
Not_flushed_delayed_rows [604]	numeric	GLOBAL
Open_files [604]	numeric	GLOBAL
Open_streams [604]	numeric	GLOBAL
Open_table_definitions [604]	numeric	GLOBAL

Server Status Variables

Variable Name	Variable Type	Variable Scope
Open_tables [604]	numeric	GLOBAL SESSION
Opened_files [604]	numeric	GLOBAL
Opened_table_definitions [604]	numeric	GLOBAL SESSION
Opened_tables [604]	numeric	GLOBAL SESSION
Performance_schema_cond_classes_lost [2559]	numeric	GLOBAL
Performance_schema_cond_instances_lost [2559]	numeric	GLOBAL
Performance_schema_file_classes_lost [2559]	numeric	GLOBAL
Performance_schema_file_handles_lost [2559]	numeric	GLOBAL
Performance_schema_file_instances_lost [2559]	numeric	GLOBAL
Performance_schema_locker_lost [2559]	numeric	GLOBAL
Performance_schema_mutex_classes_lost [2559]	numeric	GLOBAL
Performance_schema_mutex_instances_lost [2559]	numeric	GLOBAL
Performance_schema_rwlock_classes_lost [2559]	numeric	GLOBAL
Performance_schema_rwlock_instances_lost [2559]	numeric	GLOBAL
Performance_schema_table_handles_lost [2559]	numeric	GLOBAL
Performance_schema_table_instances_lost [2559]	numeric	GLOBAL
Performance_schema_thread_classes_lost [2559]	numeric	GLOBAL
Performance_schema_thread_instances_lost [2559]	numeric	GLOBAL
Prepared_stmt_count [604]	numeric	GLOBAL
Qcache_free_blocks [604]	numeric	GLOBAL
Qcache_free_memory [605]	numeric	GLOBAL
Qcache_hits [605]	numeric	GLOBAL
Qcache_inserts [605]	numeric	GLOBAL
Qcache_lowmem_prunes [605]	numeric	GLOBAL
Qcache_not_cached [605]	numeric	GLOBAL
Qcache_queries_in_cache [605]	numeric	GLOBAL
Qcache_total_blocks [605]	numeric	GLOBAL
Queries [605]	numeric	GLOBAL SESSION
Questions [605]	numeric	GLOBAL SESSION
Rpl_semi_sync_master_clients [605]	numeric	GLOBAL
Rpl_semi_sync_master_net_avg_wait_time [605]	numeric	GLOBAL
Rpl_semi_sync_master_net_wait_time [605]	numeric	GLOBAL
Rpl_semi_sync_master_net_waits [606]	numeric	GLOBAL
Rpl_semi_sync_master_no_times [606]	numeric	GLOBAL
Rpl_semi_sync_master_no_tx [606]	numeric	GLOBAL
Rpl_semi_sync_master_status [606]	boolean	GLOBAL
Rpl_semi_sync_master_timefunc_failures [606]	numeric	GLOBAL
Rpl_semi_sync_master_tx_avg_wait_time [606]	numeric	GLOBAL

Server Status Variables

Variable Name	Variable Type	Variable Scope
Rpl_semi_sync_master_tx_wait_time [606]	numeric	GLOBAL
Rpl_semi_sync_master_tx_waits [606]	numeric	GLOBAL
Rpl_semi_sync_master_wait_pos_backtraverse_num [606]	numeric	GLOBAL
Rpl_semi_sync_master_wait_sessions [607]	numeric	GLOBAL
Rpl_semi_sync_master_yes_tx [607]	numeric	GLOBAL
Rpl_semi_sync_slave_status [607]	boolean	GLOBAL
Rpl_status [607]	string	GLOBAL
Select_full_join [607]	numeric	GLOBAL SESSION
Select_full_range_join [607]	numeric	GLOBAL SESSION
Select_range [607]	numeric	GLOBAL SESSION
Select_range_check [607]	numeric	GLOBAL SESSION
Select_scan [607]	numeric	GLOBAL SESSION
Slave_heartbeat_period [607]		GLOBAL
Slave_open_temp_tables [607]	numeric	GLOBAL
Slave_received_heartbeats [607]		GLOBAL
Slave_retried_transactions [608]	numeric	GLOBAL
Slave_running [608]	boolean	GLOBAL
Slow_launch_threads [608]	numeric	GLOBAL SESSION
Slow_queries [608]	numeric	GLOBAL SESSION
Sort_merge_passes [608]	numeric	GLOBAL SESSION
Sort_range [608]	numeric	GLOBAL SESSION
Sort_rows [608]	numeric	GLOBAL SESSION
Sort_scan [608]	numeric	GLOBAL SESSION
Ssl_accept_renegotiates [608]	numeric	GLOBAL
Ssl_accepts [608]	numeric	GLOBAL
Ssl_callback_cache_hits [608]	numeric	GLOBAL
Ssl_cipher [608]	string	GLOBAL SESSION
Ssl_cipher_list [608]	string	GLOBAL SESSION
Ssl_client_connects [608]	numeric	GLOBAL
Ssl_connect_renegotiates [609]	numeric	GLOBAL
Ssl_ctx_verify_depth [609]	numeric	GLOBAL
Ssl_ctx_verify_mode [609]	numeric	GLOBAL
Ssl_default_timeout [609]	numeric	GLOBAL SESSION
Ssl_finished_accepts [609]	numeric	GLOBAL
Ssl_finished_connects [609]	numeric	GLOBAL
Ssl_session_cache_hits [609]	numeric	GLOBAL
Ssl_session_cache_misses [609]	numeric	GLOBAL
Ssl_session_cache_mode [609]	string	GLOBAL

Variable Name	Variable Type	Variable Scope
Ssl_session_cache_overflows [609]	numeric	GLOBAL
Ssl_session_cache_size [609]	numeric	GLOBAL
Ssl_session_cache_timeouts [609]	numeric	GLOBAL
Ssl_sessions_reused [609]	numeric	GLOBAL SESSION
Ssl_used_session_cache_entries [609]	numeric	GLOBAL
Ssl_verify_depth [609]	numeric	GLOBAL SESSION
Ssl_verify_mode [610]	numeric	GLOBAL SESSION
Ssl_version [610]	string	GLOBAL SESSION
Table_locks_immediate [610]	numeric	GLOBAL
Table_locks_waited [610]	numeric	GLOBAL
Tc_log_max_pages_used [610]	numeric	GLOBAL
Tc_log_page_size [610]	numeric	GLOBAL
Tc_log_page_waits [610]	numeric	GLOBAL
Threads_cached [610]	numeric	GLOBAL
Threads_connected [610]	numeric	GLOBAL
Threads_created [610]	numeric	GLOBAL
Threads_running [611]	numeric	GLOBAL
Uptime [611]	numeric	GLOBAL
Uptime_since_flush_status [611]	numeric	GLOBAL

The status variables have the following meanings. For meanings of status variables specific to MySQL Cluster, see [Section 17.3.4.4, “MySQL Cluster Status Variables”](#).

- [Aborted_clients \[596\]](#)

The number of connections that were aborted because the client died without closing the connection properly. See [Section C.5.2.11, “Communication Errors and Aborted Connections”](#).

- [Aborted_connects \[596\]](#)

The number of failed attempts to connect to the MySQL server. See [Section C.5.2.11, “Communication Errors and Aborted Connections”](#).

- [Binlog_cache_disk_use \[596\]](#)

The number of transactions that used the binary log cache but that exceeded the value of [binlog_cache_size \[1898\]](#) and used a temporary file to store changes from the transaction.

In MySQL versions 5.5.3 through 5.5.8, this variable also included the number of nontransactional statements that caused the binary log transaction cache to be written to disk. Beginning with MySQL 5.5.9, the number of such nontransactional statements is tracked separately in the [Binlog_stmt_cache_disk_use \[596\]](#) status variable.

- [Binlog_cache_use \[596\]](#)

The number of transactions that used the binary log cache.

- [Binlog_stmt_cache_disk_use \[596\]](#)

The number of nontransaction statements that used the binary log statement cache but that exceeded the value of `binlog_stmt_cache_size` [1901] and used a temporary file to store those statements.

- `Binlog_stmt_cache_use` [597]

The number of nontransactional statements that used the binary log statement cache.

- `Bytes_received` [597]

The number of bytes received from all clients.

- `Bytes_sent` [597]

The number of bytes sent to all clients.

- `Com_xxx`

The `Com_xxx` statement counter variables indicate the number of times each `xxx` statement has been executed. There is one status variable for each type of statement. For example, `Com_delete` and `Com_insert` count `DELETE` and `INSERT` statements, respectively. However, if a query result is returned from query cache, the server increments the `Qcache_hits` [605] status variable, not `Com_select`. See Section 8.9.3.4, “Query Cache Status and Maintenance”.

All of the `Com_stmt_xxx` variables are increased even if a prepared statement argument is unknown or an error occurred during execution. In other words, their values correspond to the number of requests issued, not to the number of requests successfully completed.

The `Com_stmt_xxx` status variables are as follows:

- `Com_stmt_prepare`
- `Com_stmt_execute`
- `Com_stmt_fetch`
- `Com_stmt_send_long_data`
- `Com_stmt_reset`
- `Com_stmt_close`

Those variables stand for prepared statement commands. Their names refer to the `COM_xxx` command set used in the network layer. In other words, their values increase whenever prepared statement API calls such as `mysql_stmt_prepare()`, `mysql_stmt_execute()`, and so forth are executed. However, `Com_stmt_prepare`, `Com_stmt_execute` and `Com_stmt_close` also increase for `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE`, respectively. Additionally, the values of the older statement counter variables `Com_prepare_sql`, `Com_execute_sql`, and `Com_dealloc_sql` increase for the `PREPARE`, `EXECUTE`, and `DEALLOCATE PREPARE` statements. `Com_stmt_fetch` stands for the total number of network round-trips issued when fetching from cursors.

`Com_stmt_reprepare` indicates the number of times statements were automatically reprepared by the server after metadata changes to tables or views referred to by the statement. A reprepare operation increments `Com_stmt_reprepare`, and also `Com_stmt_prepare`.

- `Compression` [597]

Whether the client connection uses compression in the client/server protocol.

- [Connections \[598\]](#)

The number of connection attempts (successful or not) to the MySQL server.

- [Created_tmp_disk_tables \[598\]](#)

The number of internal on-disk temporary tables created by the server while executing statements.

If an internal temporary table is created initially as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the [tmp_table_size \[568\]](#) and [max_heap_table_size \[521\]](#) values. If [Created_tmp_disk_tables \[598\]](#) is large, you may want to increase the [tmp_table_size \[568\]](#) or [max_heap_table_size \[521\]](#) values. value to lessen the likelihood that internal temporary tables in memory will be converted to on-disk tables.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the [Created_tmp_disk_tables \[598\]](#) and [Created_tmp_tables \[598\]](#) variables.

See also [Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#).

- [Created_tmp_files \[598\]](#)

How many temporary files `mysqld` has created.

- [Created_tmp_tables \[598\]](#)

The number of internal temporary tables created by the server while executing statements.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the [Created_tmp_disk_tables \[598\]](#) and [Created_tmp_tables \[598\]](#) variables.

See also [Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#).

Each invocation of the `SHOW STATUS` statement uses an internal temporary table and increments the global [Created_tmp_tables \[598\]](#) value.

- [Delayed_errors \[598\]](#)

The number of rows written with `INSERT DELAYED` for which some error occurred (probably `duplicate key`).

- [Delayed_insert_threads \[598\]](#)

The number of `INSERT DELAYED` handler threads in use.

- [Delayed_writes \[598\]](#)

The number of `INSERT DELAYED` rows written.

- [Flush_commands \[598\]](#)

The number of times the server flushes tables, whether because a user executed a `FLUSH TABLES` statement or due to internal server operation. It is also incremented by receipt of a `COM_REFRESH` packet. This is in contrast to [Com_flush \[597\]](#), which indicates how many `FLUSH` statements have been executed, whether `FLUSH TABLES`, `FLUSH LOGS`, and so forth.

- [Handler_commit \[598\]](#)

The number of internal `COMMIT` statements.

- [Handler_delete \[599\]](#)

The number of times that rows have been deleted from tables.

- [Handler_prepare \[599\]](#)

A counter for the prepare phase of two-phase commit operations.

- [Handler_read_first \[599\]](#)

The number of times the first entry in an index was read. If this value is high, it suggests that the server is doing a lot of full index scans; for example, `SELECT coll FROM foo`, assuming that `coll` is indexed.

- [Handler_read_key \[599\]](#)

The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- [Handler_read_last \[599\]](#)

The number of requests to read the last key in an index. With `ORDER BY`, the server will issue a first-key request followed by several next-key requests, whereas with `ORDER BY DESC`, the server will issue a last-key request followed by several previous-key requests. This variable was added in MySQL 5.5.7.

- [Handler_read_next \[599\]](#)

The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- [Handler_read_prev \[599\]](#)

The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`.

- [Handler_read_rnd \[599\]](#)

The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that do not use keys properly.

- [Handler_read_rnd_next \[599\]](#)

The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- [Handler_rollback \[599\]](#)

The number of requests for a storage engine to perform a rollback operation.

- [Handler_savepoint \[599\]](#)

The number of requests for a storage engine to place a savepoint.

- [Handler_savepoint_rollback \[599\]](#)

The number of requests for a storage engine to roll back to a savepoint.

- [Handler_update \[600\]](#)

The number of requests to update a row in a table.

- [Handler_write \[600\]](#)

The number of requests to insert a row in a table.

- [Innodb_buffer_pool_pages_data \[600\]](#)

The number of pages containing data (dirty or clean).

- [Innodb_buffer_pool_pages_dirty \[600\]](#)

The number of pages currently dirty.

- [Innodb_buffer_pool_pages_flushed \[600\]](#)

The number of buffer pool page-flush requests.

- [Innodb_buffer_pool_pages_free \[600\]](#)

The number of free pages.

- [Innodb_buffer_pool_pages_latched \[600\]](#)

The number of latched pages in `InnoDB` buffer pool. These are pages currently being read or written or that cannot be flushed or removed for some other reason. Calculation of this variable is expensive, so it is available only when the `UNIV_DEBUG` system is defined at server build time.

- [Innodb_buffer_pool_pages_misc \[600\]](#)

The number of pages that are busy because they have been allocated for administrative overhead such as row locks or the adaptive hash index. This value can also be calculated as [Innodb_buffer_pool_pages_total \[600\]](#) – [Innodb_buffer_pool_pages_free \[600\]](#) – [Innodb_buffer_pool_pages_data \[600\]](#).

- [Innodb_buffer_pool_pages_total \[600\]](#)

The total size of the buffer pool, in pages.

- [Innodb_buffer_pool_read_ahead \[600\]](#)

The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread.

- [Innodb_buffer_pool_read_ahead_evicted \[600\]](#)

The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.

- [Innodb_buffer_pool_read_requests \[600\]](#)

The number of logical read requests `InnoDB` has done.

- [Innodb_buffer_pool_reads \[600\]](#)

The number of logical reads that `InnoDB` could not satisfy from the buffer pool, and had to read directly from the disk.

- `Innodb_buffer_pool_wait_free` [601]

Normally, writes to the `InnoDB` buffer pool happen in the background. However, if it is necessary to read or create a page and no clean pages are available, it is also necessary to wait for pages to be flushed first. This counter counts instances of these waits. If the buffer pool size has been set properly, this value should be small.

- `Innodb_buffer_pool_write_requests` [601]

The number writes done to the `InnoDB` buffer pool.

- `Innodb_data_fsyncs` [601]

The number of `fsync()` operations so far.

- `Innodb_data_pending_fsyncs` [601]

The current number of pending `fsync()` operations.

- `Innodb_data_pending_reads` [601]

The current number of pending reads.

- `Innodb_data_pending_writes` [601]

The current number of pending writes.

- `Innodb_data_read` [601]

The amount of data read since the server was started.

- `Innodb_data_reads` [601]

The total number of data reads.

- `Innodb_data_writes` [601]

The total number of data writes.

- `Innodb_data_written` [601]

The amount of data written so far, in bytes.

- `Innodb_dblwr_pages_written` [601]

The number of pages that have been written for doublewrite operations. See [Section 14.3.12.1, “InnoDB Disk I/O”](#).

- `Innodb_dblwr_writes` [601]

The number of doublewrite operations that have been performed. See [Section 14.3.12.1, “InnoDB Disk I/O”](#).

- `Innodb_have_atomic_builtins` [601]

Indicates whether the server was built with atomic instructions.

- [Innodb_log_waits \[602\]](#)

The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing.
- [Innodb_log_write_requests \[602\]](#)

The number of log write requests.
- [Innodb_log_writes \[602\]](#)

The number of physical writes to the log file.
- [Innodb_os_log_fsyncs \[602\]](#)

The number of `fsync()` writes done to the log file.
- [Innodb_os_log_pending_fsyncs \[602\]](#)

The number of pending log file `fsync()` operations.
- [Innodb_os_log_pending_writes \[602\]](#)

The number of pending log file writes.
- [Innodb_os_log_written \[602\]](#)

The number of bytes written to the log file.
- [Innodb_page_size \[602\]](#)

The compiled-in `InnoDB` page size (default 16KB). Many values are counted in pages; the page size enables them to be easily converted to bytes.
- [Innodb_pages_created \[602\]](#)

The number of pages created.
- [Innodb_pages_read \[602\]](#)

The number of pages read.
- [Innodb_pages_written \[602\]](#)

The number of pages written.
- [Innodb_row_lock_current_waits \[602\]](#)

The number of row locks currently being waited for.
- [Innodb_row_lock_time \[602\]](#)

The total time spent in acquiring row locks, in milliseconds.
- [Innodb_row_lock_time_avg \[602\]](#)

The average time to acquire a row lock, in milliseconds.
- [Innodb_row_lock_time_max \[602\]](#)

The maximum time to acquire a row lock, in milliseconds.

- [Innodb_row_lock_waits \[603\]](#)

The number of times a row lock had to be waited for.
- [Innodb_rows_deleted \[603\]](#)

The number of rows deleted from `InnoDB` tables.
- [Innodb_rows_inserted \[603\]](#)

The number of rows inserted into `InnoDB` tables.
- [Innodb_rows_read \[603\]](#)

The number of rows read from `InnoDB` tables.
- [Innodb_rows_updated \[603\]](#)

The number of rows updated in `InnoDB` tables.
- [Innodb_truncated_status_writes \[603\]](#)

The number of times output from the `SHOW ENGINE INNODB STATUS` is truncated. Monitoring applications that parse the output from this command can test this value before and after issuing the `SHOW ENGINE` command, to confirm if the output is complete or not.
- [Key_blocks_not_flushed \[603\]](#)

The number of key blocks in the key cache that have changed but have not yet been flushed to disk.
- [Key_blocks_unused \[603\]](#)

The number of unused blocks in the key cache. You can use this value to determine how much of the key cache is in use; see the discussion of [key_buffer_size \[508\]](#) in [Section 5.1.4, “Server System Variables”](#).
- [Key_blocks_used \[603\]](#)

The number of used blocks in the key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.
- [Key_read_requests \[603\]](#)

The number of requests to read a key block from the cache.
- [Key_reads \[603\]](#)

The number of physical reads of a key block from disk. If [Key_reads \[603\]](#) is large, then your [key_buffer_size \[508\]](#) value is probably too small. The cache miss rate can be calculated as $\text{Key_reads [603]} / \text{Key_read_requests [603]}$.
- [Key_write_requests \[603\]](#)

The number of requests to write a key block to the cache.
- [Key_writes \[603\]](#)

The number of physical writes of a key block to disk.
- [Last_query_cost \[603\]](#)

The total cost of the last compiled query as computed by the query optimizer. This is useful for comparing the cost of different query plans for the same query. The default value of 0 means that no query has been compiled yet. The default value is 0. `Last_query_cost` [603] has session scope.

The `Last_query_cost` [603] value can be computed accurately only for simple “flat” queries, not complex queries such as those with subqueries or `UNION`. For the latter, the value is set to 0.

- `Max_used_connections` [604]

The maximum number of connections that have been in use simultaneously since the server started.

- `Not_flushed_delayed_rows` [604]

The number of rows waiting to be written in `INSERT DELAYED` queues.

- `Open_files` [604]

The number of files that are open. This count includes regular files opened by the server. It does not include other types of files such as sockets or pipes. Also, the count does not include files that storage engines open using their own internal functions rather than asking the server level to do so.

- `Open_streams` [604]

The number of streams that are open (used mainly for logging).

- `Open_table_definitions` [604]

The number of cached `.frm` files.

- `Open_tables` [604]

The number of tables that are open.

- `Opened_files` [604]

The number of files that have been opened with `my_open()` (a `mysys` library function). Parts of the server that open files without using this function do not increment the count.

- `Opened_table_definitions` [604]

The number of `.frm` files that have been cached.

- `Opened_tables` [604]

The number of tables that have been opened. If `Opened_tables` [604] is big, your `table_open_cache` [561] value is probably too small.

- `Performance_schema_xxx`

Performance Schema status variables are listed in [Section 21.9, “Performance Schema Status Variables”](#).

- `Prepared_stmt_count` [604]

The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` [523] system variable.)

- `Qcache_free_blocks` [604]

The number of free memory blocks in the query cache.

- [Qcache_free_memory \[605\]](#)

The amount of free memory for the query cache.

- [Qcache_hits \[605\]](#)

The number of query cache hits.

- [Qcache_inserts \[605\]](#)

The number of queries added to the query cache.

- [Qcache_lowmem_prunes \[605\]](#)

The number of queries that were deleted from the query cache because of low memory.

- [Qcache_not_cached \[605\]](#)

The number of noncached queries (not cacheable, or not cached due to the [query_cache_type \[542\]](#) setting).

- [Qcache_queries_in_cache \[605\]](#)

The number of queries registered in the query cache.

- [Qcache_total_blocks \[605\]](#)

The total number of blocks in the query cache.

- [Queries \[605\]](#)

The number of statements executed by the server. This variable includes statements executed within stored programs, unlike the [Questions](#) variable. It does not count [COM_PING](#) or [COM_STATISTICS](#) commands.

- [Questions \[605\]](#)

The number of statements executed by the server. This includes only statements sent to the server by clients and not statements executed within stored programs, unlike the [Queries](#) variable. This variable does not count [COM_PING](#), [COM_STATISTICS](#), [COM_STMT_PREPARE](#), [COM_STMT_CLOSE](#), or [COM_STMT_RESET](#) commands.

- [Rpl_semi_sync_master_clients \[605\]](#)

The number of semisynchronous slaves.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_net_avg_wait_time \[605\]](#)

The average time in microseconds the master waited for a slave reply.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_net_wait_time \[605\]](#)

The total time in microseconds the master waited for slave replies.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_net_waits \[606\]](#)

The total number of times the master waited for slave replies.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_no_times \[606\]](#)

The number of times the master turned off semisynchronous replication.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_no_tx \[606\]](#)

The number of commits that were not acknowledged successfully by a slave.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_status \[606\]](#)

Whether semisynchronous replication currently is operational on the master. The value is `ON` if the plugin has been enabled and a commit acknowledgment has occurred. It is `OFF` if the plugin is not enabled or the master has fallen back to asynchronous replication due to commit acknowledgment timeout.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_timefunc_failures \[606\]](#)

The number of times the master failed when calling time functions such as `gettimeofday()`.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_tx_avg_wait_time \[606\]](#)

The average time in microseconds the master waited for each transaction.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_tx_wait_time \[606\]](#)

The total time in microseconds the master waited for transactions.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_tx_waits \[606\]](#)

The total number of times the master waited for transactions.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_wait_pos_backtraverse \[606\]](#)

The total number of times the master waited for an event with binary coordinates lower than events waited for previously. This can occur when the order in which transactions start waiting for a reply is different from the order in which their binary log events are written.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_wait_sessions \[607\]](#)

The number of sessions currently waiting for slave replies.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_master_yes_tx \[607\]](#)

The number of commits that were acknowledged successfully by a slave.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- [Rpl_semi_sync_slave_status \[607\]](#)

Whether semisynchronous replication currently is operational on the slave. This is `ON` if the plugin has been enabled and the slave I/O thread is running, `OFF` otherwise.

This variable is available only if the slave-side semisynchronous replication plugin is installed.

- [Rpl_status \[607\]](#)

The status of fail-safe replication (not implemented). This variable is unused and is removed in MySQL 5.6.

- [Select_full_join \[607\]](#)

The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables.

- [Select_full_range_join \[607\]](#)

The number of joins that used a range search on a reference table.

- [Select_range \[607\]](#)

The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

- [Select_range_check \[607\]](#)

The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

- [Select_scan \[607\]](#)

The number of joins that did a full scan of the first table.

- [Slave_heartbeat_period \[607\]](#)

Shows the replication heartbeat interval (in seconds) on a replication slave.

- [Slave_open_temp_tables \[607\]](#)

The number of temporary tables that the slave SQL thread currently has open. If the value is greater than zero, it is not safe to shut down the slave; see [Section 16.4.1.21, “Replication and Temporary Tables”](#).

- [Slave_received_heartbeats \[607\]](#)

This counter increments with each replication heartbeat received by a replication slave since the last time that the slave was restarted or reset, or a `CHANGE MASTER TO` statement was issued.

- [Slave_retried_transactions \[608\]](#)

The total number of times since startup that the replication slave SQL thread has retried transactions.

- [Slave_running \[608\]](#)

This is `ON` if this server is a replication slave that is connected to a replication master, and both the I/O and SQL threads are running; otherwise, it is `OFF`.

- [Slow_launch_threads \[608\]](#)

The number of threads that have taken more than [slow_launch_time \[552\]](#) seconds to create.

- [Slow_queries \[608\]](#)

The number of queries that have taken more than [long_query_time \[517\]](#) seconds. See [Section 5.2.5, “The Slow Query Log”](#).

- [Sort_merge_passes \[608\]](#)

The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the [sort_buffer_size \[554\]](#) system variable.

- [Sort_range \[608\]](#)

The number of sorts that were done using ranges.

- [Sort_rows \[608\]](#)

The number of sorted rows.

- [Sort_scan \[608\]](#)

The number of sorts that were done by scanning the table.

- [Ssl_accept_renegotiates \[608\]](#)

The number of negotiates needed to establish the connection.

- [Ssl_accepts \[608\]](#)

The number of accepted SSL connections.

- [Ssl_callback_cache_hits \[608\]](#)

The number of callback cache hits.

- [Ssl_cipher \[608\]](#)

The current SSL cipher (empty for non-SSL connections).

- [Ssl_cipher_list \[608\]](#)

The list of possible SSL ciphers.

- [Ssl_client_connects \[608\]](#)

The number of SSL connection attempts to an SSL-enabled master.

- [Ssl_connect_renegotiates \[609\]](#)

The number of negotiates needed to establish the connection to an SSL-enabled master.

- [Ssl_ctx_verify_depth \[609\]](#)

The SSL context verification depth (how many certificates in the chain are tested).

- [Ssl_ctx_verify_mode \[609\]](#)

The SSL context verification mode.

- [Ssl_default_timeout \[609\]](#)

The default SSL timeout.

- [Ssl_finished_accepts \[609\]](#)

The number of successful SSL connections to the server.

- [Ssl_finished_connects \[609\]](#)

The number of successful slave connections to an SSL-enabled master.

- [Ssl_session_cache_hits \[609\]](#)

The number of SSL session cache hits.

- [Ssl_session_cache_misses \[609\]](#)

The number of SSL session cache misses.

- [Ssl_session_cache_mode \[609\]](#)

The SSL session cache mode.

- [Ssl_session_cache_overflows \[609\]](#)

The number of SSL session cache overflows.

- [Ssl_session_cache_size \[609\]](#)

The SSL session cache size.

- [Ssl_session_cache_timeouts \[609\]](#)

The number of SSL session cache timeouts.

- [Ssl_sessions_reused \[609\]](#)

How many SSL connections were reused from the cache.

- [Ssl_used_session_cache_entries \[609\]](#)

How many SSL session cache entries were used.

- [Ssl_verify_depth \[609\]](#)

The verification depth for replication SSL connections.

- [Ssl_verify_mode \[610\]](#)

The verification mode for replication SSL connections.

- [Ssl_version \[610\]](#)

The SSL version number.

- [Table_locks_immediate \[610\]](#)

The number of times that a request for a table lock could be granted immediately.

- [Table_locks_waited \[610\]](#)

The number of times that a request for a table lock could not be granted immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

- [Tc_log_max_pages_used \[610\]](#)

For the memory-mapped implementation of the log that is used by `mysqld` when it acts as the transaction coordinator for recovery of internal XA transactions, this variable indicates the largest number of pages used for the log since the server started. If the product of [Tc_log_max_pages_used \[610\]](#) and [Tc_log_page_size \[610\]](#) is always significantly less than the log size, the size is larger than necessary and can be reduced. (The size is set by the `--log-tc-size [451]` option. Currently, this variable is unused: It is unneeded for binary log-based recovery, and the memory-mapped recovery log method is not used unless the number of storage engines capable of two-phase commit is greater than one. (InnoDB is the only applicable engine.)

- [Tc_log_page_size \[610\]](#)

The page size used for the memory-mapped implementation of the XA recovery log. The default value is determined using `getpagesize()`. Currently, this variable is unused for the same reasons as described for [Tc_log_max_pages_used \[610\]](#).

- [Tc_log_page_waits \[610\]](#)

For the memory-mapped implementation of the recovery log, this variable increments each time the server was not able to commit a transaction and had to wait for a free page in the log. If this value is large, you might want to increase the log size (with the `--log-tc-size [451]` option). For binary log-based recovery, this variable increments each time the binary log cannot be closed because there are two-phase commits in progress. (The close operation waits until all such transactions are finished.)

- [Threads_cached \[610\]](#)

The number of threads in the thread cache.

- [Threads_connected \[610\]](#)

The number of currently open connections.

- [Threads_created \[610\]](#)

The number of threads created to handle connections. If [Threads_created \[610\]](#) is big, you may want to increase the [thread_cache_size \[561\]](#) value. The cache miss rate can be calculated as [Threads_created \[610\]/Connections \[598\]](#).

- [Threads_running \[611\]](#)
The number of threads that are not sleeping.
- [Uptime \[611\]](#)
The number of seconds that the server has been up.
- [Uptime_since_flush_status \[611\]](#)
The number of seconds since the most recent `FLUSH STATUS` statement.

5.1.7. Server SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients. This capability enables each application to tailor the server's operating mode to its own requirements.

For answers to some questions that are often asked about server SQL modes in MySQL, see [Section B.3, "MySQL 5.5 FAQ: Server SQL Mode"](#).

Modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

When working with `InnoDB` tables, consider also the [innodb_strict_mode \[1572\]](#) configuration option. It enables additional error checks for `InnoDB` tables, as listed in [Section 14.4.8.4, "InnoDB Strict Mode"](#).

Setting the SQL Mode

You can set the default SQL mode by starting `mysqld` with the `--sql-mode="modes" [464]` option, or by using `sql-mode="modes" [464]` in `my.cnf` (Unix operating systems) or `my.ini` (Windows). `modes` is a list of different modes separated by comma (",") characters. The default value is empty (no modes set). The `modes` value also can be empty (`--sql-mode="" [464]` on the command line, or `sql-mode="" [464]` in `my.cnf` on Unix systems or in `my.ini` on Windows) if you want to clear it explicitly.

You can change the SQL mode at runtime by using a `SET [GLOBAL|SESSION] sql_mode='modes'` statement to set the `sql_mode [556]` system value. Setting the `GLOBAL` variable requires the `SUPER [697]` privilege and affects the operation of all clients that connect from that time on. Setting the `SESSION` variable affects only the current client. Any client can change its own session `sql_mode [556]` value at any time.



Important

SQL mode and user-defined partitioning. Changing the server SQL mode after creating and inserting data into partitioned tables can cause major changes in the behavior of such tables, and could lead to loss or corruption of data. It is strongly recommended that you never change the SQL mode once you have created tables employing user-defined partitioning.

When replicating partitioned tables, differing SQL modes on master and slave can also lead to problems. For best results, you should always use the same server SQL mode on the master and on the slave.

See [Section 18.5, "Restrictions and Limitations on Partitioning"](#), for more information.

You can retrieve the current global or session `sql_mode [556]` value with the following statements:

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

Most Important SQL Modes

The most important `sql_mode` [556] values are probably these:

- [ANSI](#) [617]

This mode changes syntax and behavior to conform more closely to standard SQL. It is one of the special [combination modes](#) listed at the end of this section.

- [STRICT_TRANS_TABLES](#) [617]

If a value could not be inserted as given into a transactional table, abort the statement. For a nontransactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More detail is given later in this section.

- [TRADITIONAL](#) [618]

Make MySQL behave like a “traditional” SQL database system. A simple description of this mode is “give an error instead of a warning” when inserting an incorrect value into a column. It is one of the special [combination modes](#) listed at the end of this section.



Note

The `INSERT/UPDATE` aborts as soon as the error is noticed. This may not be what you want if you are using a nontransactional storage engine, because data changes made prior to the error may not be rolled back, resulting in a “partially done” update.

When this manual refers to “strict mode,” it means a mode where at least one of [STRICT_TRANS_TABLES](#) [617] or [STRICT_ALL_TABLES](#) [616] is enabled.

Full List of SQL Modes

The following list describes all supported modes:

- [ALLOW_INVALID_DATES](#) [612]

Do not perform full checking of dates. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This is very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation). This mode applies to `DATE` and `DATETIME` columns. It does not apply `TIMESTAMP` columns, which always require a valid date.

The server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable [ALLOW_INVALID_DATES](#) [612].

- [ANSI_QUOTES](#) [612]

Treat `"` as an identifier quote character (like the ``` quote character) and not as a string quote character. You can still use ``` to quote identifiers with this mode enabled. With [ANSI_QUOTES](#) [612] enabled, you cannot use double quotation marks to quote literal strings, because it is interpreted as an identifier.

- [ERROR_FOR_DIVISION_BY_ZERO \[613\]](#)

Produce an error in strict mode (otherwise a warning) when a division by zero (or `MOD(x, 0)` [1127]) occurs during an `INSERT` or `UPDATE`. If this mode is not enabled, MySQL instead returns `NULL` for divisions by zero. For `INSERT IGNORE` or `UPDATE IGNORE`, MySQL generates a warning for divisions by zero, but the result of the operation is `NULL`.

- [HIGH_NOT_PRECEDENCE \[613\]](#)

The precedence of the `NOT` [1094] operator is such that expressions such as `NOT a BETWEEN b AND c` are parsed as `NOT (a BETWEEN b AND c)`. In some older versions of MySQL, the expression was parsed as `(NOT a) BETWEEN b AND c`. The old higher-precedence behavior can be obtained by enabling the `HIGH_NOT_PRECEDENCE` [613] SQL mode.

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 1
```

- [IGNORE_SPACE \[613\]](#)

Permit spaces between a function name and the “(” character. This causes built-in function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in [Section 9.2, “Schema Object Names”](#). For example, because there is a `COUNT()` [1204] function, the use of `count` as a table name in the following statement causes an error:

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```

The table name should be quoted:

```
mysql> CREATE TABLE `count` (i INT);
Query OK, 0 rows affected (0.00 sec)
```

The `IGNORE_SPACE` [613] SQL mode applies to built-in functions, not to user-defined functions or stored functions. It is always permissible to have spaces after a UDF or stored function name, regardless of whether `IGNORE_SPACE` [613] is enabled.

For further discussion of `IGNORE_SPACE` [613], see [Section 9.2.4, “Function Name Parsing and Resolution”](#).

- [NO_AUTO_CREATE_USER \[613\]](#)

Prevent the `GRANT` statement from automatically creating new users if it would otherwise do so, unless authentication information is specified. The statement must specify a nonempty password using `IDENTIFIED BY` or an authentication plugin using `IDENTIFIED WITH`.

- [NO_AUTO_VALUE_ON_ZERO \[613\]](#)

`NO_AUTO_VALUE_ON_ZERO` [613] affects handling of `AUTO_INCREMENT` columns. Normally, you generate the next sequence number for the column by inserting either `NULL` or `0` into it. `NO_AUTO_VALUE_ON_ZERO` [613] suppresses this behavior for `0` so that only `NULL` generates the next sequence number.

This mode can be useful if `0` has been stored in a table's `AUTO_INCREMENT` column. (Storing `0` is not a recommended practice, by the way.) For example, if you dump the table with `mysqldump` and then reload it, MySQL normally generates new sequence numbers when it encounters the

0 values, resulting in a table with contents different from the one that was dumped. Enabling `NO_AUTO_VALUE_ON_ZERO` [613] before reloading the dump file solves this problem. `mysqldump` now automatically includes in its output a statement that enables `NO_AUTO_VALUE_ON_ZERO` [613], to avoid this problem.

- `NO_BACKSLASH_ESCAPES` [614]

Disable the use of the backslash character (“\”) as an escape character within strings. With this mode enabled, backslash becomes an ordinary character like any other.

- `NO_DIR_IN_CREATE` [614]

When creating a table, ignore all `INDEX DIRECTORY` and `DATA DIRECTORY` directives. This option is useful on slave replication servers.

- `NO_ENGINE_SUBSTITUTION` [614]

Control automatic substitution of the default storage engine when a statement such as `CREATE TABLE` or `ALTER TABLE` specifies a storage engine that is disabled or not compiled in.

Because storage engines can be pluggable at runtime, unavailable engines are treated the same way:

With `NO_ENGINE_SUBSTITUTION` [614] disabled, for `CREATE TABLE` the default engine is used and a warning occurs if the desired engine is unavailable. For `ALTER TABLE`, a warning occurs and the table is not altered.

With `NO_ENGINE_SUBSTITUTION` [614] enabled, an error occurs and the table is not created or altered if the desired engine is unavailable.

- `NO_FIELD_OPTIONS` [614]

Do not print MySQL-specific column options in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode.

- `NO_KEY_OPTIONS` [614]

Do not print MySQL-specific index options in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode.

- `NO_TABLE_OPTIONS` [614]

Do not print MySQL-specific table options (such as `ENGINE`) in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode.

- `NO_UNSIGNED_SUBTRACTION` [614]

By default, subtraction between integer operands produces an `UNSIGNED` result if any operand is `UNSIGNED`. When `NO_UNSIGNED_SUBTRACTION` [614] is enabled, the subtraction result is signed, *even if any operand is unsigned*. For example, compare the type of column `c2` in table `t1` with that of column `c2` in table `t2`:

```
mysql> SET sql_mode='';
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| c2    | bigint(21) unsigned |      |     | 0       |      |
+-----+-----+-----+-----+-----+-----+
```

```
mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| c2    | bigint(21)   |      |     | 0        |       |
+-----+-----+-----+-----+-----+-----+
```

Note that this means that `BIGINT UNSIGNED` is not 100% usable in all contexts. See [Section 12.10, “Cast Functions and Operators”](#).

```
mysql> SET sql_mode = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
| 18446744073709551615 |
+-----+

mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
| -1 |
+-----+
```

- [NO_ZERO_DATE \[615\]](#)

In strict mode, do not permit `'0000-00-00'` as a valid date. You can still insert zero dates with the `IGNORE` option. When not in strict mode, the date is accepted but a warning is generated.

- [NO_ZERO_IN_DATE \[615\]](#)

In strict mode, do not accept dates where the year part is nonzero but the month or day part is 0 (for example, `'0000-00-00'` is legal but `'2010-00-01'` and `'2010-01-00'` are not). If used with the `IGNORE` option, MySQL inserts a `'0000-00-00'` date for any such date. When not in strict mode, the date is accepted but a warning is generated.

- [ONLY_FULL_GROUP_BY \[615\]](#)

Do not permit queries for which the select list or `HAVING` list refers to nonaggregated columns that are not named in the `GROUP BY` clause.

The following queries are invalid with `ONLY_FULL_GROUP_BY [615]` enabled. The first is invalid because `address` in the select list is not named in the `GROUP BY` clause, and the second because `max_age` in the `HAVING` clause is not named in the `GROUP BY` clause:

```
mysql> SELECT name, address, MAX(age) FROM t GROUP BY name;
ERROR 1055 (42000): 't.address' isn't in GROUP BY
```

```
mysql> SELECT name, MAX(age) AS max_age FROM t GROUP BY name
-> HAVING max_age < 30;
Empty set (0.00 sec)
ERROR 1463 (42000): Non-grouping field 'max_age' is used in HAVING clause
```

In the second example, the query could be rewritten to use `HAVING MAX(age)` instead, so that the reference is to a column named in an aggregate function. (`max_age` fails because it is an aggregate function.)

In addition, if a query has aggregate functions and no `GROUP BY` clause, it cannot have nonaggregated columns in the select list or `ORDER BY` list:

```
mysql> SELECT name, MAX(age) FROM t;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

For more information, see [Section 12.16.3, “MySQL Extensions to GROUP BY”](#).

- [PAD_CHAR_TO_FULL_LENGTH \[616\]](#)

By default, trailing spaces are trimmed from `CHAR` column values on retrieval. If [PAD_CHAR_TO_FULL_LENGTH \[616\]](#) is enabled, trimming does not occur and retrieved `CHAR` values are padded to their full length. This mode does not apply to `VARCHAR` columns, for which trailing spaces are retained on retrieval.

```
mysql> CREATE TABLE t1 (c1 CHAR(10));
Query OK, 0 rows affected (0.37 sec)

mysql> INSERT INTO t1 (c1) VALUES('xy');
Query OK, 1 row affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+-----+-----+
| c1    | CHAR_LENGTH(c1) |
+-----+-----+
| xy    |                2 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+-----+-----+
| c1          | CHAR_LENGTH(c1) |
+-----+-----+
| xy          |                10 |
+-----+-----+
1 row in set (0.00 sec)
```

- [PIPES_AS_CONCAT \[616\]](#)

Treat `||` [\[1095\]](#) as a string concatenation operator (same as `CONCAT()` [\[1102\]](#)) rather than as a synonym for `OR` [\[1095\]](#).

- [REAL_AS_FLOAT \[616\]](#)

Treat `REAL` as a synonym for `FLOAT`. By default, MySQL treats `REAL` as a synonym for `DOUBLE`.

- [STRICT_ALL_TABLES \[616\]](#)

Enable strict mode for all storage engines. Invalid data values are rejected. Additional detail follows.

- [STRICT_TRANS_TABLES \[617\]](#)

Enable strict mode for transactional storage engines, and when possible for nontransactional storage engines. Additional details follow.

Strict mode controls how MySQL handles input values that are invalid or missing. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a non-`NULL` column that has no explicit `DEFAULT` clause in its definition. (For a `NULL` column, `NULL` is inserted if the value is missing.)

For transactional tables, an error occurs for invalid or missing values in a statement when either of the [STRICT_ALL_TABLES \[616\]](#) or [STRICT_TRANS_TABLES \[617\]](#) modes are enabled. The statement is aborted and rolled back.

For nontransactional tables, the behavior is the same for either mode, if the bad value occurs in the first row to be inserted or updated. The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict option is enabled:

- For [STRICT_ALL_TABLES \[616\]](#), MySQL returns an error and ignores the rest of the rows. However, in this case, the earlier rows still have been inserted or updated. This means that you might get a partial update, which might not be what you want. To avoid this, it is best to use single-row statements because these can be aborted without changing the table.
- For [STRICT_TRANS_TABLES \[617\]](#), MySQL converts an invalid value to the closest valid value for the column and insert the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and continues processing the statement. Implicit defaults are described in [Section 11.5, “Data Type Default Values”](#).

Strict mode disallows invalid date values such as `'2004-04-31'`. It does not disallow dates with zero month or day parts such as `'2004-04-00'` or “zero” dates. To disallow these as well, enable the [NO_ZERO_IN_DATE \[615\]](#) and [NO_ZERO_DATE \[615\]](#) SQL modes in addition to strict mode.

If you are not using strict mode (that is, neither [STRICT_TRANS_TABLES \[617\]](#) nor [STRICT_ALL_TABLES \[616\]](#) is enabled), MySQL inserts adjusted values for invalid or missing values and produces warnings. In strict mode, you can produce this behavior by using `INSERT IGNORE` or `UPDATE IGNORE`. See [Section 13.7.5.41, “SHOW WARNINGS Syntax”](#).

Strict mode does not affect whether foreign key constraints are checked. [foreign_key_checks \[500\]](#) can be used for that. (See [Section 5.1.4, “Server System Variables”](#).)

Combination Modes

The following special modes are provided as shorthand for combinations of mode values from the preceding list.

The descriptions include all mode values that are available in the most recent version of MySQL. For older versions, a combination mode does not include individual mode values that are not available except in newer versions.

- [ANSI \[617\]](#)

Equivalent to [REAL_AS_FLOAT \[616\]](#), [PIPES_AS_CONCAT \[616\]](#), [ANSI_QUOTES \[612\]](#), [IGNORE_SPACE \[613\]](#).

[ANSI \[617\]](#) mode also causes the server to return an error for queries where a set function `S` with an outer reference `S(outer_ref)` cannot be aggregated in the outer query against which the outer reference has been resolved. This is such a query:

```
SELECT * FROM t1 WHERE t1.a IN (SELECT MAX(t1.b) FROM t2 WHERE ...);
```

Here, `MAX(t1.b)` [1205] cannot be aggregated in the outer query because it appears in the `WHERE` clause of that query. Standard SQL requires an error in this situation. If `ANSI` [617] mode is not enabled, the server treats `S(outer_ref)` in such queries the same way that it would interpret `S(const)`.

See [Section 1.8.3, “Running MySQL in ANSI Mode”](#).

- [DB2](#) [618]

Equivalent to [PIPES_AS_CONCAT](#) [616], [ANSI_QUOTES](#) [612], [IGNORE_SPACE](#) [613], [NO_KEY_OPTIONS](#) [614], [NO_TABLE_OPTIONS](#) [614], [NO_FIELD_OPTIONS](#) [614].

- [MAXDB](#) [618]

Equivalent to [PIPES_AS_CONCAT](#) [616], [ANSI_QUOTES](#) [612], [IGNORE_SPACE](#) [613], [NO_KEY_OPTIONS](#) [614], [NO_TABLE_OPTIONS](#) [614], [NO_FIELD_OPTIONS](#) [614], [NO_AUTO_CREATE_USER](#) [613].

- [MSSQL](#) [618]

Equivalent to [PIPES_AS_CONCAT](#) [616], [ANSI_QUOTES](#) [612], [IGNORE_SPACE](#) [613], [NO_KEY_OPTIONS](#) [614], [NO_TABLE_OPTIONS](#) [614], [NO_FIELD_OPTIONS](#) [614].

- [MYSQL323](#) [618]

Equivalent to [NO_FIELD_OPTIONS](#) [614], [HIGH_NOT_PRECEDENCE](#) [613].

- [MYSQL40](#) [618]

Equivalent to [NO_FIELD_OPTIONS](#) [614], [HIGH_NOT_PRECEDENCE](#) [613].

- [ORACLE](#) [618]

Equivalent to [PIPES_AS_CONCAT](#) [616], [ANSI_QUOTES](#) [612], [IGNORE_SPACE](#) [613], [NO_KEY_OPTIONS](#) [614], [NO_TABLE_OPTIONS](#) [614], [NO_FIELD_OPTIONS](#) [614], [NO_AUTO_CREATE_USER](#) [613].

- [POSTGRESQL](#) [618]

Equivalent to [PIPES_AS_CONCAT](#) [616], [ANSI_QUOTES](#) [612], [IGNORE_SPACE](#) [613], [NO_KEY_OPTIONS](#) [614], [NO_TABLE_OPTIONS](#) [614], [NO_FIELD_OPTIONS](#) [614].

- [TRADITIONAL](#) [618]

Equivalent to [STRICT_TRANS_TABLES](#) [617], [STRICT_ALL_TABLES](#) [616], [NO_ZERO_IN_DATE](#) [615], [NO_ZERO_DATE](#) [615], [ERROR_FOR_DIVISION_BY_ZERO](#) [613], [NO_AUTO_CREATE_USER](#) [613], and [NO_ENGINE_SUBSTITUTION](#) [614].

5.1.8. Server Plugins

MySQL supports a plugin API that enables creation of server components. Plugins can be loaded at server startup, or loaded and unloaded at runtime without restarting the server. The components supported by this interface include, but are not limited to, storage engines, full-text parser plugins, partitioning support, and server extensions.

5.1.8.1. Installing and Uninstalling Plugins

Server plugins must be loaded in to the server before they can be used. MySQL enables you to load a plugin at server startup or at runtime. It is also possible to control the activation of loaded plugins at startup, and to unload them at runtime.

- [Installing plugins](#)
- [Controlling plugin activation](#)
- [Uninstalling plugins](#)

Installing Plugins

Server plugins must be known to the server before they can be used. A plugin can be made known several ways, as described here. In the following descriptions, *plugin_name* stands for a plugin name such as `innodb` or `csv`.

Built-in plugins:

A plugin that is built in to the server is known by the server automatically. Normally, the server enables the plugin at startup, although this can be changed with the `--plugin_name` option.

Plugins registered in the `mysql.plugin` table:

The `mysql.plugin` table serves as a registry of plugins. The server normally enables each plugin listed in the table at startup, although whether a given plugin is enabled can be changed with the `--plugin_name` option. If the server is started with the `--skip-grant-tables` [461] option, it does not consult this table and does not load the plugins listed there.

Plugins named with the `--plugin-load` [457] option:

A plugin that is located in a plugin library file can be loaded at server startup with the `--plugin-load` [457] option. Normally, the server enables the plugin at startup, although this can be changed with the `--plugin_name` option.

The option value is a semicolon-separated list of `name=plugin_library` pairs. Each *name* is the name of the plugin, and *plugin_library* is the name of the shared library that contains the plugin code. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. Each library file must be located in the directory named by the `plugin_dir` [537] system variable.

This option does not register any plugin in the `mysql.plugin` table. For subsequent restarts, the server loads the plugin again only if `--plugin-load` [457] is given again. That is, this option effects a one-time installation that persists only for one server invocation.

`--plugin-load` [457] enables plugins to be loaded even when `--skip-grant-tables` [461] is given (which causes the server to ignore the `mysql.plugin` table). `--plugin-load` [457] also enables plugins to be loaded at startup under configurations when plugins cannot be loaded at runtime.

Plugins installed with the `INSTALL PLUGIN` statement:

A plugin that is located in a plugin library file can be loaded at runtime with the `INSTALL PLUGIN` statement. The statement also registers the plugin in the `mysql.plugin` table to cause the server to load it on subsequent restarts. For this reason, `INSTALL PLUGIN` requires the `INSERT` [696] privilege for the `mysql.plugin` table.

If a plugin is named both using a `--plugin-load` [457] option and in the `mysql.plugin` table, the server starts but writes these messages to the error log:

```
100310 19:15:44 [ERROR] Function 'plugin_name' already exists
100310 19:15:44 [Warning] Couldn't load plugin named 'plugin_name'
with soname 'plugin_object_file'.
```

Example: The `--plugin-load` [457] option installs a plugin at server startup. To install a plugin named `myplugin` in a plugin library file named `somepluglib.so`, use these lines in a `my.cnf` file:

```
[mysqld]
plugin-load=myplugin=somepluglib.so
```

In this case, the plugin is not registered in `mysql.plugin`. Restarting the server without the `--plugin-load` [457] option causes the plugin not to be loaded at startup.

Alternatively, the `INSTALL PLUGIN` statement causes the server to load the plugin code from the library file at runtime:

```
mysql> INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
```

`INSTALL PLUGIN` also causes “permanent” plugin registration: The server lists the plugin in the `mysql.plugin` table to ensure that it is loaded on subsequent server restarts.

Many plugins can be loaded either at server startup or at runtime. However, if a plugin is designed such that it must be loaded and initialized during server startup, use `--plugin-load` [457] rather than `INSTALL PLUGIN`.

While a plugin is loaded, information about it is available at runtime from several sources, such as the `INFORMATION_SCHEMA.PLUGINS` table and the `SHOW PLUGINS` statement. For more information, see Section 5.1.8.2, “Obtaining Server Plugin Information”.

Controlling Plugin Activation

If the server knows about a plugin when it starts (for example, because the plugin is named using a `--plugin-load` [457] option or registered in the `mysql.plugin` table), the server loads and enables the plugin by default. It is possible to control activation for such a plugin using a `--plugin_name[=value]` startup option named after the plugin. In the following descriptions, `plugin_name` stands for a plugin name such as `innodb` or `csv`. As with other options, dashes and underscores are interchangeable in option names. For example, `--my_plugin=ON` and `--my-plugin=ON` are equivalent.

- `--plugin_name=OFF`

Tells the server to disable the plugin.

- `--plugin_name[=ON]`

Tells the server to enable the plugin. (Specifying the option as `--plugin_name` without a value has the same effect.) If the plugin fails to initialize, the server runs with the plugin disabled.

- `--plugin_name=FORCE`

Tells the server to enable the plugin, but if plugin initialization fails, the server does not start. In other words, this option forces the server to run with the plugin enabled or not at all.

- `--plugin_name=FORCE_PLUS_PERMANENT`

Like `FORCE`, but in addition prevents the plugin from being unloaded at runtime. If a user attempts to do so with `UNINSTALL PLUGIN`, an error occurs. This value is available as of MySQL 5.5.7.

The values `OFF`, `ON`, `FORCE`, and `FORCE_PLUS_PERMANENT` are not case sensitive.

The activation state for plugins is visible in the `LOAD_OPTION` column of the `INFORMATION_SCHEMA.PLUGINS` table.

Suppose that `CSV`, `BLACKHOLE`, and `ARCHIVE` are built-in pluggable storage engines and that you want the server to load them at startup, subject to these conditions: The server is permitted to run if `CSV` initialization fails, but must require that `BLACKHOLE` initialization succeeds, and `ARCHIVE` should be disabled. To accomplish that, use these lines in an option file:

```
[mysqld]
csv=ON
blackhole=FORCE
archive=OFF
```

The `--enable-plugin_name` option format is supported as a synonym for `--plugin_name=ON`. The `--disable-plugin_name` and `--skip-plugin_name` option formats are supported as synonyms for `--plugin_name=OFF`.

Before MySQL 5.1.36, plugin options are boolean options (see [Section 4.2.3.2, “Program Option Modifiers”](#)). That is, any of these options enable the plugin:

```
--plugin_name
--plugin_name=1
--enable-plugin_name
```

And these options disable the plugin:

```
--plugin_name=0
--disable-plugin_name
--skip-plugin_name
```

If you upgrade to MySQL 5.5 from a version older than 5.1.36 and previously used options of the form `--plugin_name=0` or `--plugin_name=1`, the equivalent options are now `--plugin_name=OFF` and `--plugin_name=ON`, respectively. You also have the choice of requiring plugins to start successfully by using `--plugin_name=FORCE` or `--plugin_name=FORCE_PLUS_PERMANENT`.

If a plugin is disabled, either explicitly with `OFF` or implicitly because it was enabled with `ON` but failed to initialize, aspects of server operation that require the plugin will change. For example, if the plugin implements a storage engine, existing tables for the storage engine become inaccessible, and attempts to create new tables for the storage engine result in tables that use the default storage engine unless the `NO_ENGINE_SUBSTITUTION` [614] SQL mode has been enabled to cause an error to occur instead.

Disabling a plugin may require adjustment to other options. For example, if you start the server using `--skip-innodb` [1546] to disable InnoDB, other `innodb_XXX` options likely will need to be omitted from the startup command. In addition, because InnoDB is the default storage engine, it will not start unless you specify another available storage engine with `--default-storage-engine` [441].

Uninstalling Plugins

A plugin known to the server can be uninstalled to disable it at runtime with the `UNINSTALL PLUGIN` statement. The statement unloads the plugin and removes it from the `mysql.plugin` table if it is registered there. For this reason, `UNINSTALL PLUGIN` statement requires the `DELETE` [696] privilege for the `mysql.plugin` table. With the plugin no longer registered in the table, the server will not load the plugin automatically for subsequent restarts.

`UNINSTALL PLUGIN` can unload plugins regardless of whether they were loaded with `INSTALL PLUGIN` or `--plugin-load` [457].

`UNINSTALL PLUGIN` is subject to these exceptions:

- It cannot unload plugins that are built in to the server. These can be identified as those that have a library name of `NULL` in the output from `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`.
- It cannot unload plugins for which the server was started with `--plugin_name=FORCE_PLUS_PERMANENT`, which prevents plugin unloading at runtime. These can be identified from the `LOAD_OPTION` column of the `INFORMATION_SCHEMA.PLUGINS` table.

5.1.8.2. Obtaining Server Plugin Information

There are several ways to determine which plugins are installed in the server:

- The `INFORMATION_SCHEMA.PLUGINS` table contains a row for each loaded plugin. Any that have a `PLUGIN_LIBRARY` value of `NULL` are built in and cannot be unloaded.

```
mysql> SELECT * FROM information_schema.PLUGINS\G
***** 1. row *****
      PLUGIN_NAME: binlog
      PLUGIN_VERSION: 1.0
      PLUGIN_STATUS: ACTIVE
      PLUGIN_TYPE: STORAGE ENGINE
      PLUGIN_TYPE_VERSION: 50158.0
      PLUGIN_LIBRARY: NULL
      PLUGIN_LIBRARY_VERSION: NULL
      PLUGIN_AUTHOR: MySQL AB
      PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
      PLUGIN_LICENSE: GPL
      LOAD_OPTION: FORCE
...
***** 10. row *****
      PLUGIN_NAME: InnoDB
      PLUGIN_VERSION: 1.0
      PLUGIN_STATUS: ACTIVE
      PLUGIN_TYPE: STORAGE ENGINE
      PLUGIN_TYPE_VERSION: 50158.0
      PLUGIN_LIBRARY: ha_innodb_plugin.so
      PLUGIN_LIBRARY_VERSION: 1.0
      PLUGIN_AUTHOR: Innobase Oy
      PLUGIN_DESCRIPTION: Supports transactions, row-level locking,
                        and foreign keys
      PLUGIN_LICENSE: GPL
      LOAD_OPTION: ON
...
```

- The `SHOW PLUGINS` statement displays a row for each loaded plugin. Any that have a `Library` value of `NULL` are built in and cannot be unloaded.

```
mysql> SHOW PLUGINS\G
***** 1. row *****
      Name: binlog
      Status: ACTIVE
      Type: STORAGE ENGINE
      Library: NULL
      License: GPL
...
***** 10. row *****
      Name: InnoDB
      Status: ACTIVE
      Type: STORAGE ENGINE
      Library: ha_innodb_plugin.so
      License: GPL
...
```

- The `mysql.plugin` table shows which plugins have been registered with `INSTALL PLUGIN`. The table contains only plugin names and library file names, so it does not provide as much information as the `PLUGINS` table or the `SHOW PLUGINS` statement.

5.1.9. IPv6 Support

As of MySQL 5.5.3, support for IPv6 includes these capabilities:

- MySQL Server can accept TCP/IP connections from clients connecting over IPv6. For example, this command connects over IPv6 to the MySQL server on the local host:

```
shell> mysql -h ::1
```

To use this capability, two things must be true:

- Your system must be configured to support IPv6. See [Section 5.1.9.1, “Verifying System Support for IPv6”](#).
- The default MySQL server configuration permits only IPv4 connections, so the server must be configured for IPv6 connections. To permit IPv6 connections in addition to or instead of IPv4 connections, start the server with an appropriate `--bind-address [435]` option. See [Section 5.1.4, “Server System Variables”](#).
- MySQL account names permit IPv6 addresses to enable DBAs to specify privileges for clients that connect to the server over IPv6. See [Section 6.2.3, “Specifying Account Names”](#). IPv6 addresses can be specified in account names in statements such as `CREATE USER`, `GRANT`, and `REVOKE`. For example:

```
mysql> CREATE USER 'bill'@':::1' IDENTIFIED BY 'secret';
mysql> GRANT SELECT ON mydb.* TO 'bill'@':::1';
```

The following sections describe how to set up MySQL so that clients can connect to the server over IPv6.

5.1.9.1. Verifying System Support for IPv6

Before MySQL Server can accept IPv6 connections, the operating system on your server host must support IPv6. As a simple test to determine whether that is true, try this command:

```
shell> ping6 ::1
16 bytes from ::1, icmp_seq=0 hlim=64 time=0.171 ms
16 bytes from ::1, icmp_seq=1 hlim=64 time=0.077 ms
...
```

To produce a description of your system's network interfaces, invoke `ifconfig -a` and look for IPv6 addresses in the output.

If your host does not support IPv6, consult your system documentation for instructions on enabling it. It might be that you need only reconfigure an existing network interface to add an IPv6 address. Or a more extensive change might be needed, such as rebuilding the kernel with IPv6 options enabled.

These links may be helpful in setting up IPv6 on various platforms:

- [Windows XP](#)
- [Gentoo Linux](#)
- [Ubuntu Linux](#)
- [Linux \(Generic\)](#)
- [Mac OS X](#)

5.1.9.2. Configuring the MySQL Server to Permit IPv6 Connections

The MySQL server listens on a single network socket for TCP/IP connections. This socket is bound to a single address, but it is possible for an address to map onto multiple network interfaces. The default address is `0.0.0.0`. To specify an address explicitly, use the `--bind-address=addr` [435] option at server startup, where `addr` is an IPv4 or IPv6 address or a host name. (IPv6 addresses are not supported before MySQL 5.5.3.) If `addr` is a host name, the server resolves the name to an IP address and binds to that address.

The server treats different types of addresses as follows:

- If the address is `0.0.0.0`, the server accepts TCP/IP connections on all server host IPv4 interfaces.
- If the address is `::`, the server accepts TCP/IP connections on all server host IPv4 and IPv6 interfaces. Use this address to permit both IPv4 and IPv6 connections on all server interfaces.
- If the address is an IPv4-mapped address, the server accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if the server is bound to `::ffff:127.0.0.1`, clients can connect using `--host=127.0.0.1` or `--host>::ffff:127.0.0.1`.
- If the address is a “regular” IPv4 or IPv6 address (such as `127.0.0.1` or `::1`), the server accepts TCP/IP connections only for that IPv4 or IPv6 address.

If you intend to bind the server to a specific address, be sure that the `mysql.user` grant table contains an account with administrative privileges that you can use to connect to that address. Otherwise, you will not be able to shut down the server. For example, if you bind the server to `::`, you can connect to it using all existing accounts. But if you bind the server to `::1`, it accepts connections only on that address. In that case, first make sure that the `'root'@'::1'` account is present in the `mysql.user` table so you can still connect to the server to shut it down.

5.1.9.3. Connecting Using the IPv6 Local Host Address

The following procedure shows how to configure MySQL to permit IPv6 connections by clients that connect to the local server using the `::1` local host address. The instructions given here assume that your system supports IPv6.

1. Start the MySQL server with an appropriate `--bind-address` [435] option to permit it to accept IPv6 connections. For example, put the following lines in your server option file and restart the server:

```
[mysqld]
bind-address = ::
```

Alternatively, you can bind the server to `::1`, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see [Section 5.1.9.2, “Configuring the MySQL Server to Permit IPv6 Connections”](#).

2. As an administrator, connect to the server and create an account for a local user who will connect from the `::1` local IPv6 host address:

```
mysql> CREATE USER 'ipv6user'@'::1' IDENTIFIED BY 'ipv6pass';
```

For the permitted syntax of IPv6 addresses in account names, see [Section 6.2.3, “Specifying Account Names”](#). In addition to the `CREATE USER` statement, you can issue `GRANT` statements that give specific privileges to the account, although that is not necessary for the remaining steps in this procedure.

3. Invoke the `mysql` client to connect to the server using the new account:

```
shell> mysql -h ::1 -u ipv6user -pipv6pass
```

4. Try some simple statements that show connection information:

```
mysql> STATUS
...
Connection:  ::1 via TCP/IP
...

mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| ipv6user@::1   |
+-----+
```

5.1.9.4. Connecting Using IPv6 Nonlocal Host Addresses

The following procedure shows how to configure MySQL to permit IPv6 connections by remote clients. It is similar to the preceding procedure for local clients, but the server and client hosts are distinct and each has its own nonlocal IPv6 address. The example uses these addresses:

```
Server host: 2001:db8:0:f101::1
Client host: 2001:db8:0:f101::2
```

These addresses are chosen from the nonroutable address range recommended by [IANA](#) for documentation purposes and suffice for testing on your local network. To accept IPv6 connections from clients outside the local network, the server host must have a public address. If your network provider assigns you an IPv6 address, you can use that. Otherwise, another way to obtain an address is to use an IPv6 broker; see [Section 5.1.9.5](#), “Obtaining an IPv6 Address from a Broker”.

1. Start the MySQL server with an appropriate `--bind-address` [\[435\]](#) option to permit it to accept IPv6 connections. For example, put the following lines in your server option file and restart the server:

```
[mysqld]
bind-address = ::
```

Alternatively, you can bind the server to `2001:db8:0:f101::1`, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see [Section 5.1.9.2](#), “Configuring the MySQL Server to Permit IPv6 Connections”.

2. On the server host (`2001:db8:0:f101::1`), create an account for a user who will connect from the client host (`2001:db8:0:f101::2`):

```
mysql> CREATE USER 'remoteipv6user'@'2001:db8:0:f101::2' IDENTIFIED BY 'remoteipv6pass';
```

3. On the client host (`2001:db8:0:f101::2`), invoke the `mysql` client to connect to the server using the new account:

```
shell> mysql -h 2001:db8:0:f101::1 -u remoteipv6user -premoveipv6pass
```

4. Try some simple statements that show connection information:

```
mysql> STATUS
...
Connection:  2001:db8:0:f101::1 via TCP/IP
...

mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| remoteipv6user@2001:db8:0:f101::2 |
+-----+
```

```
+-----+
```

5.1.9.5. Obtaining an IPv6 Address from a Broker

If you do not have a public IPv6 address that enables your system to communicate over IPv6 outside your local network, you can obtain one from an IPv6 broker. The [Wikipedia IPv6 Tunnel Broker page](#) lists several brokers and their features, such as whether they provide static addresses and the supported routing protocols.

After configuring your server host to use a broker-supplied IPv6 address, start the MySQL server with an appropriate `--bind-address [435]` option to permit the server to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

```
[mysqld]
bind-address = ::
```

Alternatively, you can bind the server to to the specific IPv6 address provided by the broker, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see [Section 5.1.9.2, “Configuring the MySQL Server to Permit IPv6 Connections”](#). In addition, if the broker allocates dynamic addresses, the address provided for your system might change the next time you connect to the broker. If so, any accounts you create that name the original address become invalid. To bind to a specific address but avoid this change-of-address problem, you may be able to arrange with the broker for a static IPv6 address.

The following example shows how to use Freenet6 as the broker and the `gogoc` IPv6 client package on Gentoo Linux.

1. Create a account at Freenet6 by visiting this URL and signing up:

```
http://gogonet.gogo6.com
```

2. After creating the account, go to this URL, sign in, and create a user ID and password for the IPv6 broker:

```
http://gogonet.gogo6.com/page/freenet6-registration
```

3. As `root`, install `gogoc`:

```
shell> emerge gogoc
```

4. Edit `/etc/gogoc/gogoc.conf` to set the `userid` and `password` values. For example:

```
userid=gogouser
passwd=gogopass
```

5. Start `gogoc`:

```
shell> /etc/init.d/gogoc start
```

To start `gogoc` each time your system boots, execute this command:

```
shell> rc-update add gogoc default
```

6. Use `ping6` to try to ping a host:

```
shell> ping6 ipv6.google.com
```

7. To see your IPv6 address:

```
shell> ifconfig tun
```

5.1.10. Server-Side Help

MySQL Server supports a `HELP` statement that returns online information from the MySQL Reference manual (see [Section 13.8.3, “HELP Syntax”](#)). The proper operation of this statement requires that the help tables in the `mysql` database be initialized with help topic information, which is done by processing the contents of the `fill_help_tables.sql` script.

If you install MySQL using a binary or source distribution on Unix, help table setup occurs when you run `mysql_install_db`. For an RPM distribution on Linux or binary distribution on Windows, help table setup occurs as part of the MySQL installation process.

If you upgrade MySQL using a binary distribution, the help tables are not upgraded automatically, but you can upgrade them manually. Locate the `fill_help_tables.sql` file in the `share` or `share/mysql` directory. Change location into that directory and process the file with the `mysql` client as follows:

```
shell> mysql -u root mysql < fill_help_tables.sql
```

You can also obtain the latest `fill_help_tables.sql` at any time to upgrade your help tables. Download the proper file for your version of MySQL from <http://dev.mysql.com/doc/index-other.html>. After downloading and uncompressing the file, process it with `mysql` as described previously.

If you are working with Bazaar and a MySQL development source tree, you will need to download the `fill_help_tables.sql` file because the tree contains only a “stub” version.

5.1.11. Server Response to Signals

On Unix, signals can be sent to processes. `mysqld` responds to signals sent to it as follows:

- `SIGTERM` causes the server to shut down.
- `SIGHUP` causes the server to reload the grant tables and flush the logs (like `FLUSH PRIVILEGES` and `FLUSH LOGS`). It also writes a status report to the error log that has this format:

```
Status information:

Current dir: /var/mysql/data/
Running threads: 0  Stack size: 196608
Current locks:

Key caches:
default
Buffer_size:          8388600
Block_size:           1024
Division_limit:       100
Age_limit:            300
blocks used:          0
not flushed:          0
w_requests:           0
writes:               0
r_requests:           0
reads:                0

handler status:
read_key:             0
read_next:            0
read_rnd              0
read_first:           1
write:                0
delete:               0
update:               0

Table status:
```

```

Opened tables:      5
Open tables:       0
Open files:        7
Open streams:      0

Alarm status:
Active alarms:     1
Max used alarms:   2
Next alarm time:  67

```

On some Mac OS X 10.3 versions, `mysqld` ignores `SIGHUP` and `SIGQUIT`.

5.1.12. The Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

This can occur initiated several ways. For example, a user with the `SHUTDOWN [697]` privilege can execute a `mysqladmin shutdown` command. `mysqladmin` can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to.

2. The server creates a shutdown thread if necessary.

Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

```
Error: Can't create thread to kill server
```

3. The server stops accepting new connections.

To prevent new activity from being initiated during shutdown, the server stops accepting new client connections by closing the handlers for the network interfaces to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

For each thread associated with a client connection, the server breaks the connection to the client and marks the thread as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see [Section 13.7.6.4, “KILL Syntax”](#), in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on `MyISAM` tables.

For threads that have an open transaction, the transaction is rolled back. Note that if a thread is updating a nontransactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated because the operation can terminate before completion.

If the server is a master replication server, it treats threads associated with currently connected slaves like other client threads. That is, each one is marked as killed and exits when it next checks its state.

If the server is a slave replication server, it stops the I/O and SQL threads, if they are active, before marking client threads as killed. The SQL thread is permitted to finish its current statement (to avoid

causing replication problems), and then stops. If the SQL thread is in the middle of a transaction at this point, the server waits until the current replication event group (if any) has finished executing, or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. See also [Section 13.4.2.6, “STOP SLAVE Syntax”](#).

If the slave is updating a non-transactional table when it is forcibly killed, the slave's data may become inconsistent with the master.

5. The server shuts down or closes storage engines.

At this stage, the server flushes the table cache and closes all open tables.

Each storage engine performs any actions necessary for tables that it manages. For example, `MyISAM` flushes any pending index writes for a table. `InnoDB` flushes its buffer pool to disk (unless `innodb_fast_shutdown` [1553] is 2), writes the current LSN to the tablespace, and terminates its own internal threads.

6. The server exits.

5.2. MySQL Server Logs

MySQL Server has several logs that can help you find out what activity is taking place.

Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping <code>mysqld</code>
General query log	Established client connections and statements received from clients
Binary log	Statements that change data (also used for replication)
Relay log	Data changes received from a replication master server
Slow query log	Queries that took more than <code>long_query_time</code> [517] seconds to execute

By default, no logs are enabled (except the error log on Windows). The following log-specific sections provide information about the server options that enable logging.

By default, the server writes files for all enabled logs in the data directory. You can force the server to close and reopen the log files (or in some cases switch to a new log file) by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement; execute `mysqladmin` with a `flush-logs` or `refresh` argument; or execute `mysqldump` with a `--flush-logs` [316] or `--master-data` [317] option. See [Section 13.7.6.3, “FLUSH Syntax”](#), [Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”](#), and [Section 4.5.4, “mysqldump — A Database Backup Program”](#). In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` [1903] system variable.

You can control the general query and slow query logs during runtime. You can enable or disable logging, or change the log file name. You can tell the server to write general query and slow query entries to log tables, log files, or both. For details, see [Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”](#), [Section 5.2.3, “The General Query Log”](#), and [Section 5.2.5, “The Slow Query Log”](#).

The relay log is used only on slave replication servers, to hold data changes from the master server that must also be made on the slave. For discussion of relay log contents and configuration, see [Section 16.2.2.1, “The Slave Relay Log”](#).

For information about log maintenance operations such as expiration of old log files, see [Section 5.2.6, “Server Log Maintenance”](#).

For information about keeping logs secure, see [Section 6.1.2.3, “Passwords and Logging”](#).

5.2.1. Selecting General Query and Slow Query Log Output Destinations

MySQL Server provides flexible control over the destination of output to the general query log and the slow query log, if those logs are enabled. Possible destinations for log entries are log files or the `general_log` and `slow_log` tables in the `mysql` database. Either or both destinations can be selected.

Currently, logging to tables incurs significantly more server overhead than logging to files. If you enable the general log or slow query log and require highest performance, you should use file logging, not table logging.

Log control at server startup. The `--log-output` [449] option specifies the destination for log output. This option does not in itself enable the logs. Its syntax is `--log-output[=value,...]` [449]:

- If `--log-output` [449] is given with a value, the value should be a comma-separated list of one or more of the words `TABLE` (log to tables), `FILE` (log to files), or `NONE` (do not log to tables or files). `NONE`, if present, takes precedence over any other specifiers.
- If `--log-output` [449] is omitted, the default logging destination is `FILE`.

The `general_log` [502] system variable controls logging to the general query log for the selected log destinations. If specified at server startup, `general_log` [502] takes an optional argument of 1 or 0 to enable or disable the log. To specify a file name other than the default for file logging, set the `general_log_file` [503] variable. Similarly, the `slow_query_log` [552] variable controls logging to the slow query log for the selected destinations and setting `slow_query_log_file` [553] specifies a file name for file logging. If either log is enabled, the server opens the corresponding log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected.

Examples:

- To write general query log entries to the log table and the log file, use `--log-output=TABLE,FILE` [449] to select both log destinations and `--general_log` [502] to enable the general query log.
- To write general and slow query log entries only to the log tables, use `--log-output=TABLE` [449] to select tables as the log destination and `--general_log` [502] and `--slow_query_log` [552] to enable both logs.
- To write slow query log entries only to the log file, use `--log-output=FILE` [449] to select files as the log destination and `--slow_query_log` [552] to enable the slow query log. (In this case, because the default log destination is `FILE`, you could omit the `--log-output` [449] option.)

Log control at runtime. The system variables associated with log tables and files enable runtime control over logging:

- The global `log_output` [515] system variable indicates the current logging destination. It can be modified at runtime to change the destination.
- The global `general_log` [502] and `slow_query_log` [552] variables indicate whether the general query log and slow query log are enabled (`ON`) or disabled (`OFF`). You can set these variables at runtime to control whether the logs are enabled.
- The global `general_log_file` [503] and `slow_query_log_file` [553] variables indicate the names of the general query log and slow query log files. You can set these variables at server startup or at runtime to change the names of the log files.
- To disable or enable general query logging for the current connection, set the session `sql_log_off` [556] variable to `ON` or `OFF`.

The use of tables for log output offers the following benefits:

- Log entries have a standard format. To display the current structure of the log tables, use these statements:

```
SHOW CREATE TABLE mysql.general_log;
SHOW CREATE TABLE mysql.slow_log;
```

- Log contents are accessible through SQL statements. This enables the use of queries that select only those log entries that satisfy specific criteria. For example, to select log contents associated with a particular client (which can be useful for identifying problematic queries from that client), it is easier to do this using a log table than a log file.
- Logs are accessible remotely through any client that can connect to the server and issue queries (if the client has the appropriate log table privileges). It is not necessary to log in to the server host and directly access the file system.

The log table implementation has the following characteristics:

- In general, the primary purpose of log tables is to provide an interface for users to observe the runtime execution of the server, not to interfere with its runtime execution.
- `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE` are valid operations on a log table. For `ALTER TABLE` and `DROP TABLE`, the log table cannot be in use and must be disabled, as described later.
- By default, the log tables use the `CSV` storage engine that writes data in comma-separated values format. For users who have access to the `.CSV` files that contain log table data, the files are easy to import into other programs such as spreadsheets that can process CSV input.

The log tables can be altered to use the `MyISAM` storage engine. You cannot use `ALTER TABLE` to alter a log table that is in use. The log must be disabled first. No engines other than `CSV` or `MyISAM` are legal for the log tables.

- To disable logging so that you can alter (or drop) a log table, you can use the following strategy. The example uses the general query log; the procedure for the slow query log is similar but uses the `slow_log` table and `slow_query_log` [552] system variable.

```
SET @old_log_state = @@global.general_log;
SET GLOBAL general_log = 'OFF';
ALTER TABLE mysql.general_log ENGINE = MyISAM;
SET GLOBAL general_log = @old_log_state;
```

- `TRUNCATE TABLE` is a valid operation on a log table. It can be used to expire log entries.
- `RENAME TABLE` is a valid operation on a log table. You can atomically rename a log table (to perform log rotation, for example) using the following strategy:

```
USE mysql;
DROP TABLE IF EXISTS general_log2;
CREATE TABLE general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

- As of MySQL 5.5.7, `CHECK TABLE` is a valid operation on a log table.
- `LOCK TABLES` cannot be used on a log table.
- `INSERT`, `DELETE`, and `UPDATE` cannot be used on a log table. These operations are permitted only internally to the server itself.
- `FLUSH TABLES WITH READ LOCK` and the state of the global `read_only` [544] system variable have no effect on log tables. The server can always write to the log tables.

- Entries written to the log tables are not written to the binary log and thus are not replicated to slave servers.
- To flush the log tables or log files, use `FLUSH TABLES` or `FLUSH LOGS`, respectively.
- Partitioning of log tables is not permitted.
- Before MySQL 5.5.25, `mysqldump` does not dump the `general_log` or `slow_query_log` tables for dumps of the `mysql` database. As of 5.5.25, the dump includes statements to recreate those tables so that they are not missing after reloading the dump file. Log table contents are not dumped.

5.2.2. The Error Log

The error log contains information indicating when `mysqld` was started and stopped and also any critical errors that occur while the server is running. If `mysqld` notices a table that needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if `mysqld` dies. The trace can be used to determine where `mysqld` died. See [MySQL Internals: Porting to Other Systems](#).

You can specify where `mysqld` writes the error log with the `--log-error[=file_name]` [448] option. If the option is given with no `file_name` value, `mysqld` uses the name `host_name.err` by default. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

If you do not specify `--log-error` [448], or (on Windows) if you use the `--console` [438] option, errors are written to `stderr`, the standard error output. Usually this is your terminal.

On Windows, error output is always written to the error log if `--console` [438] is not given.

In addition, on Windows, events and error messages are written to the Windows Event Log within the Application log. Entries marked as `Warning` and `Note` are written to the Event Log, but informational messages (such as information statements from individual storage engines) are not copied to the Event Log. The log entries have a source of **MySQL**. You cannot disable writing information to the Windows Event Log.

If you flush the logs using `FLUSH LOGS` or `mysqladmin flush-logs` and `mysqld` is writing the error log to a file (for example, if it was started with the `--log-error` [448] option), the effect is version dependent:

- As of MySQL 5.5.7, the server closes and reopens the log file. To rename the file, you can do so manually before flushing. Then flushing the logs reopens a new file with the original file name. For example, you can rename the file and create a new one using the following commands:

```
shell> mv host_name.err host_name.err-old
shell> mysqladmin flush-logs
shell> mv host_name.err-old backup-directory
```

On Windows, use `rename` rather than `mv`.

- Prior to MySQL 5.5.7, the server renames the current log file with the suffix `-old`, then creates a new empty log file. Be aware that a second log-flushing operation thus causes the original error log file to be lost unless you save it under a different name. On Windows, you cannot rename the error log while the server has it open before MySQL 5.5.7. To avoid a restart, flush the logs first to cause the server to rename the original file and create a new one, then save the renamed file. That also works on Unix, or you can use the commands shown earlier.

No error log renaming occurs when the logs are flushed in any case if the server is not writing to a named file.

If you use `mysqld_safe` to start `mysqld`, `mysqld_safe` arranges for `mysqld` to write error messages to a log file or to `syslog`. `mysqld_safe` has three error-logging options, `--syslog` [258], `--skip-syslog` [258], and `--log-error` [256]. The default with no logging options or with `--skip-syslog` [258] is to use the default log file. To explicitly specify use of an error log file, specify `--log-error=file_name` [256] to `mysqld_safe`, and `mysqld_safe` will arrange for `mysqld` to write messages to a log file. To use `syslog` instead, specify the `--syslog` [258] option.

If you specify `--log-error` [448] in an option file in a `[mysqld]`, `[server]`, or `[mysqld_safe]` section, `mysqld_safe` will find and use the option.

If `mysqld_safe` is used to start `mysqld` and `mysqld` dies unexpectedly, `mysqld_safe` notices that it needs to restart `mysqld` and writes a `restarted mysqld` message to the error log.

The `--log-warnings` [451] option or `log_warnings` [516] system variable can be used to control warning logging to the error log. The default value is enabled (1). Warning logging can be disabled using a value of 0. If the value is greater than 1, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See Section C.5.2.11, “Communication Errors and Aborted Connections”.

5.2.3. The General Query Log

The general query log is a general record of what `mysqld` is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to `mysqld`.

`mysqld` writes statements to the query log in the order that it receives them, which might differ from the order in which they are executed. This logging order contrasts to the binary log, for which statements are written after they are executed but before any locks are released. (Also, the query log contains all statements, whereas the binary log does not contain statements that only select data.)

By default, the general query log is disabled. To specify the initial general query log state explicitly, use `--general_log[={0|1}]` [502]. With no argument or an argument of 1, `--general_log` [502] enables the log. With an argument of 0, this option disables the log. To specify a log file name, use `--general_log_file=file_name` [503]. To specify the log destination, use `--log-output` [449] (as described in Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”). The older options to enable the general query log, `--log` [447] and `-l`, are deprecated.

If you specify no name for the general query log file, the default name is `host_name.log`. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the general query log or change the log file name at runtime, use the global `general_log` [502] and `general_log_file` [503] system variables. Set `general_log` [502] to 0 (or `OFF`) to disable the log or to 1 (or `ON`) to enable it. Set `general_log_file` [503] to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the general query log is enabled, the server writes output to any destinations specified by the `--log-output` [449] option or `log_output` [515] system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected. If the destination is `NONE`, the server writes no queries even if the general log is enabled. Setting the log file name has no effect on logging if the log destination value does not contain `FILE`.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). You can rename the file and create a new one by using the following commands:

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> mv host_name-old.log backup-directory
```

On Windows, use `rename` rather than `mv`.

You can also rename the general query log file at runtime by disabling the log:

```
SET GLOBAL general_log = 'OFF';
```

With the log disabled, rename the log file externally; for example, from the command line. Then enable the log again:

```
SET GLOBAL general_log = 'ON';
```

This method works on any platform and does not require a server restart.

The session `sql_log_off` [556] variable can be set to `ON` or `OFF` to disable or enable general query logging for the current connection.

The general query log should be protected because logged statements might contain passwords. See [Section 6.1.2.3, “Passwords and Logging”](#).

5.2.4. The Binary Log

The binary log contains “events” that describe database changes such as table creation operations or changes to table data. It also contains events for statements that potentially could have made changes (for example, a `DELETE` which matched no rows), unless row-based logging is used. The binary log also contains information about how long each statement took that updated data. The binary log has two important purposes:

- For replication, the binary log on a master replication server provides a record of the data changes to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See [Section 16.2, “Replication Implementation”](#).
- Certain data recovery operations require use of the binary log. After a backup has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See [Section 7.5, “Point-in-Time \(Incremental\) Recovery Using the Binary Log”](#).

The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. To log all statements (for example, to identify a problem query), use the general query log. See [Section 5.2.3, “The General Query Log”](#).

Running a server with binary logging enabled makes performance slightly slower. However, the benefits of the binary log in enabling you to set up replication and for restore operations generally outweigh this minor performance decrement.

The binary log should be protected because logged statements might contain passwords. See [Section 6.1.2.3, “Passwords and Logging”](#).

The following discussion describes some of the server options and variables that affect the operation of binary logging. For a complete list, see [Section 16.1.3.4, “Binary Log Options and Variables”](#).

To enable the binary log, start the server with the `--log-bin[=base_name] [1892]` option. If no *base_name* value is given, the default name is the value of the `pid-file` option (which by default is the name of host machine) followed by `-bin`. If the basename is given, the server writes the file in the data directory unless the basename is given with a leading absolute path name to specify a different directory. It

is recommended that you specify a basename explicitly rather than using the default of the host name; see [Section C.5.8, “Known Issues in MySQL”](#), for the reason.

If you supply an extension in the log name (for example, `--log-bin=base_name.extension` [1892]), the extension is silently removed and ignored.

`mysqld` appends a numeric extension to the binary log basename to generate binary log file names. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new file in the series each time it starts or flushes the logs. The server also creates a new binary log file automatically after the current log's size reaches `max_binlog_size` [1903]. A binary log file may become larger than `max_binlog_size` [1903] if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, `mysqld` also creates a binary log index file that contains the names of all used binary log files. By default, this has the same basename as the binary log file, with the extension `.index`. You can change the name of the binary log index file with the `--log-bin-index[=file_name]` [1892] option. You should not manually edit this file while `mysqld` is running; doing so would confuse `mysqld`.

The term “binary log file” generally denotes an individual numbered file containing database events. The term “binary log” collectively denotes the set of numbered binary log files plus the index file.

A client that has the `SUPER` [697] privilege can disable binary logging of its own statements by using a `SET sql_log_bin=0` statement. See [Section 5.1.4, “Server System Variables”](#).

The format of the events recorded in the binary log is dependent on the binary logging format. Three format types are supported, row-based logging, statement-based logging and mixed-base logging. The binary logging format used depends on the MySQL version. For general descriptions of the logging formats, see [Section 5.2.4.1, “Binary Logging Formats”](#). For detailed information about the format of the binary log, see [MySQL Internals: The Binary Log](#).

The server evaluates the `--binlog-do-db` [1894] and `--binlog-ignore-db` [1896] options in the same way as it does the `--replicate-do-db` [1875] and `--replicate-ignore-db` [1877] options. For information about how this is done, see [Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”](#).

If you are replicating from a MySQL Cluster to a standalone MySQL Server, you should be aware that the `NDB` storage engine uses default values for some binary logging options (including options specific to `NDB` such as `--ndb-log-update-as-write` [2373]) that differ from those used by other storage engines. If not corrected for, these differences can lead to divergence of the master's and slave's binary logs. For more information, see [Replication from NDB to other storage engines](#). In particular, if you are using a nontransactional storage engine such as `MyISAM` on the slave, see [Replication from NDB to a nontransactional storage engine](#).

A replication slave server by default does not write to its own binary log any data modifications that are received from the replication master. To log these modifications, start the slave with the `--log-slave-updates` [1870] option in addition to the `--log-bin` [1892] option (see [Section 16.1.3.3, “Replication Slave Options and Variables”](#)). This is done when a slave is also to act as a master to other slaves in chained replication.

You can delete all binary log files with the `RESET MASTER` statement, or a subset of them with `PURGE BINARY LOGS`. See [Section 13.7.6.6, “RESET Syntax”](#), and [Section 13.4.1.1, “PURGE BINARY LOGS Syntax”](#).

If you are using replication, you should not delete old binary log files on the master until you are sure that no slave still needs to use them. For example, if your slaves never run more than three days behind, once a day you can execute `mysqladmin flush-logs` on the master and then remove any logs that are

more than three days old. You can remove the files manually, but it is preferable to use `PURGE BINARY LOGS`, which also safely updates the binary log index file for you (and which can take a date argument). See [Section 13.4.1.1, “PURGE BINARY LOGS Syntax”](#).

You can display the contents of binary log files with the `mysqlbinlog` utility. This can be useful when you want to reprocess statements in the log for a recovery operation. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log_file | mysql -h server_name
```

`mysqlbinlog` also can be used to display replication slave relay log file contents because they are written using the same format as binary log files. For more information on the `mysqlbinlog` utility and how to use it, see [Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”](#). For more information about the binary log and recovery operations, see [Section 7.5, “Point-in-Time \(Incremental\) Recovery Using the Binary Log”](#).

Binary logging is done immediately after a statement or transaction completes but before any locks are released or any commit is done. This ensures that the log is logged in commit order.

Updates to nontransactional tables are stored in the binary log immediately after execution.

Within an uncommitted transaction, all updates (`UPDATE`, `DELETE`, or `INSERT`) that change transactional tables such as InnoDB tables are cached until a `COMMIT` statement is received by the server. At that point, `mysqld` writes the entire transaction to the binary log before the `COMMIT` is executed.

Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that the modifications to those tables are replicated.

When a thread that handles the transaction starts, it allocates a buffer of `binlog_cache_size` [1898] to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends.

The `Binlog_cache_use` [596] status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The `Binlog_cache_disk_use` [596] status variable shows how many of those transactions actually had to use a temporary file. These two variables can be used for tuning `binlog_cache_size` [1898] to a large enough value that avoids the use of temporary files.

The `max_binlog_cache_size` [1902] system variable (default 4GB, which is also the maximum) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this many bytes, it fails and rolls back. The minimum value is 4096.

If you are using the binary log and row based logging, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. If you are using statement-based logging, the original statement is written to the log.

The binary log format has some known limitations that can affect recovery from backups. See [Section 16.4.1, “Replication Features and Issues”](#).

Binary logging for stored programs is done as described in [Section 19.7, “Binary Logging of Stored Programs”](#).

Note that the binary log format differs in MySQL 5.5 from previous versions of MySQL, due to enhancements in replication. See [Section 16.4.2, “Replication Compatibility Between MySQL Versions”](#).

Writes to the binary log file and binary log index file are handled in the same way as writes to MyISAM tables. See [Section C.5.4.3, “How MySQL Handles a Full Disk”](#).

By default, the binary log is not synchronized to disk at each write. So if the operating system or machine (not only the MySQL server) crashes, there is a chance that the last statements of the binary log are lost. To prevent this, you can make the binary log be synchronized to disk after every *N* writes to the binary log, with the `sync_binlog` [1904] system variable. See [Section 5.1.4, “Server System Variables”](#). 1 is the safest value for `sync_binlog` [1904], but also the slowest. Even with `sync_binlog` [1904] set to 1, there is still the chance of an inconsistency between the table content and binary log content in case of a crash. For example, if you are using InnoDB tables and the MySQL server processes a `COMMIT` statement, it writes the whole transaction to the binary log and then commits this transaction into InnoDB. If the server crashes between those two operations, the transaction is rolled back by InnoDB at restart but still exists in the binary log. To resolve this, you should set `--innodb_support_xa` [1572] to 1. Although this option is related to the support of XA transactions in InnoDB, it also ensures that the binary log and InnoDB data files are synchronized.

For this option to provide a greater degree of safety, the MySQL server should also be configured to synchronize the binary log and the InnoDB logs to disk at every transaction. The InnoDB logs are synchronized by default, and `sync_binlog=1` can be used to synchronize the binary log. The effect of this option is that at restart after a crash, after doing a rollback of transactions, the MySQL server cuts rolled back InnoDB transactions from the binary log. This ensures that the binary log reflects the exact data of InnoDB tables, and so, that the slave remains in synchrony with the master (not receiving a statement which has been rolled back).

If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed InnoDB transaction. This should not happen if `sync_binlog=1` and the disk/file system do an actual sync when they are requested to (some do not), so the server prints an error message `The binary log file_name is shorter than its expected size`. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the master's data.

The session values of the following system variables are written to the binary log and honored by the replication slave when parsing the binary log:

- `sql_mode` [556] (except that the `NO_DIR_IN_CREATE` [614] mode is not replicated; see [Section 16.4.1.35, “Replication and Variables”](#))
- `foreign_key_checks` [500]
- `unique_checks` [571]
- `character_set_client` [486]
- `collation_connection` [489]
- `collation_database` [489]
- `collation_server` [489]
- `sql_auto_is_null` [554]

5.2.4.1. Binary Logging Formats

The server uses several logging formats to record information in the binary log. The exact format employed depends on the version of MySQL being used. There are three logging formats:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called *statement-based logging*. You can cause this format to be used by starting the server with `--binlog-format=STATEMENT` [436].

- In *row-based logging*, the master writes events to the binary log that indicate how individual table rows are affected. You can cause the server to use row-based logging by starting it with `--binlog-format=ROW` [436].
- A third option is also available: *mixed logging*. With mixed logging, statement-based logging is used by default, but the logging mode switches automatically to row-based in certain cases as described below. You can cause MySQL to use mixed logging explicitly by starting `mysqld` with the option `--binlog-format=MIXED` [436].

In MySQL 5.5, the default binary logging format is `STATEMENT`.

The logging format can also be set or limited by the storage engine being used. This helps to eliminate issues when replicating certain statements between a master and slave which are using different storage engines.

With statement-based replication, there may be issues with replicating nondeterministic statements. In deciding whether or not a given statement is safe for statement-based replication, MySQL determines whether it can guarantee that the statement can be replicated using statement-based logging. If MySQL cannot make this guarantee, it marks the statement as potentially unreliable and issues the warning, `Statement may not be safe to log in statement format`.

You can avoid these issues by using MySQL's row-based replication instead.

5.2.4.2. Setting The Binary Log Format

You can select the binary logging format explicitly by starting the MySQL server with `--binlog-format=type` [436]. The supported values for `type` are:

- `STATEMENT` causes logging to be statement based.
- `ROW` causes logging to be row based.
- `MIXED` causes logging to use mixed format.

In MySQL 5.5, the default binary logging format is `STATEMENT`. This includes MySQL Cluster NDB 7.2.1 and later MySQL Cluster NDB 7.2 releases, which are based on MySQL 5.5.

The logging format also can be switched at runtime. To specify the format globally for all clients, set the global value of the `binlog_format` [1899] system variable:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 'MIXED';
```

An individual client can control the logging format for its own statements by setting the session value of `binlog_format` [1899]:

```
mysql> SET SESSION binlog_format = 'STATEMENT';
mysql> SET SESSION binlog_format = 'ROW';
mysql> SET SESSION binlog_format = 'MIXED';
```



Note

Each MySQL Server can set its own and only its own binary logging format (true whether `binlog_format` [1899] is set with global or session scope). This means that changing the logging format on a replication master does not cause a slave to change its logging format to match. (When using `STATEMENT` mode, the `binlog_format` [1899] system variable is not replicated; when using `MIXED` or `ROW` logging mode, it is replicated but is ignored by the slave.) Changing the binary logging format on the master while replication is ongoing, or without also changing

it on the slave can thus cause unexpected results, or even cause replication to fail altogether.

To change the global or session `binlog_format` [1899] value, you must have the `SUPER` [697] privilege.

In addition to switching the logging format manually, a slave server may switch the format *automatically*. This happens when the server is running in either `STATEMENT` or `MIXED` format and encounters an event in the binary log that is written in `ROW` logging format. In that case, the slave switches to row-based replication temporarily for that event, and switches back to the previous format afterward.

There are several reasons why a client might want to set binary logging on a per-session basis:

- A session that makes many small changes to the database might want to use row-based logging.
- A session that performs updates that match many rows in the `WHERE` clause might want to use statement-based logging because it will be more efficient to log a few statements than many rows.
- Some statements require a lot of execution time on the master, but result in just a few rows being modified. It might therefore be beneficial to replicate them using row-based logging.

There are exceptions when you cannot switch the replication format at runtime:

- From within a stored function or a trigger
- If the `NDBCLUSTER` storage engine is enabled
- If the session is currently in row-based replication mode and has open temporary tables

Trying to switch the format in any of these cases results in an error.

Switching the replication format at runtime is not recommended when any temporary tables exist, because temporary tables are logged only when using statement-based replication, whereas with row-based replication they are not logged. With mixed replication, temporary tables are usually logged; exceptions happen with user-defined functions (UDFs) and with the `UUID()` [1201] function.

With the binary log format set to `ROW`, many changes are written to the binary log using the row-based format. Some changes, however, still use the statement-based format. Examples include all DDL (data definition language) statements such as `CREATE TABLE`, `ALTER TABLE`, or `DROP TABLE`.

The `--binlog-row-event-max-size` [1891] option is available for servers that are capable of row-based replication. Rows are stored into the binary log in chunks having a size in bytes not exceeding the value of this option. The value must be a multiple of 256. The default value is 1024.



Warning

When using *statement-based logging* for replication, it is possible for the data on the master and slave to become different if a statement is designed in such a way that the data modification is *nondeterministic*; that is, it is left to the will of the query optimizer. In general, this is not a good practice even outside of replication. For a detailed explanation of this issue, see [Section C.5.8, “Known Issues in MySQL”](#).

5.2.4.3. Mixed Binary Logging Format

When running in `MIXED` logging format, the server automatically switches from statement-based to row-based logging under the following conditions:

- When a DML statement updates an `NDBCLUSTER` table.
- When a function contains `UUID()` [1201].

- When one or more tables with `AUTO_INCREMENT` columns are updated and a trigger or stored function is invoked. Like all other unsafe statements, this generates a warning if `binlog_format = STATEMENT` [1899].

For more information, see [Section 16.4.1.1, “Replication and `AUTO_INCREMENT`”](#).

- When any `INSERT DELAYED` is executed.
- When the body of a view requires row-based replication, the statement creating the view also uses it. For example, this occurs when the statement creating a view uses the `UUID()` [1201] function.
- When a call to a UDF is involved.
- If a statement is logged by row and the session that executed the statement has any temporary tables, logging by row is used for all subsequent statements (except for those accessing temporary tables) until all temporary tables in use by that session are dropped.

This is true whether or not any temporary tables are actually logged.

Temporary tables cannot be logged using row-based format; thus, once row-based logging is used, all subsequent statements using that table are unsafe. The server approximates this condition by treating all statements executed during the session as unsafe until the session no longer holds any temporary tables.

- When `FOUND_ROWS()` [1193] or `ROW_COUNT()` [1196] is used. (Bug #12092, Bug #30244)
- When `USER()` [1197], `CURRENT_USER()` [1192], or `CURRENT_USER` [1192] is used. (Bug #28086)
- When a statement refers to one or more system variables. (Bug #31168)

Exception. The following system variables, when used with session scope (only), do not cause the logging format to switch:

- `auto_increment_increment` [1865]
- `auto_increment_offset` [1868]
- `character_set_client` [486]
- `character_set_connection` [487]
- `character_set_database` [487]
- `character_set_server` [488]
- `collation_connection` [489]
- `collation_database` [489]
- `collation_server` [489]
- `foreign_key_checks` [500]
- `identity` [505]
- `last_insert_id` [511]
- `lc_time_names` [512]

- `pseudo_thread_id` [539]
- `sql_auto_is_null` [554]
- `time_zone` [567]
- `timestamp` [567]
- `unique_checks` [571]

For information about determining system variable scope, see [Section 5.1.5, “Using System Variables”](#).

For information about how replication treats `sql_mode` [556], see [Section 16.4.1.35, “Replication and Variables”](#).

- When one of the tables involved is a log table in the `mysql` database.
- When the `LOAD_FILE()` [1105] function is used. (Bug #39701)



Note

A warning is generated if you try to execute a statement using statement-based logging that should be written using row-based logging. The warning is shown both in the client (in the output of `SHOW WARNINGS`) and through the `mysqld` error log. A warning is added to the `SHOW WARNINGS` table each time such a statement is executed. However, only the first statement that generated the warning for each client session is written to the error log to prevent flooding the log.

In addition to the decisions above, individual engines can also determine the logging format used when information in a table is updated. The logging capabilities of an individual engine can be defined as follows:

- If an engine supports row-based logging, the engine is said to be *row-logging capable*.
- If an engine supports statement-based logging, the engine is said to be *statement-logging capable*.

A given storage engine can support either or both logging formats. The following table lists the formats supported by each engine.

Storage Engine	Row Logging Supported	Statement Logging Supported
<code>ARCHIVE</code>	Yes	Yes
<code>BLACKHOLE</code>	Yes	Yes
<code>CSV</code>	Yes	Yes
<code>EXAMPLE</code>	Yes	No
<code>FEDERATED</code>	Yes	Yes
<code>HEAP</code>	Yes	Yes
<code>InnoDB</code>	Yes	Yes when the transaction isolation level is <code>REPEATABLE READ</code> [1401] or <code>SERIALIZABLE</code> [1402]; No otherwise.
<code>MyISAM</code>	Yes	Yes

Storage Engine	Row Logging Supported	Statement Logging Supported
MERGE	Yes	Yes
NDBCLUSTER	Yes	No

In MySQL 5.5.3 and later, whether a statement is to be logged and the logging mode to be used is determined according to the type of statement (safe, unsafe, or binary injected), the binary logging format (STATEMENT, ROW, or MIXED), and the logging capabilities of the storage engine (statement capable, row capable, both, or neither). Statements may be logged with or without a warning; failed statements are not logged, but generate errors in the log. This is shown in the following decision table, where **SLC** stands for “statement-logging capable” and **RLC** stands for “row-logging capable”.

Condition				Action	
Type	binlog_format [1899]	SLC	RLC	Error / Warning	Logged as
*	*	No	No	Error: Cannot execute statement: Binary logging is impossible since at least one engine is involved that is both row-incapable and statement-incapable.	-
Safe	STATEMENT	Yes	No	-	STATEMENT
Safe	MIXED	Yes	No	-	STATEMENT
Safe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that is not capable of row-based logging.	-
Unsafe	STATEMENT	Yes	No	Warning: Unsafe statement binlogged in statement format, since BINLOG_FORMAT = STATEMENT	STATEMENT
Unsafe	MIXED	Yes	No	Error: Cannot execute statement: Binary logging of an unsafe statement is impossible	-

Condition				Action	
Type	<code>binlog_format</code> [1899]	SLC	RLC	Error / Warning	Logged as
				when the storage engine is limited to statement-based logging, even if <code>BINLOG_FORMAT = MIXED</code> .	
Unsafe	<code>ROW</code>	Yes	No	<code>Error: Cannot execute statement: Binary logging is impossible since <code>BINLOG_FORMAT = ROW</code> and at least one table uses a storage engine that is not capable of row-based logging.</code>	-
Row Injection	<code>STATEMENT</code>	Yes	No	<code>Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.</code>	-
Row Injection	<code>MIXED</code>	Yes	No	<code>Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.</code>	-
Row Injection	<code>ROW</code>	Yes	No	<code>Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.</code>	-
Safe	<code>STATEMENT</code>	No	Yes	<code>Error: Cannot execute</code>	-

Condition				Action	
Type	binlog_format [1899]	SLC	RLC	Error / Warning	Logged as
				statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine that is not capable of statement-based logging.	
Safe	MIXED	No	Yes	-	ROW
Safe	ROW	No	Yes	-	ROW
Unsafe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine that is not capable of statement-based logging.	-
Unsafe	MIXED	No	Yes	-	ROW
Unsafe	ROW	No	Yes	-	ROW
Row Injection	STATEMENT	No	Yes	Error: Cannot execute row injection: Binary logging is not possible since BINLOG_FORMAT = STATEMENT.	-
Row Injection	MIXED	No	Yes	-	ROW
Row Injection	ROW	No	Yes	-	ROW
Safe	STATEMENT	Yes	Yes	-	STATEMENT
Safe	MIXED	Yes	Yes	-	ROW
Safe	ROW	Yes	Yes	-	ROW
Unsafe	STATEMENT	Yes	Yes	Warning: Unsafe statement binlogged in statement format since	STATEMENT

Condition				Action	
Type	<code>binlog_format</code> [1899]	SLC	RLC	Error / Warning	Logged as
				<code>BINLOG_FORMAT = STATEMENT.</code>	
Unsafe	MIXED	Yes	Yes	-	ROW
Unsafe	ROW	Yes	Yes	-	ROW
Row Injection	STATEMENT	Yes	Yes	Error: Cannot execute row injection: Binary logging is not possible because <code>BINLOG_FORMAT = STATEMENT.</code>	-
Row Injection	MIXED	Yes	Yes	-	ROW
Row Injection	ROW	Yes	Yes	-	ROW

Handling of mixed-format logging in MySQL 5.5.2 and earlier. The decision-making process for binary logging changed in MySQL 5.5.3, due to the fix for Bug #39934. Prior to MySQL 5.5.3, when determining the logging mode to be used, the capabilities of all the tables affected by the event are combined, and the set of affected tables is then marked according to these rules:

- A set of tables is defined as *row-logging restricted* if the tables are row-logging capable but not statement-logging capable.
- A set of tables is defined as *statement-logging restricted* if the tables are statement-logging capable but not row-logging capable.

Once the determination of the possible logging formats required by the statement is complete it is compared to the current `binlog_format` [1899] setting. The following table is used in MySQL 5.5.2 and earlier to decide how the information is recorded in the binary log or, if appropriate, whether an error is raised. In the table, a safe operation is defined as one that is deterministic.

In MySQL 5.5.2 and earlier, several rules decide whether the statement is deterministic, as shown in the following table, where **SLR** stands for "statement-logging restricted" and **RLR** stands for "row-logging restricted". A statement is *statement-logging restricted* if one or more of the tables it accesses is not row-logging capable. Similarly, a statement is *row-logging restricted* if any table accessed by the statement is not statement-logging capable.

Condition				Action	
Safe/unsafe	<code>binlog_format</code> [1899]	SLR	RLR	Error/Warning	Logged as
Safe	STATEMENT	Yes	Yes	Error: not loggable	
Safe	STATEMENT	Yes	No		STATEMENT
Safe	STATEMENT	No	Yes	Error: not loggable	
Safe	STATEMENT	No	No		STATEMENT

Condition				Action	
Safe/ unsafe	binlog_format [1899]	SLR	RLR	Error/Warning	Logged as
Safe	MIXED	Yes	Yes	Error: not loggable	
Safe	MIXED	Yes	No		STATEMENT
Safe	MIXED	No	Yes		ROW
Safe	MIXED	No	No		STATEMENT
Safe	ROW	Yes	Yes	Error: not loggable	
Safe	ROW	Yes	No	Error: not loggable	
Safe	ROW	No	Yes		ROW
Safe	ROW	No	No		ROW
Unsafe	STATEMENT	Yes	Yes	Error: not loggable	
Unsafe	STATEMENT	Yes	No	Warning: unsafe	STATEMENT
Unsafe	STATEMENT	No	Yes	Error: not loggable	
Unsafe	STATEMENT	No	No	Warning: unsafe	STATEMENT
Unsafe	MIXED	Yes	Yes	Error: not loggable	
Unsafe	MIXED	Yes	No	Error: not loggable	
Unsafe	MIXED	No	Yes		ROW
Unsafe	MIXED	No	No		ROW
Unsafe	ROW	Yes	Yes	Error: not loggable	
Unsafe	ROW	Yes	No	Error: not loggable	
Unsafe	ROW	No	Yes		ROW
Unsafe	ROW	No	No		ROW

In all MySQL 5.5 releases, when a warning is produced by the determination, a standard MySQL warning is produced (and is available using `SHOW WARNINGS`). The information is also written to the `mysqld` error log. Only one error for each error instance per client connection is logged to prevent flooding the log. The log message includes the SQL statement that was attempted.

If a slave server was started with `--log-warnings [451]` enabled, the slave prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth.

5.2.4.4. Logging Format for Changes to `mysql` Database Tables

The contents of the grant tables in the `mysql` database can be modified directly (for example, with `INSERT` or `DELETE`) or indirectly (for example, with `GRANT` or `CREATE USER`). Statements that affect `mysql` database tables are written to the binary log using the following rules:

- Data manipulation statements that change data in `mysql` database tables directly are logged according to the setting of the `binlog_format` [1899] system variable. This pertains to statements such as `INSERT`, `UPDATE`, `DELETE`, `REPLACE`, `DO`, `LOAD DATA INFILE`, `SELECT`, and `TRUNCATE TABLE`.
- Statements that change the `mysql` database indirectly are logged as statements regardless of the value of `binlog_format` [1899]. This pertains to statements such as `GRANT`, `REVOKE`, `SET PASSWORD`, `RENAME USER`, `CREATE` (all forms except `CREATE TABLE ... SELECT`), `ALTER` (all forms), and `DROP` (all forms).

`CREATE TABLE ... SELECT` is a combination of data definition and data manipulation. The `CREATE TABLE` part is logged using statement format and the `SELECT` part is logged according to the value of `binlog_format` [1899].

5.2.5. The Slow Query Log

The slow query log consists of SQL statements that took more than `long_query_time` [517] seconds to execute and required at least `min_examined_row_limit` [527] rows to be examined. The minimum and default values of `long_query_time` [517] are 0 and 10, respectively. The value can be specified to a resolution of microseconds. For logging to a file, times are written including the microseconds part. For logging to tables, only integer times are written; the microseconds part is ignored.

The time to acquire the initial table locks is not counted as execution time. `mysqld` writes a statement to the slow query log after it has been executed and after all locks have been released, so log order might differ from execution order.

By default, the slow query log is disabled. To specify the initial slow query log state explicitly, use `--slow_query_log[={0|1}]` [552]. With no argument or an argument of 1, `--slow_query_log` [552] enables the log. With an argument of 0, this option disables the log. To specify a log file name, use `--slow_query_log_file=file_name` [553]. To specify the log destination, use `--log-output` [449] (as described in Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”). The older option to enable the slow query log file, `--log-slow-queries` [450], is deprecated.

If you specify no name for the slow query log file, the default name is `host_name-slow.log`. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the slow query log or change the log file name at runtime, use the global `slow_query_log` [552] and `slow_query_log_file` [553] system variables. Set `slow_query_log` [552] to 0 (or `OFF`) to disable the log or to 1 (or `ON`) to enable it. Set `slow_query_log_file` [553] to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the slow query log is enabled, the server writes output to any destinations specified by the `--log-output` [449] option or `log_output` [515] system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected. If the destination is `NONE`, the server writes no queries even if the slow query log is enabled. Setting the log file name has no effect on logging if the log destination value does not contain `FILE`.

The server writes less information to the slow query log (and binary log) if you use the `--log-short-format` [450] option.

To include slow administrative statements such as `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `ALTER TABLE` in the statements written to the slow query log, use the `--log-slow-admin-statements` [450] server option.

To include queries that do not use indexes for row lookups in the statements written to the slow query log, enable the `log_queries_not_using_indexes` [515] system variable. When such queries are logged, the slow query log may grow quickly.

The server uses the controlling parameters in the following order to determine whether to write a query to the slow query log:

1. The query must either not be an administrative statement, or `--log-slow-admin-statements` [450] must have been specified.
2. The query must have taken at least `long_query_time` [517] seconds, or `log_queries_not_using_indexes` [515] must be enabled and the query used no indexes for row lookups.
3. The query must have examined at least `min_examined_row_limit` [527] rows.

The server does not write queries handled by the query cache to the slow query log, nor queries that would not benefit from the presence of an index because the table has zero rows or one row.

By default, a replication slave does not write replicated queries to the slow query log. To change this, use the `--log-slow-slave-statements` [1871] server option.

The slow query log should be protected because logged statements might contain passwords. See [Section 6.1.2.3, “Passwords and Logging”](#).

The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can become a difficult task. To make this easier, you can process a slow query log file using the `mysqldumpslow` command to summarize the queries that appear in the log. See [Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”](#).

5.2.6. Server Log Maintenance

As described in [Section 5.2, “MySQL Server Logs”](#), MySQL Server can create several different log files to help you see what activity is taking place. However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See [Section 7.2, “Database Backup Methods”](#).

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for this. If you installed MySQL from an RPM distribution, this script should have been installed automatically. Be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all slaves.

On other systems, you must install a short script yourself that you start from `cron` (or its equivalent) for handling log files.

For the binary log, you can set the `expire_logs_days` [498] system variable to expire binary log files automatically after a given number of days (see [Section 5.1.4, “Server System Variables”](#)). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master. To remove binary logs on demand, use the `PURGE BINARY LOGS` statement (see [Section 13.4.1.1, “PURGE BINARY LOGS Syntax”](#)).

You can force MySQL to start using new log files by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement or execute a `mysqladmin flush-logs`, `mysqladmin refresh`, `mysqldump --flush-logs`, or `mysqldump --master-data` command. See [Section 13.7.6.3, “FLUSH Syntax”](#), [Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”](#), and [Section 4.5.4, “mysqldump — A Database Backup Program”](#). In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size [1903]` system variable.

As of MySQL 5.5.3, `FLUSH LOGS` supports optional modifiers to enable selective flushing of individual logs (for example, `FLUSH BINARY LOGS`).

A log-flushing operation does the following:

- If general query logging or slow query logging to a log file is enabled, the server closes and reopens the general query log file or slow query log file.
- If binary logging is enabled, the server closes the current binary log file and opens a new log file with the next sequence number.
- If the server was started with the `--log-error [448]` option to cause the error log to be written to a file, the result of a log-flushing operation is version dependent:
 - As of MySQL 5.5.7, the server closes and reopens the log file.
 - Prior to MySQL 5.5.7, the server renames the current log file with the suffix `-old`, then creates a new empty log file.

The server creates a new binary log file when you flush the logs. However, it just closes and reopens the general and slow query log files. To cause new files to be created on Unix, rename the current log files before flushing them. At flush time, the server opens new log files with the original names. For example, if the general and slow query log files are named `mysql.log` and `mysql-slow.log`, you can use a series of commands like this:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mv mysql-slow.log mysql-slow.old
shell> mysqladmin flush-logs
```

On Windows, use `rename` rather than `mv`.

At this point, you can make a backup of `mysql.old` and `mysql-slow.old` and then remove them from disk.

A similar strategy can be used to back up the error log file, if there is one, except that, on Windows, you cannot rename the error log file while the server has it open before MySQL 5.5.7. To rename the error log file, a stop and restart can be avoided by flushing the logs to cause the server to rename the current log file with the suffix `-old` and create a new empty error log file. For further information, see [Section 5.2.2, “The Error Log”](#).

You can rename the general query log or slow query log at runtime by disabling the log:

```
SET GLOBAL general_log = 'OFF';
SET GLOBAL slow_query_log = 'OFF';
```

With the logs disabled, rename the log files externally; for example, from the command line. Then enable the logs again:

```
SET GLOBAL general_log = 'ON';
SET GLOBAL slow_query_log = 'ON';
```

This method works on any platform and does not require a server restart.

5.3. Running Multiple MySQL Instances on One Machine

In some cases, you might want to run multiple instances of MySQL on a single machine. You might want to test a new MySQL release while leaving an existing production setup undisturbed. Or you might want to give different users access to different `mysqld` servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

It is possible to use a different MySQL server binary per instance, or use the same binary for multiple instances, or any combination of the two approaches. For example, you might run a server from MySQL 5.1 and one from MySQL 5.5, to see how different versions handle a given workload. Or you might run multiple instances of the current production version, each managing a different set of databases.

Whether or not you use distinct server binaries, each instance that you run must be configured with unique values for several operating parameters. This eliminates the potential for conflict between instances. Parameters can be set on the command line, in option files, or by setting environment variables. See [Section 4.2.3, “Specifying Program Options”](#). To see the values used by a given instance, connect to it and execute a `SHOW VARIABLES` statement.

The primary resource managed by a MySQL instance is the data directory. Each instance should use a different data directory, the location of which is specified using the `--datadir=path` [\[439\]](#) option. For methods of configuring each instance with its own data directory, and warnings about the dangers of failing to do so, see [Section 5.3.1, “Setting Up Multiple Data Directories”](#).

In addition to using different data directories, several other options must have different values for each server instance:

- `--port=port_num` [\[458\]](#)

`--port` [\[458\]](#) controls the port number for TCP/IP connections. Alternatively, if the host has multiple network addresses, you can use `--bind-address` [\[435\]](#) to cause each server to listen to a different address.

- `--socket=path` [\[464\]](#)

`--socket` [\[464\]](#) controls the Unix socket file path on Unix or the named pipe name on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers configured to permit named-pipe connections.

- `--shared-memory-base-name=name` [\[460\]](#)

This option is used only on Windows. It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory. It is necessary to specify distinct shared-memory names only for those servers configured to permit shared-memory connections.

- `--pid-file=file_name` [\[456\]](#)

This option indicates the path name of the file in which the server writes its process ID.

If you use the following log file options, their values must differ for each server:

- `--general_log_file=file_name` [\[503\]](#)

- `--log-bin[=file_name]` [\[1892\]](#)

- `--slow_query_log_file=file_name` [553]
- `--log-error[=file_name]` [448]

For further discussion of log file options, see [Section 5.2, “MySQL Server Logs”](#).

To achieve better performance, you can specify the following option differently for each server, to spread the load between several physical disks:

- `--tmpdir=path` [466]

Having different temporary directories also makes it easier to determine which MySQL server created any given temporary file.

If you have multiple MySQL installations in different locations, you can specify the base directory for each installation with the `--basedir=path` [435] option. This causes each instance to automatically use a different data directory, log files, and PID file because the default for each of those parameters is relative to the base directory. In that case, the only other options you need to specify are the `--socket` [464] and `--port` [458] options. Suppose that you install different versions of MySQL using `tar` file binary distributions. These install in different locations, so you can start the server for each installation using the command `bin/mysqld_safe` under its corresponding base directory. `mysqld_safe` determines the proper `--basedir` [435] option to pass to `mysqld`, and you need specify only the `--socket` [258] and `--port` [257] options to `mysqld_safe`.

As discussed in the following sections, it is possible to start additional servers by specifying appropriate command options or by setting environment variables. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The `--defaults-file` [248] option is useful for this purpose.

5.3.1. Setting Up Multiple Data Directories

Each MySQL Instance on a machine should have its own data directory. The location is specified using the `--datadir=path` [439] option.

There are different methods of setting up a data directory for a new instance:

- Create a new data directory.
- Copy an existing data directory.

The following discussion provides more detail about each method.



Warning

Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log file names that are unique to each server. Otherwise, the servers try to log to the same files.

Even when the preceding precautions are observed, this kind of setup works only with `MyISAM` and `MERGE` tables, and not with any of the other storage engines. Also, this warning against sharing a data directory among servers always applies in an NFS environment. Permitting multiple MySQL servers to access a common data directory over NFS is a *very bad idea*. The primary problem is that NFS is the speed bottleneck. It is not meant for such use. Another risk with NFS is that you must

devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the `lockd` daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Create a New Data Directory

With this method, the data directory will be in the same state as when you first install MySQL. It will have the default set of MySQL accounts and no user data.

On Unix, initialize the data directory by running `mysql_install_db`. See [Section 2.11.1, “Unix Postinstallation Procedures”](#).

On Windows, the data directory is included in the MySQL distribution:

- MySQL Zip archive distributions for Windows contain an unmodified data directory. You can unpack such a distribution into a temporary location, then copy it `data` directory to where you are setting up the new instance.
- As of MySQL 5.5.5, Windows MSI package installers create and set up the data directory that the installed server will use, but also create a pristine “template” data directory named `data` under the installation directory. After an installation has been performed using an MSI package, the template data directory can be copied to set up additional MySQL instances.

Copy an Existing Data Directory

With this method, any MySQL accounts or user data present in the data directory are carried over to the new data directory.

1. Stop the existing MySQL instance using the data directory. This must be a clean shutdown so that the instance flushes any pending changes to disk.
2. Copy the data directory to the location where the new data directory should be.
3. Copy the `my.cnf` or `my.ini` option file used by the existing instance. This serves as a basis for the new instance.
4. Modify the new option file so that any pathnames referring to the original data directory refer to the new data directory. Also, modify any other options that must be unique per instance, such as the TCP/IP port number and the log files. For a list of parameters that must be unique per instance, see [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#).
5. Start the new instance, telling it to use the new option file.

5.3.2. Running Multiple MySQL Instances on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters, or by installing several servers as Windows services and running them that way. General instructions for running MySQL from the command line or as a service are given in [Section 2.3, “Installing MySQL on Microsoft Windows”](#). The following sections describe how to start each server with different values for those options that must be unique per server, such as the data directory. These options are listed in [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#).

5.3.2.1. Starting Multiple MySQL Instances at the Windows Command Line

The procedure for starting a single MySQL server manually from the command line is described in [Section 2.3.7.5, “Starting MySQL from the Windows Command Line”](#). To start multiple servers this way,

you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the file name with a `--defaults-file` [248] option when you run it.

Suppose that you want to run `mysqld` on port 3307 with a data directory of `C:\mydata1`, and `mysqld-debug` on port 3308 with a data directory of `C:\mydata2`. Use this procedure:

1. Make sure that each data directory exists, including its own copy of the `mysql` database that contains the grant tables.
2. Create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

Create a second file named `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

3. Use the `--defaults-file` [248] option to start each server with its own option file:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-debug --defaults-file=C:\my-opts2.cnf
```

Each server starts in the foreground (no new prompt appears until the server exits later), so you will need to issue those two commands in separate console windows.

To shut down the servers, connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 shutdown
```

Servers configured as just described permit clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to permit named-pipe connections, use the `mysqld` or `mysqld-debug` server and specify options that enable the named pipe and specify its name. Each server that supports named-pipe connections must use a unique pipe name. For example, the `C:\my-opts1.cnf` file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

Modify `C:\my-opts2.cnf` similarly for use by the second server. Then start the servers as described previously.

A similar procedure applies for servers that you want to permit shared-memory connections. Enable such connections with the `--shared-memory` [460] option and specify a unique shared-memory name for each server with the `--shared-memory-base-name` [460] option.

5.3.2.2. Starting Multiple MySQL Instances as Windows Services

On Windows, a MySQL server can run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in [Section 2.3.7.7, "Starting MySQL as a Windows Service"](#).

To set up multiple MySQL services, you must make sure that each instance uses a different service name in addition to the other parameters that must be unique per instance.

For the following instructions, suppose that you want to run the `mysqld` server from two different versions of MySQL that are installed at `C:\mysql-5.1.55` and `C:\mysql-5.5.31`, respectively. (This might be the case if you are running 5.1.55 as your production server, but also want to conduct tests using 5.5.31.)

To install MySQL as a Windows service, use the `--install` or `--install-manual` option. For information about these options, see [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, shut down and remove any existing MySQL services.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 5.1.55 `mysqld` using the service name of `mysqld1` and the 5.5.31 `mysqld` using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 5.1.55 and the `[mysqld2]` group for 5.5.31. For example, you can set up `C:\my.cnf` like this:

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-5.1.55
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-5.5.31
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows, using the full server path names to ensure that Windows registers the correct executable program for each service:

```
C:\> C:\mysql-5.1.55\bin\mysqld --install mysqld1
C:\> C:\mysql-5.5.31\bin\mysqld --install mysqld2
```

To start the services, use the services manager, or use `NET START` with the appropriate service names:

```
C:\> NET START mysqld1
C:\> NET START mysqld2
```

To stop the services, use the services manager, or use `NET STOP` with the appropriate service names:

```
C:\> NET STOP mysqld1
C:\> NET STOP mysqld2
```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` [\[248\]](#) when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

With this approach, to specify options for the 5.1.55 `mysqld`, create a file `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
basedir = C:/mysql-5.1.55
port = 3307
enable-named-pipe
```

```
socket = mypipe1
```

For the 5.5.31 `mysqld`, create a file `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
basedir = C:/mysql-5.5.31
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows (enter each command on a single line):

```
C:\> C:\mysql-5.1.55\bin\mysqld --install mysqld1
      --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-5.5.31\bin\mysqld --install mysqld2
      --defaults-file=C:\my-opts2.cnf
```

When you install a MySQL server as a service and use a `--defaults-file` [248] option, the service name must precede the option.

After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use `mysqld --remove` for each one, specifying a service name following the `--remove` [458] option. If the service name is the default (`MySQL`), you can omit it.

5.3.3. Running Multiple MySQL Instances on Unix

One way is to run multiple MySQL instances on Unix is to compile different servers with different default TCP/IP ports and Unix socket files so that each one listens on different network interfaces. Compiling in different base directories for each installation also results automatically in a separate, compiled-in data directory, log file, and PID file location for each server.

Assume that an existing 5.1 server is configured for the default TCP/IP port number (3306) and Unix socket file (`/tmp/mysql.sock`). To configure a new 5.5.31 server to have different operating parameters, use a `CMake` command something like this:

```
shell> cmake . -DMYSQL_TCP_PORT=port_number \
            -DMYSQL_UNIX_ADDR=file_name \
            -DCMAKE_INSTALL_PREFIX=/usr/local/mysql-5.5.31
```

Here, `port_number` and `file_name` must be different from the default TCP/IP port number and Unix socket file path name, and the `CMAKE_INSTALL_PREFIX` [151] value should specify an installation directory different from the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket file name:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

If you specify `localhost` as the host name, `mysqladmin` defaults to using a Unix socket file connection rather than TCP/IP. To explicitly specify the connection protocol, use the `--protocol={TCP|SOCKET|PIPE|MEMORY}` [239] option.

You need not compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to use the same server binary and start each invocation of it with different parameter values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` [464] and `--port` [458] option values, and pass a `--datadir=path` [439] option to `mysqld_safe` so that the server uses a different data directory.

Alternatively, put the options for each server in a different option file, then start each server using a `--defaults-file` [248] option that specifies the path to the appropriate option file. For example, if the option files for two server instances are named `/usr/local/mysql/my.cnf` and `/usr/local/mysql/my.cnf2`, start the servers like this: command:

```
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf2
```

Another way to achieve a similar effect is to use environment variables to set the Unix socket file name and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server.

Section 2.13, “Environment Variables”, includes a list of other environment variables you can use to affect MySQL programs.

On Unix, the `mysqld_multi` script provides another way to start multiple servers. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

5.3.4. Using Client Programs in a Multiple-Server Environment

To connect with a client program to a MySQL server that is listening to different network interfaces from those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name` [239] `--port=port_number` [239] to connect using TCP/IP to a remote server, with `--host=127.0.0.1` [239] `--port=port_number` [239] to connect using TCP/IP to a local server, or with `--host=localhost` [239] `--socket=file_name` [240] to connect to a local server using a Unix socket file or a Windows named pipe.
- Start the client with `--protocol=TCP` [239] to connect using TCP/IP, `--protocol=SOCKET` [239] to connect using a Unix socket file, `--protocol=PIPE` [239] to connect using a named pipe, or `--protocol=MEMORY` [239] to connect using shared memory. For TCP/IP connections, you may also need to specify `--host` [239] and `--port` [239] options. For the other types of connections, you may need to specify a `--socket` [240] option to specify a Unix socket file or Windows named-pipe name, or a `--shared-memory-base-name` [240] option to specify the shared-memory name. Shared-memory connections are supported only on Windows.
- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See Section 2.13, “Environment Variables”.
- Specify the default Unix socket file and TCP/IP port number in the `[client]` group of an option file. For example, you can use `C:\my.cnf` on Windows, or the `.my.cnf` file in your home directory on Unix. See Section 4.2.3.3, “Using Option Files”.

- In a C program, you can specify the socket file or port number arguments in the `mysql_real_connect()` call. You can also have the program read option files by calling `mysql_options()`. See [Section 22.8.3, “C API Function Descriptions”](#).
- If you are using the Perl `DBD::mysql` module, you can read options from MySQL option files. For example:

```
$dsn = "DBI:mysql:test;mysql_read_default_group=client;"  
      . "mysql_read_default_file=/usr/local/mysql/data/my.cnf";  
$dbh = DBI->connect($dsn, $user, $password);
```

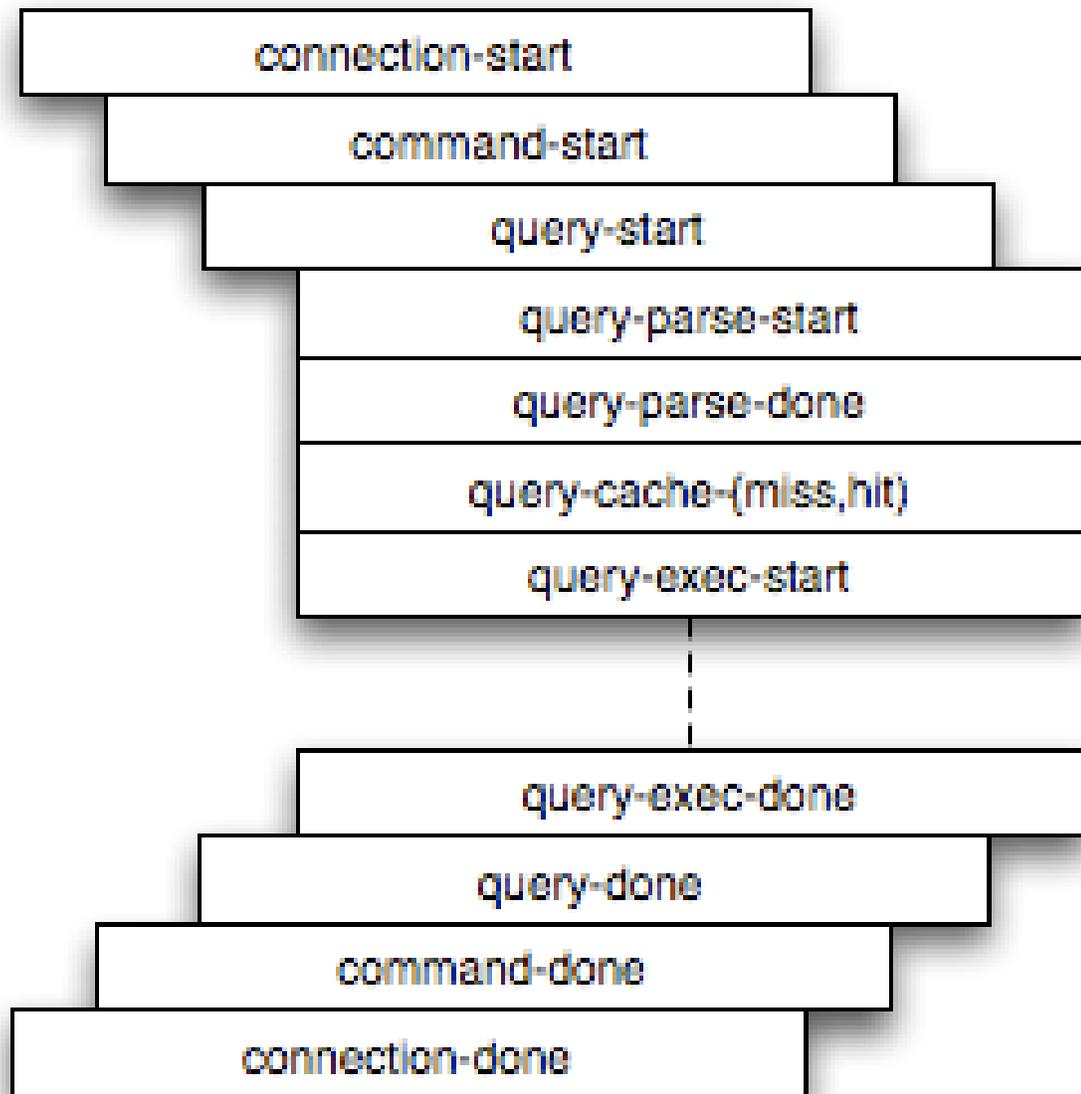
See [Section 22.10, “MySQL Perl API”](#).

Other programming interfaces may provide similar capabilities for reading option files.

5.4. Tracing `mysqld` Using DTrace

The DTrace probes in the MySQL server are designed to provide information about the execution of queries within MySQL and the different areas of the system being utilized during that process. The organization and triggering of the probes means that the execution of an entire query can be monitored with one level of probes (`query-start` and `query-done`) but by monitoring other probes you can get successively more detailed information about the execution of the query in terms of the locks used, sort methods and even row-by-row and storage-engine level execution information.

The DTrace probes are organized so that you can follow the entire query process, from the point of connection from a client, through the query execution, row-level operations, and back out again. You can think of the probes as being fired within a specific sequence during a typical client connect/execute/disconnect sequence, as shown in the following figure.

Figure 5.1. The MySQL Architecture Using Pluggable Storage Engines

Global information is provided in the arguments to the DTrace probes at various levels. Global information, that is, the connection ID and user/host and where relevant the query string, is provided at key levels (`connection-start`, `command-start`, `query-start`, and `query-exec-start`). As you go deeper into the probes, it is assumed either you are only interested in the individual executions (row-level probes provide information on the database and table name only), or that you will combine the row-level probes with the notional parent probes to provide the information about a specific query. Examples of this will be given as the format and arguments of each probe are provided.

For more information on DTrace and writing DTrace scripts, read the [DTrace User Guide](#).

MySQL 5.5 includes support for DTrace probes on Solaris 10 Update 5 (Solaris 5/08) on SPARC, x86 and x86_64 platforms. Probes are also supported on Mac OS X 10.4 and higher. Enabling the probes

should be automatic on these platforms. To explicitly enable or disable the probes during building, use the `-DENABLE_DTRACE=1` [154] or `-DENABLE_DTRACE=0` [154] option to CMake.

5.4.1. mysqld DTrace Probe Reference

MySQL supports the following static probes, organized into groups of functionality.

Table 5.5. MySQL DTrace Probes

Group	Probes	Introduced
Connection	connection-start, connection-done	5.4.0
Command	command-start, command-done	5.4.0
Query	query-start, query-done	5.4.0
Query Parsing	query-parse-start, query-parse-done	5.4.0
Query Cache	query-cache-hit, query-cache-miss	5.4.0
Query Execution	query-exec-start, query-exec-done	5.4.0
Row Level	insert-row-start, insert-row-done	5.4.0
	update-row-start, update-row-done	5.4.0
	delete-row-start, delete-row-done	5.4.0
Row Reads	read-row-start, read-row-done	5.4.0
Index Reads	index-read-row-start, index-read-row-done	5.4.0
Lock	handler-rdlock-start, handler-rdlock-done	5.4.0
	handler-wrlock-start, handler-wrlock-done	5.4.0
	handler-unlock-start, handler-unlock-done	5.4.0
Filesort	filesort-start, filesort-done	5.4.0
Statement	select-start, select-done	5.4.0
	insert-start, insert-done	5.4.0
	insert-select-start, insert-select-done	5.4.0
	update-start, update-done	5.4.0
	multi-update-start, multi-update-done	5.4.0
	delete-start, delete-done	5.4.0
	multi-delete-start, multi-delete-done	5.4.0
Network	net-read-start, net-read-done, net-write-start, net-write-done	5.4.0
Keycache	keycache-read-start, keycache-read-block, keycache-read-done, keycache-read-hit, keycache-read-miss, keycache-write-start, keycache-write-block, keycache-write-done	5.4.0



Note

When extracting the argument data from the probes, each argument is available as `argN`, starting with `arg0`. To identify each argument within the definitions they are provided with a descriptive name, but you must access the information using the corresponding `argN` parameter.

5.4.1.1. Connection Probes

The `connection-start` and `connection-done` probes enclose a connection from a client, regardless of whether the connection is through a socket or network connection.

```
connection-start(connectionid, user, host)
connection-done(status, connectionid)
```

- `connection-start`: Triggered after a connection and successful login/authentication have been completed by a client. The arguments contain the connection information:
 - `connectionid`: An `unsigned long` containing the connection ID. This is the same as the process ID shown as the `Id` value in the output from `SHOW PROCESSLIST`.
 - `user`: The username used when authenticating. The value will be blank for the anonymous user.
 - `host`: The host of the client connection. For a connection made using UNIX sockets, the value will be blank.
- `connection-done`: Triggered just as the connection to the client has been closed. The arguments are:
 - `status`: The status of the connection when it was closed. A logout operation will have a value of 0; any other termination of the connection has a nonzero value.
 - `connectionid`: The connection ID of the connection that was closed.

The following D script will quantify and summarize the average duration of individual connections, and provide a count, dumping the information every 60 seconds:

```
#!/usr/sbin/dtrace -s

mysql*:::connection-start
{
    self->start = timestamp;
}

mysql*:::connection-done
/self->start/
{
    @ = quantize(((timestamp - self->start)/1000000));
    self->start = 0;
}

tick-60s
{
    printa(@);
}
```

When executed on a server with a large number of clients you might see output similar to this:

```
1 57413                               :tick-60s

      value  ----- Distribution ----- count
      -1 |
      0 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 30011
      1 |
      2 |
      4 |
      8 |
     16 |
     32 |
     64 |
                                59
                                5
                                20
                                29
                                18
                                27
                                30
```

128		11
256		10
512		1
1024		6
2048		8
4096		9
8192		8
16384		2
32768		1
65536		1
131072		0
262144		1
524288		0

5.4.1.2. Command Probes

The command probes are executed before and after a client command is executed, including any SQL statement that might be executed during that period. Commands include operations such as the initialization of the DB, use of the `COM_CHANGE_USER` operation (supported by the MySQL protocol), and manipulation of prepared statements. Many of these commands are used only by the MySQL client API from various connectors such as PHP and Java.

```
command-start(connectionid, command, user, host)
command-done(status)
```

- `command-start`: Triggered when a command is submitted to the server.
 - `connectionid`: The connection ID of the client executing the command.
 - `command`: An integer representing the command that was executed. Possible values are shown in the following table.

Value	Name	Description
00	COM_SLEEP	Internal thread state
01	COM_QUIT	Close connection
02	COM_INIT_DB	Select database (<code>USE ...</code>)
03	COM_QUERY	Execute a query
04	COM_FIELD_LIST	Get a list of fields
05	COM_CREATE_DB	Create a database (deprecated)
06	COM_DROP_DB	Drop a database (deprecated)
07	COM_REFRESH	Refresh connection
08	COM_SHUTDOWN	Shutdown server
09	COM_STATISTICS	Get statistics
10	COM_PROCESS_INFO	Get processes (<code>SHOW PROCESSLIST</code>)
11	COM_CONNECT	Initialize connection
12	COM_PROCESS_KILL	Kill process
13	COM_DEBUG	Get debug information
14	COM_PING	Ping
15	COM_TIME	Internal thread state
16	COM_DELAYED_INSERT	Internal thread state
17	COM_CHANGE_USER	Change user

Value	Name	Description
18	COM_BINLOG_DUMP	Used by a replication slave or <code>mysqlbinlog</code> to initiate a binary log read
19	COM_TABLE_DUMP	Used by a replication slave to get the master table information
20	COM_CONNECT_OUT	Used by a replication slave to log a connection to the server
21	COM_REGISTER_SLAVE	Used by a replication slave during registration
22	COM_STMT_PREPARE	Prepare a statement
23	COM_STMT_EXECUTE	Execute a statement
24	COM_STMT_SEND_LONG_DATA	Used by a client when requesting extended data
25	COM_STMT_CLOSE	Close a prepared statement
26	COM_STMT_RESET	Reset a prepared statement
27	COM_SET_OPTIONS	Set a server option
28	COM_STMT_FETCH	Fetch a prepared statement

- `user`: The user executing the command.
- `host`: The client host.
- `command-done`: Triggered when the command execution completes. The `status` argument contains 0 if the command executed successfully, or 1 if the statement was terminated before normal completion.

The `command-start` and `command-done` probes are best used when combined with the statement probes to get an idea of overall execution time.

5.4.1.3. Query Probes

The `query-start` and `query-done` probes are triggered when a specific query is received by the server and when the query has been completed and the information has been successfully sent to the client.

```
query-start(query, connectionid, database, user, host)
query-done(status)
```

- `query-start`: Triggered after the query string has been received from the client. The arguments are:
 - `query`: The full text of the submitted query.
 - `connectionid`: The connection ID of the client that submitted the query. The connection ID equals the connection ID returned when the client first connects and the `Id` value in the output from `SHOW PROCESSLIST`.
 - `database`: The database name on which the query is being executed.
 - `user`: The username used to connect to the server.
 - `host`: The hostname of the client.
- `query-done`: Triggered once the query has been executed and the information has been returned to the client. The probe includes a single argument, `status`, which returns 0 when the query is successfully executed and 1 if there was an error.

You can get a simple report of the execution time for each query using the following D script:

```
#!/usr/sbin/dtrace -s
```

```
#pragma D option quiet

dtrace:::BEGIN
{
    printf("%-20s %-20s %-40s %-9s\n", "Who", "Database", "Query", "Time(ms)");
}

mysql*:::query-start
{
    self->query = copyinstr(arg0);
    self->connid = arg1;
    self->db = copyinstr(arg2);
    self->who = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
    self->querystart = timestamp;
}

mysql*:::query-done
{
    printf("%-20s %-20s %-40s %-9d\n",self->who,self->db,self->query,
        (timestamp - self->querystart) / 1000000);
}
```

When executing the above script you should get a basic idea of the execution time of your queries:

```
shell> ./query.d
Who Database Query Time(ms)
root@localhost test select * from t1 order by i limit 10 0
root@localhost test set global query_cache_size=0 0
root@localhost test select * from t1 order by i limit 10 776
root@localhost test select * from t1 order by i limit 10 773
root@localhost test select * from t1 order by i desc limit 10 795
```

5.4.1.4. Query Parsing Probes

The query parsing probes are triggered before the original SQL statement is parsed and when the parsing of the statement and determination of the execution model required to process the statement has been completed:

```
query-parse-start(query)
query-parse-done(status)
```

- **query-parse-start**: Triggered just before the statement is parsed by the MySQL query parser. The single argument, **query**, is a string containing the full text of the original query.
- **query-parse-done**: Triggered when the parsing of the original statement has been completed. The **status** is an integer describing the status of the operation. A **0** indicates that the query was successfully parsed. A **1** indicates that the parsing of the query failed.

For example, you could monitor the execution time for parsing a given query using the following D script:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

mysql*:::query-parse-start
{
    self->parsestart = timestamp;
    self->parsequery = copyinstr(arg0);
}

mysql*:::query-parse-done
/arg0 == 0/
{
    printf("Parsing %s: %d microseconds\n", self->parsequery,((timestamp - self->parsestart)/1000));
}
```

```

}

mysql*:::query-parse-done
/arg0 != 0/
{
    printf("Error parsing %s: %d microseconds\n", self->parsequery,((timestamp - self->parsestart)/1000));
}

```

In the above script a predicate is used on `query-parse-done` so that different output is generated based on the status value of the probe.

When running the script and monitoring the execution:

```

shell> ./query-parsing.d
Error parsing select from t1 join (t2) on (t1.i = t2.i) order by t1.s,t1.i limit 10: 36 ms
Parsing select * from t1 join (t2) on (t1.i = t2.i) order by t1.s,t1.i limit 10: 176 ms

```

5.4.1.5. Query Cache Probes

The query cache probes are fired when executing any query. The `query-cache-hit` query is triggered when a query exists in the query cache and can be used to return the query cache information. The arguments contain the original query text and the number of rows returned from the query cache for the query. If the query is not within the query cache, or the query cache is not enabled, then the `query-cache-miss` probe is triggered instead.

```

query-cache-hit(query, rows)
query-cache-miss(query)

```

- `query-cache-hit`: Triggered when the query has been found within the query cache. The first argument, `query`, contains the original text of the query. The second argument, `rows`, is an integer containing the number of rows in the cached query.
- `query-cache-miss`: Triggered when the query is not found within the query cache. The first argument, `query`, contains the original text of the query.

The query cache probes are best combined with a probe on the main query so that you can determine the differences in times between using or not using the query cache for specified queries. For example, in the following D script, the query and query cache information are combined into the information output during monitoring:

```

#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
    printf("%-20s %-20s %-40s %2s %-9s\n", "Who", "Database", "Query", "QC", "Time(ms)");
}

mysql*:::query-start
{
    self->query = copyinstr(arg0);
    self->connid = arg1;
    self->db = copyinstr(arg2);
    self->who = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
    self->querystart = timestamp;
    self->qc = 0;
}

mysql*:::query-cache-hit
{
    self->qc = 1;
}

```

```
mysql*:::query-cache-miss
{
    self->qc = 0;
}

mysql*:::query-done
{
    printf("%-20s %-20s %-40s %-2s %-9d\n",self->who,self->db,self->query,(self->qc ? "Y" : "N"),
        (timestamp - self->querystart) / 1000000);
}
```

When executing the script you can see the effects of the query cache. Initially the query cache is disabled. If you set the query cache size and then execute the query multiple times you should see that the query cache is being used to return the query data:

```
shell> ./query-cache.d
root@localhost      test      select * from t1 order by i limit 10      N 1072
root@localhost      test      set global query_cache_size=262144      N 0
root@localhost      test      select * from t1 order by i limit 10      N 781
root@localhost      test      select * from t1 order by i limit 10      Y 0
```

5.4.1.6. Query Execution Probes

The query execution probe is triggered when the actual execution of the query starts, after the parsing and checking the query cache but before any privilege checks or optimization. By comparing the difference between the start and done probes you can monitor the time actually spent servicing the query (instead of just handling the parsing and other elements of the query).

```
query-exec-start(query, connectionid, database, user, host, exec_type)
query-exec-done(status)
```



Note

The information provided in the arguments for `query-start` and `query-exec-start` are almost identical and designed so that you can choose to monitor either the entire query process (using `query-start`) or only the execution (using `query-exec-start`) while exposing the core information about the user, client, and query being executed.

- `query-exec-start`: Triggered when the execution of a individual query is started. The arguments are:
 - `query`: The full text of the submitted query.
 - `connectionid`: The connection ID of the client that submitted the query. The connection ID equals the connection ID returned when the client first connects and the `Id` value in the output from `SHOW PROCESSLIST`.
 - `database`: The database name on which the query is being executed.
 - `user`: The username used to connect to the server.
 - `host`: The hostname of the client.
 - `exec_type`: The type of execution. Execution types are determined based on the contents of the query and where it was submitted. The values for each type are shown in the following table.

Value	Description
0	Executed query from <code>sql_parse</code> , top-level query.

Value	Description
1	Executed prepared statement
2	Executed cursor statement
3	Executed query in stored procedure

- `query-exec-done`: Triggered when the execution of the query has completed. The probe includes a single argument, `status`, which returns 0 when the query is successfully executed and 1 if there was an error.

5.4.1.7. Row-Level Probes

The `*row-{start,done}` probes are triggered each time a row operation is pushed down to a storage engine. For example, if you execute an `INSERT` statement with 100 rows of data, then the `insert-row-start` and `insert-row-done` probes will be triggered 100 times each, for each row insert.

```
insert-row-start(database, table)
insert-row-done(status)

update-row-start(database, table)
update-row-done(status)

delete-row-start(database, table)
delete-row-done(status)
```

- `insert-row-start`: Triggered before a row is inserted into a table.
- `insert-row-done`: Triggered after a row is inserted into a table.
- `update-row-start`: Triggered before a row is updated in a table.
- `update-row-done`: Triggered before a row is updated in a table.
- `delete-row-start`: Triggered before a row is deleted from a table.
- `delete-row-done`: Triggered before a row is deleted from a table.

The arguments supported by the probes are consistent for the corresponding `start` and `done` probes in each case:

- `database`: The database name.
- `table`: The table name.
- `status`: The status; 0 for success or 1 for failure.

Because the row-level probes are triggered for each individual row access, these probes can be triggered many thousands of times each second, which may have a detrimental effect on both the monitoring script and MySQL. The DTrace environment should limit the triggering on these probes to prevent the performance being adversely affected. Either use the probes sparingly, or use counter or aggregation functions to report on these probes and then provide a summary when the script terminates or as part of a `query-done` or `query-exec-done` probes.

The following example script summarizes the duration of each row operation within a larger query:

```
#!/usr/sbin/dtrace -s
#pragma D option quiet
```

```

dtrace:::BEGIN
{
    printf("%-2s %-10s %-10s %9s %9s %-s \n",
          "St", "Who", "DB", "ConnID", "Dur ms", "Query");
}

mysql*:::query-start
{
    self->query = copyinstr(arg0);
    self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
    self->db    = copyinstr(arg2);
    self->connid = arg1;
    self->querystart = timestamp;
    self->rowdur = 0;
}

mysql*:::query-done
{
    this->elapsed = (timestamp - self->querystart) /1000000;
    printf("%2d %-10s %-10s %9d %9d %s\n",
          arg0, self->who, self->db,
          self->connid, this->elapsed, self->query);
}

mysql*:::query-done
/ self->rowdur /
{
    printf("%34s %9d %s\n", "", (self->rowdur/1000000), "-> Row ops");
}

mysql*:::insert-row-start
{
    self->rowstart = timestamp;
}

mysql*:::delete-row-start
{
    self->rowstart = timestamp;
}

mysql*:::update-row-start
{
    self->rowstart = timestamp;
}

mysql*:::insert-row-done
{
    self->rowdur += (timestamp-self->rowstart);
}

mysql*:::delete-row-done
{
    self->rowdur += (timestamp-self->rowstart);
}

mysql*:::update-row-done
{
    self->rowdur += (timestamp-self->rowstart);
}

```

Running the above script with a query that inserts data into a table, you can monitor the exact time spent performing the raw row insertion:

St	Who	DB	ConnID	Dur ms	Query
0	@localhost	test	13	20767	insert into t1(select * from t2)
				4827	-> Row ops

5.4.1.8. Read Row Probes

The read row probes are triggered at a storage engine level each time a row read operation occurs. These probes are specified within each storage engine (as opposed to the `*row-start` probes which are in the storage engine interface). These probes can therefore be used to monitor individual storage engine row-level operations and performance. Because these probes are triggered around the storage engine row read interface, they may be hit a significant number of times during a basic query.

```
read-row-start(database, table, scan_flag)
read-row-done(status)
```

- `read-row-start`: Triggered when a row is read by the storage engine from the specified `database` and `table`. The `scan_flag` is set to 1 (true) when the read is part of a table scan (that is, a sequential read), or 0 (false) when the read is of a specific record.
- `read-row-done`: Triggered when a row read operation within a storage engine completes. The `status` returns 0 on success, or a positive value on failure.

5.4.1.9. Index Probes

The index probes are triggered each time a a row is read using one of the indexes for the specified table. The probe is triggered within the corresponding storage engine for the table.

```
index-read-row-start(database, table)
index-read-row-done(status)
```

- `index-read-row-start`: Triggered when a row is read by the storage engine from the specified `database` and `table`.
- `index-read-row-done`: Triggered when an indexed row read operation within a storage engine completes. The `status` returns 0 on success, or a positive value on failure.

5.4.1.10. Lock Probes

The lock probes are called whenever an external lock is requested by MySQL for a table using the corresponding lock mechanism on the table as defined by the table's engine type. There are three different types of lock, the read lock, write lock, and unlock operations. Using the probes you can determine the duration of the external locking routine (that is, the time taken by the storage engine to implement the lock, including any time waiting for another lock to become free) and the total duration of the lock/unlock process.

```
handler-rdlock-start(database, table)
handler-rdlock-done(status)

handler-wrlock-start(database, table)
handler-wrlock-done(status)

handler-unlock-start(database, table)
handler-unlock-done(status)
```

- `handler-rdlock-start`: Triggered when a read lock is requested on the specified `database` and `table`.
- `handler-wrlock-start`: Triggered when a write lock is requested on the specified `database` and `table`.
- `handler-unlock-start`: Triggered when an unlock request is made on the specified `database` and `table`.

- `handler-rdlock-done`: Triggered when a read lock request completes. The `status` is 0 if the lock operation succeeded, or `>0` on failure.
- `handler-wrlock-done`: Triggered when a write lock request completes. The `status` is 0 if the lock operation succeeded, or `>0` on failure.
- `handler-unlock-done`: Triggered when an unlock request completes. The `status` is 0 if the unlock operation succeeded, or `>0` on failure.

You can use arrays to monitor the locking and unlocking of individual tables and then calculate the duration of the entire table lock using the following script:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

mysql*:::handler-rdlock-start
{
    self->rdlockstart = timestamp;
    this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
    self->lockmap[this->lockref] = self->rdlockstart;
    printf("Start: Lock->Read   %s.%s\n",copyinstr(arg0),copyinstr(arg1));
}

mysql*:::handler-wrlock-start
{
    self->wrlockstart = timestamp;
    this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
    self->lockmap[this->lockref] = self->rdlockstart;
    printf("Start: Lock->Write  %s.%s\n",copyinstr(arg0),copyinstr(arg1));
}

mysql*:::handler-unlock-start
{
    self->unlockstart = timestamp;
    this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
    printf("Start: Lock->Unlock %s.%s (%d ms lock duration)\n",
        copyinstr(arg0),copyinstr(arg1),
        (timestamp - self->lockmap[this->lockref])/1000000);
}

mysql*:::handler-rdlock-done
{
    printf("End:   Lock->Read   %d ms\n",
        (timestamp - self->rdlockstart)/1000000);
}

mysql*:::handler-wrlock-done
{
    printf("End:   Lock->Write  %d ms\n",
        (timestamp - self->wrlockstart)/1000000);
}

mysql*:::handler-unlock-done
{
    printf("End:   Lock->Unlock %d ms\n",
        (timestamp - self->unlockstart)/1000000);
}
```

When executed, you should get information both about the duration of the locking process itself, and of the locks on a specific table:

```
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
Start: Lock->Unlock test.t2 (25743 ms lock duration)
```

```

End:   Lock->Unlock 0 ms
Start: Lock->Read  test.t2
End:   Lock->Read  0 ms
Start: Lock->Unlock test.t2 (1 ms lock duration)
End:   Lock->Unlock 0 ms
Start: Lock->Read  test.t2
End:   Lock->Read  0 ms
Start: Lock->Unlock test.t2 (1 ms lock duration)
End:   Lock->Unlock 0 ms
Start: Lock->Read  test.t2
End:   Lock->Read  0 ms

```

5.4.1.11. Filesort Probes

The filesort probes are triggered whenever a filesort operation is applied to a table. For more information on filesort and the conditions under which it occurs, see [Section 8.13.9, “ORDER BY Optimization”](#).

```

filesort-start(database, table)
filesort-done(status, rows)

```

- **filesort-start**: Triggered when the filesort operation starts on a table. The two arguments to the probe, `database` and `table`, will identify the table being sorted.
- **filesort-done**: Triggered when the filesort operation completes. Two arguments are supplied, the `status` (0 for success, 1 for failure), and the number of rows sorted during the filesort process.

An example of this is in the following script, which tracks the duration of the filesort process in addition to the duration of the main query:

```

#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
    printf("%-2s %-10s %-10s %9s %18s %-s \n",
        "St", "Who", "DB", "ConnID", "Dur microsec", "Query");
}

mysql*:::query-start
{
    self->query = copyinstr(arg0);
    self->who   = strjoin(copyinstr(arg3), strjoin("@", copyinstr(arg4)));
    self->db    = copyinstr(arg2);
    self->connid = arg1;
    self->querystart = timestamp;
    self->filesort = 0;
    self->fsdb = "";
    self->fstable = "";
}

mysql*:::filesort-start
{
    self->filesort = timestamp;
    self->fsdb = copyinstr(arg0);
    self->fstable = copyinstr(arg1);
}

mysql*:::filesort-done
{
    this->elapsed = (timestamp - self->filesort) / 1000;
    printf("%2d %-10s %-10s %9d %18d Filesort on %s\n",
        arg0, self->who, self->fsdb,
        self->connid, this->elapsed, self->fstable);
}

```

```
mysql*:::query-done
{
    this->elapsed = (timestamp - self->querystart) /1000;
    printf("%2d %-10s %-10s %9d %18d %s\n",
        arg0, self->who, self->db,
        self->connid, this->elapsed, self->query);
}
```

Executing a query on a large table with an `ORDER BY` clause that triggers a filesort, and then creating an index on the table and then repeating the same query, you can see the difference in execution speed:

St	Who	DB	ConnID	Dur	microsec	Query
0	@localhost	test	14		11335469	Filesort on t1
0	@localhost	test	14		11335787	select * from t1 order by i limit 100
0	@localhost	test	14		466734378	create index t1a on t1 (i)
0	@localhost	test	14		26472	select * from t1 order by i limit 100

5.4.1.12. Statement Probes

The individual statement probes are provided to give specific information about different statement types. For the start probes the string of the query is provided as the only argument. Depending on the statement type, the information provided by the corresponding done probe will differ. For all done probes the status of the operation (0 for success, >0 for failure) is provided. For `SELECT`, `INSERT`, `INSERT ... (SELECT FROM ...)`, `DELETE`, and `DELETE FROM t1,t2` operations the number of rows affected is returned.

For `UPDATE` and `UPDATE t1,t2 ...` statements the number of rows matched and the number of rows actually changed is provided. This is because the number of rows actually matched by the corresponding `WHERE` clause, and the number of rows changed can differ. MySQL does not update the value of a row if the value already matches the new setting.

```
select-start(query)
select-done(status,rows)

insert-start(query)
insert-done(status,rows)

insert-select-start(query)
insert-select-done(status,rows)

update-start(query)
update-done(status,rowsmatched,rowschanged)

multi-update-start(query)
multi-update-done(status,rowsmatched,rowschanged)

delete-start(query)
delete-done(status,rows)

multi-delete-start(query)
multi-delete-done(status,rows)
```

- `select-start`: Triggered before a `SELECT` statement.
- `select-done`: Triggered at the end of a `SELECT` statement.
- `insert-start`: Triggered before a `INSERT` statement.
- `insert-done`: Triggered at the end of an `INSERT` statement.
- `insert-select-start`: Triggered before an `INSERT ... SELECT` statement.
- `insert-select-done`: Triggered at the end of an `INSERT ... SELECT` statement.

- `update-start`: Triggered before an `UPDATE` statement.
- `update-done`: Triggered at the end of an `UPDATE` statement.
- `multi-update-start`: Triggered before an `UPDATE` statement involving multiple tables.
- `multi-update-done`: Triggered at the end of an `UPDATE` statement involving multiple tables.
- `delete-start`: Triggered before a `DELETE` statement.
- `delete-done`: Triggered at the end of a `DELETE` statement.
- `multi-delete-start`: Triggered before a `DELETE` statement involving multiple tables.
- `multi-delete-done`: Triggered at the end of a `DELETE` statement involving multiple tables.

The arguments for the statement probes are:

- `query`: The query string.
- `status`: The status of the query. `0` for success, and `>0` for failure.
- `rows`: The number of rows affected by the statement. This returns the number rows found for `SELECT`, the number of rows deleted for `DELETE`, and the number of rows successfully inserted for `INSERT`.
- `rowsmatched`: The number of rows matched by the `WHERE` clause of an `UPDATE` operation.
- `rowschanged`: The number of rows actually changed during an `UPDATE` operation.

You use these probes to monitor the execution of these statement types without having to monitor the user or client executing the statements. A simple example of this is to track the execution times:

```
#!/usr/sbin/dtrace -s
#pragma D option quiet
dtrace:::BEGIN
{
    printf("%-60s %-8s %-8s %-8s\n", "Query", "RowsU", "RowsM", "Dur (ms)");
}

mysql*:::update-start, mysql*:::insert-start,
mysql*:::delete-start, mysql*:::multi-delete-start,
mysql*:::multi-delete-done, mysql*:::select-start,
mysql*:::insert-select-start, mysql*:::multi-update-start
{
    self->query = copyinstr(arg0);
    self->querystart = timestamp;
}

mysql*:::insert-done, mysql*:::select-done,
mysql*:::delete-done, mysql*:::multi-delete-done, mysql*:::insert-select-done
/ self->querystart /
{
    this->elapsed = ((timestamp - self->querystart)/1000000);
    printf("%-60s %-8d %-8d %-8d\n",
        self->query,
        0,
        arg1,
        this->elapsed);
    self->querystart = 0;
}
```

```
mysql*:::update-done, mysql*:::multi-update-done
/ self->querystart /
{
    this->elapsed = ((timestamp - self->querystart)/1000000);
    printf("%-60s %-8d %-8d %d\n",
        self->query,
        arg1,
        arg2,
        this->elapsed);
    self->querystart = 0;
}
```

When executed you can see the basic execution times and rows matches:

Query	RowsU	RowsM	Dur (ms)
select * from t2	0	275	0
insert into t2 (select * from t2)	0	275	9
update t2 set i=5 where i > 75	110	110	8
update t2 set i=5 where i < 25	254	134	12
delete from t2 where i < 5	0	0	0

Another alternative is to use the aggregation functions in DTrace to aggregate the execution time of individual statements together:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

mysql*:::update-start, mysql*:::insert-start,
mysql*:::delete-start, mysql*:::multi-delete-start,
mysql*:::multi-delete-done, mysql*:::select-start,
mysql*:::insert-select-start, mysql*:::multi-update-start
{
    self->querystart = timestamp;
}

mysql*:::select-done
{
    @statements["select"] = sum(((timestamp - self->querystart)/1000000));
}

mysql*:::insert-done, mysql*:::insert-select-done
{
    @statements["insert"] = sum(((timestamp - self->querystart)/1000000));
}

mysql*:::update-done, mysql*:::multi-update-done
{
    @statements["update"] = sum(((timestamp - self->querystart)/1000000));
}

mysql*:::delete-done, mysql*:::multi-delete-done
{
    @statements["delete"] = sum(((timestamp - self->querystart)/1000000));
}

tick-30s
{
    printa(@statements);
}
```

The script just shown aggregates the times spent doing each operation, which could be used to help benchmark a standard suite of tests.

```
delete
```

```
0
```

update	0
insert	23
select	2484
delete	0
update	0
insert	39
select	10744
delete	0
update	26
insert	56
select	10944
delete	0
update	26
insert	2287
select	15985

5.4.1.13. Network Probes

The network probes monitor the transfer of information from the MySQL server and clients of all types over the network. The probes are defined as follows:

```
net-read-start()
net-read-done(status, bytes)
net-write-start(bytes)
net-write-done(status)
```

- **net-read-start**: Triggered when a network read operation is started.
- **net-read-done**: Triggered when the network read operation completes. The `status` is an `integer` representing the return status for the operation, `0` for success and `1` for failure. The `bytes` argument is an integer specifying the number of bytes read during the process.
- **net-start-bytes**: Triggered when data is written to a network socket. The single argument, `bytes`, specifies the number of bytes written to the network socket.
- **net-write-done**: Triggered when the network write operation has completed. The single argument, `status`, is an integer representing the return status for the operation, `0` for success and `1` for failure.

You can use the network probes to monitor the time spent reading from and writing to network clients during execution. The following D script provides an example of this. Both the cumulative time for the read or write is calculated, and the number of bytes. Note that the dynamic variable size has been increased (using the `dynvarsize` option) to cope with the rapid firing of the individual probes for the network reads/writes.

```
#!/usr/sbin/dtrace -s

#pragma D option quiet
#pragma D option dynvarsize=4m

dtrace:::BEGIN
{
    printf("%-2s %-30s %-10s %9s %18s %-s \n",
          "St", "Who", "DB", "ConnID", "Dur microsec", "Query");
}

mysql*:::query-start
{
    self->query = copyinstr(arg0);
    self->who   = strjoin(copyinstr(arg3), strjoin("@", copyinstr(arg4)));
    self->db    = copyinstr(arg2);
}
```

```

self->connid = arg1;
self->querystart = timestamp;
self->netwrite = 0;
self->netwritecum = 0;
self->netwritebase = 0;
self->netread = 0;
self->netreadcum = 0;
self->netreadbase = 0;
}

mysql*:::net-write-start
{
    self->netwrite += arg0;
    self->netwritebase = timestamp;
}

mysql*:::net-write-done
{
    self->netwritecum += (timestamp - self->netwritebase);
    self->netwritebase = 0;
}

mysql*:::net-read-start
{
    self->netreadbase = timestamp;
}

mysql*:::net-read-done
{
    self->netread += arg1;
    self->netreadcum += (timestamp - self->netreadbase);
    self->netreadbase = 0;
}

mysql*:::query-done
{
    this->elapsed = (timestamp - self->querystart) / 1000000;
    printf("%2d %-30s %-10s %9d %18d %s\n",
        arg0, self->who, self->db,
        self->connid, this->elapsed, self->query);
    printf("Net read: %d bytes (%d ms) write: %d bytes (%d ms)\n",
        self->netread, (self->netreadcum/1000000),
        self->netwrite, (self->netwritecum/1000000));
}

```

When executing the above script on a machine with a remote client, you can see that approximately a third of the time spent executing the query is related to writing the query results back to the client.

St	Who	DB	ConnID	Dur	microsec	Query
0	root@::ffff:192.168.0.108	test	31		3495	select * from t1 limit 1000000
Net read: 0 bytes (0 ms) write: 10000075 bytes (1220 ms)						

5.4.1.14. Keycache Probes

The keycache probes are triggered when using the index key cache used with the MyISAM storage engine. Probes exist to monitor when data is read into the keycache, cached key data is written from the cache into a cached file, or when accessing the keycache.

Keycache usage indicates when data is read or written from the index files into the cache, and can be used to monitor how efficient the memory allocated to the keycache is being used. A high number of keycache reads across a range of queries may indicate that the keycache is too small for size of data being accessed.

```
keycache-read-start(filepath, bytes, mem_used, mem_free)
```

```
keycache-read-block(bytes)
keycache-read-hit()
keycache-read-miss()
keycache-read-done(mem_used, mem_free)
keycache-write-start(filepath, bytes, mem_used, mem_free)
keycache-write-block(bytes)
keycache-write-done(mem_used, mem_free)
```

When reading data from the index files into the keycache, the process first initializes the read operation (indicated by `keycache-read-start`), then loads blocks of data (`keycache-read-block`), and then the read block is either matches the data being identified (`keycache-read-hit`) or more data needs to be read (`keycache-read-miss`). Once the read operation has completed, reading stops with the `keycache-read-done`.

Data will be read from the index file into the keycache only when the specified key is not already within the keycache.

- `keycache-read-start`: Triggered when the keycache read operation is started. Data is read from the specified `filepath`, reading the specified number of `bytes`. The `mem_used` and `mem_avail` indicate memory currently used by the keycache and the amount of memory available within the keycache.
- `keycache-read-block`: Triggered when the keycache reads a block of data, of the specified number of `bytes`, from the index file into the keycache.
- `keycache-read-hit`: Triggered when the block of data read from the index file matches the key data requested.
- `keycache-read-miss`: Triggered when the block of data read from the index file does not match the key data needed.
- `keycache-read-done`: Triggered when the keycache read operation has completed. The `mem_used` and `mem_avail` indicate memory currently used by the keycache and the amount of memory available within the keycache.

Keycache writes occur when the index information is updated during an `INSERT`, `UPDATE`, or `DELETE` operation, and the cached key information is flushed back to the index file.

- `keycache-write-start`: Triggered when the keycache write operation is started. Data is written to the specified `filepath`, reading the specified number of `bytes`. The `mem_used` and `mem_avail` indicate memory currently used by the keycache and the amount of memory available within the keycache.
- `keycache-write-block`: Triggered when the keycache writes a block of data, of the specified number of `bytes`, to the index file from the keycache.
- `keycache-write-done`: Triggered when the keycache write operation has completed. The `mem_used` and `mem_avail` indicate memory currently used by the keycache and the amount of memory available within the keycache.

Chapter 6. Security

Table of Contents

6.1. General Security Issues	678
6.1.1. Security Guidelines	678
6.1.2. Keeping Passwords Secure	679
6.1.3. Making MySQL Secure Against Attackers	687
6.1.4. Security-Related <code>mysqld</code> Options and Variables	689
6.1.5. How to Run MySQL as a Normal User	690
6.1.6. Security Issues with <code>LOAD DATA LOCAL</code>	690
6.1.7. Client Programming Security Guidelines	691
6.2. The MySQL Access Privilege System	692
6.2.1. Privileges Provided by MySQL	693
6.2.2. Privilege System Grant Tables	698
6.2.3. Specifying Account Names	703
6.2.4. Access Control, Stage 1: Connection Verification	705
6.2.5. Access Control, Stage 2: Request Verification	708
6.2.6. When Privilege Changes Take Effect	710
6.2.7. Causes of Access-Denied Errors	711
6.3. MySQL User Account Management	716
6.3.1. User Names and Passwords	716
6.3.2. Adding User Accounts	718
6.3.3. Removing User Accounts	721
6.3.4. Setting Account Resource Limits	721
6.3.5. Assigning Account Passwords	723
6.3.6. Pluggable Authentication	725
6.3.7. Proxy Users	742
6.3.8. Using SSL for Secure Connections	745
6.3.9. Connecting to MySQL Remotely from Windows with SSH	756
6.3.10. MySQL Enterprise Audit	757
6.3.11. SQL-Based MySQL Account Activity Auditing	768

When thinking about security within a MySQL installation, you should consider a wide range of possible topics and how they affect the security of your MySQL server and related applications:

- General factors that affect security. These include choosing good passwords, not granting unnecessary privileges to users, ensuring application security by preventing SQL injections and data corruption, and others. See [Section 6.1, “General Security Issues”](#).
- Security of the installation itself. The data files, log files, and the all the application files of your installation should be protected to ensure that they are not readable or writable by unauthorized parties. For more information, see [Section 2.11, “Postinstallation Setup and Testing”](#).
- Access control and security within the database system itself, including the users and databases granted with access to the databases, views and stored programs in use within the database. For more information, see [Section 6.2, “The MySQL Access Privilege System”](#), and [Section 6.3, “MySQL User Account Management”](#).
- Network security of MySQL and your system. The security is related to the grants for individual users, but you may also wish to restrict MySQL so that it is available only locally on the MySQL server host, or to a limited set of other hosts.

- Ensure that you have adequate and appropriate backups of your database files, configuration and log files. Also be sure that you have a recovery solution in place and test that you are able to successfully recover the information from your backups. See [Chapter 7, Backup and Recovery](#).

6.1. General Security Issues

This section describes general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see [Section 6.2, “The MySQL Access Privilege System”](#).

For answers to some questions that are often asked about MySQL Server security issues, see [Section B.9, “MySQL 5.5 FAQ: Security”](#).

6.1.1. Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, it is necessary to consider fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines:

- **Do not ever give anyone (except MySQL `root` accounts) access to the `user` table in the `mysql` database!** This is critical.
- Learn how the MySQL access privilege system works (see [Section 6.2, “The MySQL Access Privilege System”](#)). Use the `GRANT` and `REVOKE` statements to control access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

Checklist:

- Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL `root` user with full privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a `root` password. See [Section 2.11.2, “Securing the Initial MySQL Accounts”](#).
- Use the `SHOW GRANTS` statement to check which accounts have access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.
- Do not store cleartext passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `SHA2()` [\[1189\]](#), `SHA1()` [\[1189\]](#), `MD5()` [\[1187\]](#), or some other one-way hashing function and store the hash value.
- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “duag98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Four score and seven years ago” results in a password of “Fsasya”). The password is easy to remember and type, but difficult to

guess for someone who does not know the sentence. In this case, you can additionally substitute digits for the number words to obtain the phrase “4 score and 7 years ago”, yielding the password “4sa7ya” which is even more difficult to guess.

- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

Checklist:

- Try to scan your ports from the Internet using a tool such as `nmap`. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. As a simple way to check whether your MySQL port is open, try the following command from some remote machine, where `server_host` is the host name or IP address of the host on which your MySQL server runs:

```
shell> telnet server_host 3306
```

If `telnet` hangs or the connection is refused, the port is blocked, which is how you want it to be. If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open.

- Applications that access MySQL should not trust any data entered by users, and should be written using proper defensive programming techniques. See [Section 6.1.7, “Client Programming Security Guidelines”](#).
- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.
- Learn to use the `tcpdump` and `strings` utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

This works under Linux and should work with small modifications under other systems.



Warning

If you do not see cleartext data, this does not always mean that the information actually is encrypted. If you need high security, consult with a security expert.

6.1.2. Keeping Passwords Secure

Passwords occur in several contexts within MySQL. The following sections provide guidelines that enable end users and administrators to keep these passwords secure and avoid exposing them. There is also a discussion of how MySQL uses password hashing internally.

6.1.2.1. End-User Guidelines for Password Security

MySQL users should use the following guidelines to keep passwords secure.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method. In short, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

- Use a `-pyour_pass` or `--password=your_pass` option on the command line. For example:

```
shell> mysql -u francis -pfrank db_name
```

This is convenient *but insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If your operating environment is set up to display your current command in the title bar of your terminal window, the password remains visible as long as the command is running, even if the command has scrolled out of view in the window content area.

- Use the `-p` or `--password` option on the command line with no password value specified. In this case, the client program solicits the password interactively:

```
shell> mysql -u francis -p db_name
Enter password: *****
```

The “*” characters indicate where you enter your password. The password is not displayed as you enter it.

It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs noninteractively, there is no opportunity to enter the password from the keyboard. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- Store your password in an option file. For example, on Unix, you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

```
[client]
password=your_pass
```

To keep the password safe, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

```
shell> chmod 600 .my.cnf
```

To name from the command line a specific option file containing the password, use the `--defaults-file=file_name` [248] option, where `file_name` is the full path name to the file. For example:

```
shell> mysql --defaults-file=/home/francis/mysql-opts
```

[Section 4.2.3.3, “Using Option Files”](#), discusses option files in more detail.

- Store your password in the `MYSQL_PWD` environment variable. See [Section 2.13, “Environment Variables”](#).

This method of specifying your MySQL password must be considered *extremely insecure* and should not be used. Some versions of `ps` include an option to display the environment of running processes. On some systems, if you set `MYSQL_PWD`, your password is exposed to any other user who runs `ps`. Even on systems without such a version of `ps`, it is unwise to assume that there are no other methods by which users can examine process environments.

On Unix, the `mysql` client writes a record of executed statements to a history file (see [Section 4.5.1.3, “mysql History File”](#)). By default, this file is named `.mysql_history` and is created in your home

directory. Passwords can appear as plain text in SQL statements such as `CREATE USER`, `GRANT`, and `SET PASSWORD`, so if you use these statements, they are logged in the history file. To keep this file safe, use a restrictive access mode, the same way as described earlier for the `.my.cnf` file.

If your command interpreter is configured to maintain a history, any file in which the commands are saved will contain MySQL passwords entered on the command line. For example, `bash` uses `~/.bash_history`. Any such file should have a restrictive access mode.

6.1.2.2. Administrator Guidelines for Password Security

Database administrators should use the following guidelines to keep passwords secure.

MySQL stores passwords for user accounts in the `mysql.user` table. Access to this table should never be granted to any nonadministrative accounts.

A user who has access to modify the plugin directory (the value of the `plugin_dir` [537] system variable) or the `my.cnf` file that specifies the location of the plugin directory can replace plugins and modify the capabilities provided by plugins, including authentication plugins.

Files such as log files to which passwords might be written should be protected. See [Section 6.1.2.3, “Passwords and Logging”](#).

6.1.2.3. Passwords and Logging

Passwords can appear as plain text in SQL statements such as `CREATE USER`, `GRANT`, and `SET PASSWORD`, or statements that invoke the `PASSWORD()` [1188] function. If these statements are logged by the MySQL server, the passwords become available to anyone with access to the logs. This applies to the general query log, the slow query log, and the binary log (see [Section 5.2, “MySQL Server Logs”](#)). To guard against unwarranted exposure to log files, they should be located in a directory that restricts access to only the server and the database administrator. If you log to tables in the `mysql` database, access to the tables should never be granted to any nonadministrative accounts.

Replication slaves store the password for the replication master in the `master.info` file. Restrict this file to be accessible only to the database administrator.

Database backups that include tables or log files containing passwords should be protected using a restricted access mode.

6.1.2.4. Password Hashing in MySQL

MySQL lists user accounts in the `user` table of the `mysql` database. Each MySQL account can be assigned a password, although the `user` table does not store the cleartext version of the password, but a hash value computed from it.

MySQL uses passwords in two phases of client/server communication:

- When a client attempts to connect to the server, there is an initial authentication step in which the client must present a password that has a hash value matching the hash value stored in the `user` table for the account the client wants to use.
- After the client connects, it can (if it has sufficient privileges) set or change the password hash for accounts listed in the `user` table. The client can do this by using the `PASSWORD()` [1188] function to generate a password hash, or by using a password-generating statement (`CREATE USER`, `GRANT`, or `SET PASSWORD`).

In other words, the server *checks* hash values during authentication when a client first attempts to connect. The server *generates* hash values if a connected client invokes the `PASSWORD()` [1188] function or uses a password-generating statement to set or change a password.

Password hashing methods in MySQL have the history described following. These changes are illustrated by changes in the result from the `PASSWORD()` [1188] function that computes password hash values and in the structure of the `user` table where passwords are stored.

The Original (Pre-4.1) Hashing Method

The original hashing method produced a 16-byte string. Such hashes look like this:

```
mysql> SELECT PASSWORD( 'mypass' );
+-----+
| PASSWORD( 'mypass' ) |
+-----+
| 6f8c114b58f2ce9e   |
+-----+
```

To store account passwords, the `Password` column of the `user` table was at this point 16 bytes long.

The 4.1 Hashing Method

MySQL 4.1 introduced password hashing that provides better security and reduces the risk of passwords being intercepted. There were several aspects to this change:

- Different `PASSWORD()` [1188] function result format
- Widening of the `Password` column
- Control over the default hashing method
- Control over the permitted hashing methods for clients attempting to connect to the server

The changes in MySQL 4.1 took place in two stages:

- MySQL 4.1.0 used a preliminary version of the 4.1 hashing method. Because this method was so short lived, the following discussion says no more about it.
- In MySQL 4.1.1, the hashing method was modified to produce a longer 41-byte hash value:

```
mysql> SELECT PASSWORD( 'mypass' );
+-----+
| PASSWORD( 'mypass' ) |
+-----+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-----+
```

The longer password hash format has better cryptographic properties, and client authentication based on long hashes is more secure than that based on the older short hashes.

To accommodate longer password hashes, the `Password` column in the `user` table was changed at this point to be 41 bytes, its current length.

A widened `Password` column can store password hashes in both the pre-4.1 and 4.1 formats. The format of any given hash value can be determined two ways:

- The length: 4.1 and pre-4.1 hashes are 41 and 16 bytes, respectively.
- Password hashes in the 4.1 format always begin with a “*” character, whereas passwords in the pre-4.1 format never do.

To permit explicit generation of pre-4.1 password hashes, two additional changes were made:

- The `OLD_PASSWORD()` [1187] function was added, which returns hash values in the 16-byte format.

- For compatibility purposes, the `old_passwords` [534] system variable was added, to enable DBAs and applications control over the hashing method. The default `old_passwords` [534] value of 0 causes hashing to use the 4.1 method (41-byte hash values), but setting `old_passwords=1` [534] causes hashing to use the pre-4.1 method. In this case, `PASSWORD()` [1188] produces 16-byte values and is equivalent to `OLD_PASSWORD()` [1187]

To permit DBAs control over how clients are permitted to connect, the `secure_auth` [550] system variable was added. Starting the server with this variable disabled or enabled permits or prohibits clients to connect using the older pre-4.1 password hashing method. Before MySQL 5.6.5, `secure_auth` [550] is disabled by default. As of 5.6.5, `secure_auth` [550] is enabled by default to promote a more secure default configuration. (DBAs can disable it at their discretion, but this is not recommended.)

In addition, the `mysql` client supports a `--secure-auth` [281] option that is analogous to `secure_auth` [550], but from the client side. It can be used to prevent connections to less secure accounts that use pre-4.1 password hashing. This option is disabled by default before MySQL 5.6.7, enabled thereafter.

Compatibility Issues Related to Hashing Methods

The widening of the `Password` column in MySQL 4.1 from 16 bytes to 41 bytes affects installation or upgrade operations as follows:

- If you perform a new installation of MySQL, the `Password` column is made 41 bytes long automatically.
- Upgrades from MySQL 4.1 or later to current versions of MySQL should not give rise to any issues in regard to the `Password` column because both versions use the same column length and password hashing method.
- For upgrades from a pre-4.1 release to 4.1 or later, you must upgrade the system tables after upgrading. (See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).)

The 4.1 hashing method is understood only by MySQL 4.1 (and newer) servers and clients, which can result in some compatibility problems. A 4.1 or newer client can connect to a pre-4.1 server, because the client understands both the pre-4.1 and 4.1 password hashing methods. However, a pre-4.1 client that attempts to connect to a 4.1 or newer server may run into difficulties. For example, a 4.0 `mysql` client may fail with the following error message:

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

This phenomenon also occurs for attempts to use the older PHP `mysql` extension after upgrading to MySQL 4.1 or newer. (See [Section 22.9.12, “Common Problems with MySQL and PHP”](#).)

The following discussion describes the differences between the pre-4.1 and 4.1 hashing methods, and what you should do if you upgrade your server but need to maintain backward compatibility with pre-4.1 clients. (However, permitting connections by old clients is not recommended and should be avoided if possible.) Additional information can be found in [Section C.5.2.4, “Client does not support authentication protocol”](#). This information is of particular importance to PHP programmers migrating MySQL databases from versions older than 4.1 to 4.1 or higher.

The differences between short and long password hashes are relevant both for how the server uses passwords during authentication and for how it generates password hashes for connected clients that perform password-changing operations.

The way in which the server uses password hashes during authentication is affected by the width of the `Password` column:

- If the column is short, only short-hash authentication is used.
- If the column is long, it can hold either short or long hashes, and the server can use either format:
 - Pre-4.1 clients can connect, but because they know only about the pre-4.1 hashing method, they can authenticate only using accounts that have short hashes.
 - 4.1 and later clients can authenticate using accounts that have short or long hashes.

Even for short-hash accounts, the authentication process is actually a bit more secure for 4.1 and later clients than for older clients. In terms of security, the gradient from least to most secure is:

- Pre-4.1 client authenticating with short password hash
- 4.1 or later client authenticating with short password hash
- 4.1 or later client authenticating with long password hash

The way in which the server generates password hashes for connected clients is affected by the width of the `Password` column and by the `old_passwords` [534] system variable. A 4.1 or later server generates long hashes only if certain conditions are met: The `Password` column must be wide enough to hold long values and `old_passwords` [534] must not be set to 1.

Those conditions apply as follows:

- The `Password` column must be wide enough to hold long hashes (41 bytes). If the column has not been updated and still has the pre-4.1 width of 16 bytes, the server notices that long hashes cannot fit into it and generates only short hashes when a client performs password-changing operations using the `PASSWORD()` [1188] function or a password-generating statement. This is the behavior that occurs if you have upgraded from a version of MySQL older than 4.1 to 4.1 or later but have not yet run the `mysql_upgrade` program to widen the `Password` column.
- If the `Password` column is wide, it can store either short or long password hashes. In this case, the `PASSWORD()` [1188] function and password-generating statements generate long hashes unless the server was started with the `old_passwords` [534] system variable set to 1 to force the server to generate short password hashes instead.

The purpose of the `old_passwords` [534] system variable is to permit backward compatibility with pre-4.1 clients under circumstances where the server would otherwise generate long password hashes. The option does not affect authentication (4.1 and later clients can still use accounts that have long password hashes), but it does prevent creation of a long password hash in the `user` table as the result of a password-changing operation. Were that permitted to occur, the account could no longer be used by pre-4.1 clients. With `old_passwords` [534] disabled, the following undesirable scenario is possible:

- An old pre-4.1 client connects to an account that has a short password hash.
- The client changes its own password. With `old_passwords` [534] disabled, this results in the account having a long password hash.
- The next time the old client attempts to connect to the account, it cannot, because the account has a long password hash that requires the 4.1 hashing method during authentication. (Once an account has a long password hash in the `user` table, only 4.1 and later clients can authenticate for it because pre-4.1 clients do not understand long hashes.)

This scenario illustrates that, if you must support older pre-4.1 clients, it is problematic to run a 4.1 or newer server without `old_passwords` [534] set to 1. By running the server with `old_passwords=1` [534], password-changing operations do not generate long password hashes and thus

do not cause accounts to become inaccessible to older clients. (Those clients cannot inadvertently lock themselves out by changing their password and ending up with a long password hash.)

The downside of `old_passwords=1` [534] is that any passwords created or changed use short hashes, even for 4.1 or later clients. Thus, you lose the additional security provided by long password hashes. To create an account that has a long hash (for example, for use by 4.1 clients) or to change an existing account to use a long password hash, an administrator can set the session value of `old_passwords` [534] set to 0 while leaving the global value set to 1:

```
mysql> SET @@session.old_passwords = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@session.old_passwords, @@global.old_passwords;
+-----+-----+
| @@session.old_passwords | @@global.old_passwords |
+-----+-----+
| 0 | 1 |
+-----+-----+
1 row in set (0.00 sec)

mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'newpass';
Query OK, 0 rows affected (0.03 sec)

mysql> SET PASSWORD FOR 'existinguser'@'localhost' = PASSWORD('existingpass');
Query OK, 0 rows affected (0.00 sec)
```

The following scenarios are possible in MySQL 4.1 or later. The factors are whether the `Password` column is short or long, and, if long, whether the server is started with `old_passwords` [534] enabled or disabled.

Scenario 1: Short `Password` column in user table:

- Only short hashes can be stored in the `Password` column.
- The server uses only short hashes during client authentication.
- For connected clients, password hash-generating operations involving the `PASSWORD()` [1188] function or password-generating statements use short hashes exclusively. Any change to an account's password results in that account having a short password hash.
- The value of `old_passwords` [534] is irrelevant because with a short `Password` column, the server generates only short password hashes anyway.

This scenario occurs when a pre-4.1 MySQL installation has been upgraded to 4.1 or later but `mysql_upgrade` has not been run to upgrade the system tables in the `mysql` database. (This is not a recommended configuration because it does not permit use of more secure 4.1 password hashing.)

Scenario 2: Long `Password` column; server started with `old_passwords=1` [534]:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate for accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving the `PASSWORD()` [1188] function or password-generating statements use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

In this scenario, newly created accounts have short password hashes because `old_passwords=1` [534] prevents generation of long hashes. Also, if you create an account with a long hash before setting

`old_passwords` [534] to 1, changing the account's password while `old_passwords=1` [534] results in the account being given a short password, causing it to lose the security benefits of a longer hash.

To create a new account that has a long password hash, or to change the password of any existing account to use a long hash, first set the session value of `old_passwords` [534] set to 0 while leaving the global value set to 1, as described previously.

In this scenario, the server has an up to date `Password` column, but is running with the default password hashing method set to generate pre-4.1 hash values. This is not a recommended configuration but may be useful during a transitional period in which pre-4.1 clients and passwords are upgraded to 4.1 or later. When that has been done, it is preferable to run the server with `old_passwords=0` [534] and `secure_auth=1` [550].

Scenario 3: Long `Password` column; server started with `old_passwords=0` [534]:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate using accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only using accounts that have short hashes.
- For connected clients, password hash-generating operations involving the `PASSWORD()` [1188] function or password-generating statements use short hashes exclusively. A change to an account's password results in that account having a long password hash.

As indicated earlier, a danger in this scenario is that it is possible for accounts that have a short password hash to become inaccessible to pre-4.1 clients. A change to such an account's password made using the `PASSWORD()` [1188] function or a password-generating statement results in the account being given a long password hash. From that point on, no pre-4.1 client can connect to the server using that account. The client must upgrade to 4.1 or later.

If this is a problem, you can change a password in a special way. For example, normally you use `SET PASSWORD` as follows to change an account password:

```
SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

To change the password but create a short hash, use the `OLD_PASSWORD()` [1187] function instead:

```
SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

`OLD_PASSWORD()` [1187] is useful for situations in which you explicitly want to generate a short hash.

The disadvantages for each of the preceding scenarios may be summarized as follows:

In scenario 1, you cannot take advantage of longer hashes that provide more secure authentication.

In scenario 2, `old_passwords=1` [534] prevents accounts with short hashes from becoming inaccessible, but password-changing operations cause accounts with long hashes to revert to short hashes unless you take care to change the session value of `old_passwords` [534] to 0 first.

In scenario 3, accounts with short hashes become inaccessible to pre-4.1 clients if you change their passwords without explicitly using `OLD_PASSWORD()` [1187].

The best way to avoid compatibility problems related to short password hashes is to not use them:

- Upgrade all client programs to MySQL 4.1 or later.
- Run the server with `old_passwords=0` [534].

- Reset the password for any account with a short password hash to use a long password hash.
- For additional security, run the server with `secure_auth=1` [550].

6.1.2.5. Implications of Password Hashing Changes in MySQL 4.1 for Application Programs

An upgrade to MySQL version 4.1 or later can cause compatibility issues for applications that use `PASSWORD()` [1188] to generate passwords for their own purposes. Applications really should not do this, because `PASSWORD()` [1188] should be used only to manage passwords for MySQL accounts. But some applications use `PASSWORD()` [1188] for their own purposes anyway.

If you upgrade to 4.1 or later from a pre-4.1 version of MySQL and run the server under conditions where it generates long password hashes, an application using `PASSWORD()` [1188] for its own passwords breaks. The recommended course of action in such cases is to modify the application to use another function, such as `SHA2()` [1189], `SHA1()` [1189], or `MD5()` [1187], to produce hashed values. If that is not possible, you can use the `OLD_PASSWORD()` [1187] function, which is provided for generate short hashes in the old format. However, you should note that `OLD_PASSWORD()` [1187] may one day no longer be supported.

If the server is running with `old_passwords=1` [534], it generates short hashes and `OLD_PASSWORD()` [1187] is equivalent to `PASSWORD()` [1188].

PHP programmers migrating their MySQL databases from version 4.0 or lower to version 4.1 or higher should see [Section 22.9, “MySQL PHP API”](#).

6.1.3. Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted in clear text over the connection. Password handling during the client connection sequence was upgraded in MySQL 4.1.1 to be very secure. If you are still using pre-4.1.1-style passwords, the encryption algorithm is not as strong as the newer algorithm. With some effort, a clever attacker who can sniff the traffic between the client and the server can crack the password. (See [Section 6.1.2.4, “Password Hashing in MySQL”](#), for a discussion of the different password handling methods.)

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure. See [Section 6.3.8, “Using SSL for Secure Connections”](#). Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at <http://www.openssh.org/>, and a commercial SSH client at <http://www.ssh.com/>.

To make a MySQL system secure, you should strongly consider the following suggestions:

- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any user name to the client program. For example, anyone can use the `mysql` program to connect as any other person simply by invoking it as `mysql -u other_user db_name` if `other_user` has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

For a discussion of methods for setting passwords, see [Section 6.3.5, “Assigning Account Passwords”](#).

- Make sure that the only Unix user account with read or write privileges in the database directories is the account that is used for running `mysqld`.
- Never run the MySQL server as the Unix `root` user. This is extremely dangerous, because any user with the `FILE` [696] privilege is able to cause the server to create files as `root` (for example,

`~root/.bashrc`). To prevent this, `mysqld` refuses to run as `root` unless that is specified explicitly using the `--user=root` [466] option.

`mysqld` can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start `mysqld` as a different Unix user, add a `user` option that specifies the user name in the `[mysqld]` group of the `my.cnf` option file where you specify server options. For example:

```
[mysqld]
user=mysql
```

This causes the server to start as the designated user whether you start it manually or by using `mysqld_safe` or `mysql.server`. For more details, see [Section 6.1.5, “How to Run MySQL as a Normal User”](#).

Running `mysqld` as a Unix user other than `root` does not mean that you need to change the `root` user name in the `user` table. *User names for MySQL accounts have nothing to do with user names for Unix accounts.*

- Do not grant the `FILE` [696] privilege to nonadministrative users. Any user that has this privilege can write a file anywhere in the file system with the privileges of the `mysqld` daemon. This includes the server's data directory containing the files that implement the privilege tables. To make `FILE` [696]-privilege operations a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

The `FILE` [696] privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

To limit the location in which files can be read and written, set the `secure_file_priv` [550] system to a specific directory. See [Section 5.1.4, “Server System Variables”](#).

- Do not grant the `PROCESS` [696] or `SUPER` [697] privilege to nonadministrative users. The output of `mysqladmin processlist` and `SHOW PROCESSLIST` shows the text of any statements currently being executed, so any user who is permitted to see the server process list might be able to see statements issued by other users such as `UPDATE user SET password=PASSWORD('not_secure')`.

`mysqld` reserves an extra connection for users who have the `SUPER` [697] privilege, so that a MySQL `root` user can log in and check server activity even if all normal connections are in use.

The `SUPER` [697] privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not permit the use of symlinks to tables. (This capability can be disabled with the `--skip-symbolic-links` [462] option.) This is especially important if you run `mysqld` as `root`, because anyone that has write access to the server's data directory then could delete any file in the system! See [Section 8.11.3.1.2, “Using Symbolic Links for MyISAM Tables on Unix”](#).
- Stored programs and views should be written using the security guidelines discussed in [Section 19.6, “Access Control for Stored Programs and Views”](#).
- If you do not trust your DNS, you should use IP addresses rather than host names in the grant tables. In any case, you should be very careful about creating grant table entries using host name values that contain wildcards.

- If you want to restrict the number of connections permitted to a single account, you can do so by setting the `max_user_connections` [525] variable in `mysqld`. The `GRANT` statement also supports resource control options for limiting the extent of server use permitted to an account. See [Section 13.7.1.3, “GRANT Syntax”](#).
- If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` [537] read only to the server or by setting `--secure-file-priv` [460] to a directory where `SELECT` writes can be made safely.

6.1.4. Security-Related `mysqld` Options and Variables

The following table shows `mysqld` options and system variables that affect security. For descriptions of each of these, see [Section 5.1.3, “Server Command Options”](#), and [Section 5.1.4, “Server System Variables”](#).

Table 6.1. Security Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
allow-suspicious-udfs [434]	Yes	Yes				
automatic_sp_privileges [484]			Yes		Global	Yes
chroot [438]	Yes	Yes				
des-key-file [442]	Yes	Yes				
local_infile [513]			Yes		Global	Yes
old_passwords [534]			Yes		Both	Yes
safe-show-database [459]	Yes	Yes	Yes		Global	Yes
safe-user-create [459]	Yes	Yes				
secure-auth [460]	Yes	Yes			Global	Yes
- Variable: <code>secure_auth</code>			Yes		Global	Yes
secure-file-priv [460]	Yes	Yes			Global	No
- Variable: <code>secure_file_priv</code>			Yes		Global	No
skip-grant-tables [461]	Yes	Yes				
skip-name-resolve [461]	Yes	Yes			Global	No
- Variable: <code>skip_name_resolve</code>			Yes		Global	No
skip-networking [461]	Yes	Yes			Global	No
- Variable: <code>skip_networking</code>			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
skip-show-database [463]	Yes	Yes			Global	No
- Variable: skip_show_database			Yes		Global	No

6.1.5. How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account.

On Unix, the MySQL server `mysqld` can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. To change `mysqld` to run as a normal unprivileged Unix user `user_name`, you must do the following:

1. Stop the server if it is running (use `mysqladmin shutdown`).
2. Change the database directories and files so that `user_name` has privileges to read and write files in them (you might need to do this as the Unix `root` user):

```
shell> chown -R user_name /path/to/mysql/datadir
```

If you do not do this, the server will not be able to access databases or tables when it runs as `user_name`.

If directories or files within the MySQL data directory are symbolic links, `chown -R` might not follow symbolic links for you. If it does not, you will also need to follow those links and change the directories and files they point to.

3. Start the server as user `user_name`. Another alternative is to start `mysqld` as the Unix `root` user and use the `--user=user_name [466]` option. `mysqld` starts up, then switches to run as the Unix user `user_name` before accepting any connections.
4. To start the server as the given user automatically at system startup time, specify the user name by adding a `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

```
[mysqld]
user=user_name
```

If your Unix machine itself is not secured, you should assign passwords to the MySQL `root` accounts in the grant tables. Otherwise, any user with a login account on that machine can run the `mysql` client with a `--user=root [282]` option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See [Section 2.11, "Postinstallation Setup and Testing"](#).

6.1.6. Security Issues with `LOAD DATA LOCAL`

The `LOAD DATA` statement can load a file that is located on the server host, or it can load a file that is located on the client host when the `LOCAL` keyword is specified.

There are two potential security issues with supporting the `LOCAL` version of `LOAD DATA` statements:

- The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD DATA` statement. Such a server could access any file on the client host to which the client user has read access.

- In a Web environment where the clients are connecting from a Web server, a user could use `LOAD DATA LOCAL` to read any files that the Web server process has read access to (assuming that a user could run any command against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not the remote program being run by the user who connects to the Web server.

To deal with these problems, we changed how `LOAD DATA LOCAL` is handled as of MySQL 3.23.49 and MySQL 4.0.2 (4.0.13 on Windows):

- By default, all MySQL clients and libraries in binary distributions are compiled with the `-DENABLED_LOCAL_INFILE=1` [154] option, to be compatible with MySQL 3.23.48 and before.
- If you build MySQL from source but do not invoke `CMake` with the `-DENABLED_LOCAL_INFILE=1` [154] option, `LOAD DATA LOCAL` cannot be used by any client unless it is written explicitly to invoke `mysql_options(... MYSQL_OPT_LOCAL_INFILE, 0)`. See [Section 22.8.3.49](#), “`mysql_options()`”.
- You can disable all `LOAD DATA LOCAL` statements from the server side by starting `mysqld` with the `--local-infile=0` [513] option.
- For the `mysql` command-line client, enable `LOAD DATA LOCAL` by specifying the `--local-infile[=1]` [278] option, or disable it with the `--local-infile=0` [278] option. For `mysqlimport`, local data file loading is off by default; enable it with the `--local` [328] or `-L` option. In any case, successful use of a local load operation requires that the server permits it.
- If you use `LOAD DATA LOCAL` in Perl scripts or other programs that read the `[client]` group from option files, you can add the `local-infile=1` option to that group. However, to keep this from causing problems for programs that do not understand `local-infile`, specify it using the `loose-` prefix:

```
[client]
loose-local-infile=1
```

- If `LOAD DATA LOCAL` is disabled, either in the server or the client, a client that attempts to issue such a statement receives the following error message:

```
ERROR 1148: The used command is not allowed with this MySQL version
```

6.1.7. Client Programming Security Guidelines

Applications that access MySQL should not trust any data entered by users, who can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like “`;
DROP DATABASE mysql;`”. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value 234, the user can enter the value `234 OR 1=1` to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotation marks around the numeric constants: `SELECT * FROM table WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing nonnumeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is permissible to display any row in the database, you should still protect against

denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

- Enable strict SQL mode to tell the server to be more restrictive of what data values it accepts. See [Section 5.1.7, “Server SQL Modes”](#).
- Try to enter single and double quotation marks (“'” and “””) in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.
- Try to modify dynamic URLs by adding %22 (“”), %23 (“#”), and %27 (“'”) to them.
- Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.
- Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!
- Check the size of data before passing it to MySQL.
- Have your application connect to the database using a user name different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:

- MySQL C API: Use the `mysql_real_escape_string()` API call.
- MySQL++: Use the `escape` and `quote` modifiers for query streams.
- PHP: Use either the `mysqli` or `pdo_mysql` extensions, and not the older `ext/mysql` extension. The preferred API's support the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders. See also [Section 22.9.1.3, “Choosing an API”](#).

If the older `ext/mysql` extension must be used, then for escaping use the `mysql_real_escape_string()` function and not `mysql_escape_string()` or `addslashes()` because only `mysql_real_escape_string()` is character set-aware; the other functions can be “bypassed” when using (invalid) multi-byte character sets.

- Perl DBI: Use placeholders or the `quote()` method.
- Ruby DBI: Use placeholders or the `quote()` method.
- Java JDBC: Use a `PreparedStatement` object and placeholders.

Other programming interfaces might have similar capabilities.

6.2. The MySQL Access Privilege System

The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Additional functionality includes the ability to have anonymous users and to grant privileges for MySQL-specific functions such as `LOAD DATA INFILE` and administrative operations.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

The user interface to the MySQL privilege system consists of SQL statements such as `CREATE USER`, `GRANT`, and `REVOKE`. See [Section 13.7.1, “Account Management Statements”](#).

Internally, the server stores privilege information in the grant tables of the `mysql` database (that is, in the database named `mysql`). The MySQL server reads the contents of these tables into memory when it starts and bases access-control decisions on the in-memory copies of the grant tables.

The MySQL privilege system ensures that all users may perform only the operations permitted to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the user name you specify*. When you issue requests after connecting, the system grants privileges according to your identity and *what you want to do*.

MySQL considers both your host name and user name in identifying you because there is no reason to assume that a given user name belongs to the same person on all hosts. For example, the user `joe` who connects from `office.example.com` need not be the same person as the user `joe` who connects from `home.example.com`. MySQL handles this by enabling you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by `joe` from `office.example.com`, and a different set of privileges for connections by `joe` from `home.example.com`. To see what privileges a given account has, use the `SHOW GRANTS` statement. For example:

```
SHOW GRANTS FOR 'joe'@'office.example.com';  
SHOW GRANTS FOR 'joe'@'home.example.com';
```

MySQL access control involves two stages when you run a client program that connects to the server:

Stage 1: The server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password.

Stage 2: Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the `SELECT [697]` privilege for the table or the `DROP [696]` privilege for the database.

For a more detailed description of what happens during each stage, see [Section 6.2.4, “Access Control, Stage 1: Connection Verification”](#), and [Section 6.2.5, “Access Control, Stage 2: Request Verification”](#).

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. For details about the conditions under which the server reloads the grant tables, see [Section 6.2.6, “When Privilege Changes Take Effect”](#).

For general security-related advice, see [Section 6.1, “General Security Issues”](#). For help in diagnosing privilege-related problems, see [Section 6.2.7, “Causes of Access-Denied Errors”](#).

6.2.1. Privileges Provided by MySQL

MySQL provides privileges that apply in different contexts and at different levels of operation:

- Administrative privileges enable users to manage operation of the MySQL server. These privileges are global because they are not specific to a particular database.
- Database privileges apply to a database and to all objects within it. These privileges can be granted for specific databases, or globally so that they apply to all databases.
- Privileges for database objects such as tables, indexes, views, and stored routines can be granted for specific objects within a database, for all objects of a given type within a database (for example, all tables in a database), or globally for all objects of a given type in all databases).

Information about account privileges is stored in the `user`, `db`, `host`, `tables_priv`, `columns_priv`, and `procs_priv` tables in the `mysql` database (see [Section 6.2.2, “Privilege System Grant Tables”](#)). The MySQL server reads the contents of these tables into memory when it starts and reloads them under the circumstances indicated in [Section 6.2.6, “When Privilege Changes Take Effect”](#). Access-control decisions are based on the in-memory copies of the grant tables.

Some releases of MySQL introduce changes to the structure of the grant tables to add new access privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

The following table shows the privilege names used at the SQL level in the `GRANT` and `REVOKE` statements, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies.

Table 6.2. Permissible Privileges for `GRANT` and `REVOKE`

Privilege	Column	Context
<code>CREATE</code> [695]	<code>Create_priv</code>	databases, tables, or indexes
<code>DROP</code> [696]	<code>Drop_priv</code>	databases, tables, or views
<code>GRANT OPTION</code> [696]	<code>Grant_priv</code>	databases, tables, or stored routines
<code>LOCK TABLES</code> [696]	<code>Lock_tables_priv</code>	databases
<code>REFERENCES</code> [696]	<code>References_priv</code>	databases or tables
<code>EVENT</code> [696]	<code>Event_priv</code>	databases
<code>ALTER</code> [695]	<code>Alter_priv</code>	tables
<code>DELETE</code> [696]	<code>Delete_priv</code>	tables
<code>INDEX</code> [696]	<code>Index_priv</code>	tables
<code>INSERT</code> [696]	<code>Insert_priv</code>	tables or columns
<code>SELECT</code> [697]	<code>Select_priv</code>	tables or columns
<code>UPDATE</code> [697]	<code>Update_priv</code>	tables or columns
<code>CREATE TEMPORARY TABLES</code> [695]	<code>Create_tmp_table_priv</code>	tables
<code>TRIGGER</code> [697]	<code>Trigger_priv</code>	tables
<code>CREATE VIEW</code> [696]	<code>Create_view_priv</code>	views
<code>SHOW VIEW</code> [697]	<code>Show_view_priv</code>	views
<code>ALTER ROUTINE</code> [695]	<code>Alter_routine_priv</code>	stored routines
<code>CREATE ROUTINE</code> [695]	<code>Create_routine_priv</code>	stored routines

Privilege	Column	Context
EXECUTE [696]	Execute_priv	stored routines
FILE [696]	File_priv	file access on server host
CREATE TABLESPACE [695]	Create_tablespace_priv	server administration
CREATE USER [696]	Create_user_priv	server administration
PROCESS [696]	Process_priv	server administration
PROXY [696]	see proxies_priv table	server administration
RELOAD [696]	Reload_priv	server administration
REPLICATION CLIENT [697]	Repl_client_priv	server administration
REPLICATION SLAVE [697]	Repl_slave_priv	server administration
SHOW DATABASES [697]	Show_db_priv	server administration
SHUTDOWN [697]	Shutdown_priv	server administration
SUPER [697]	Super_priv	server administration
ALL [PRIVILEGES] [695]		server administration
USAGE [697]		server administration

The following list provides a general description of each privilege available in MySQL. Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- The [ALL \[695\]](#) or [ALL PRIVILEGES \[695\]](#) privilege specifier is shorthand. It stands for “all privileges available at a given privilege level” (except [GRANT OPTION \[696\]](#)). For example, granting [ALL \[695\]](#) at the global or table level grants all global privileges or all table-level privileges.
- The [ALTER \[695\]](#) privilege enables use of [ALTER TABLE](#) to change the structure of tables. [ALTER TABLE](#) also requires the [CREATE \[695\]](#) and [INSERT \[696\]](#) privileges. Renaming a table requires [ALTER \[695\]](#) and [DROP \[696\]](#) on the old table, [ALTER \[695\]](#), [CREATE \[695\]](#), and [INSERT \[696\]](#) on the new table.
- The [ALTER ROUTINE \[695\]](#) privilege is needed to alter or drop stored routines (procedures and functions).
- The [CREATE \[695\]](#) privilege enables creation of new databases and tables.
- The [CREATE ROUTINE \[695\]](#) privilege is needed to create stored routines (procedures and functions).
- The [CREATE TABLESPACE \[695\]](#) privilege is needed to create, alter, or drop tablespaces and log file groups.
- The [CREATE TEMPORARY TABLES \[695\]](#) privilege enables the creation of temporary tables using the [CREATE TEMPORARY TABLE](#) statement.

However, other operations on a temporary table, such as [INSERT](#), [UPDATE](#), or [SELECT](#), require additional privileges for those operations for the database containing the temporary table, or for the nontemporary table of the same name.

To keep privileges for temporary and nontemporary tables separate, a common workaround for this situation is to create a database dedicated to the use of temporary tables. Then for that database, a user can be granted the [CREATE TEMPORARY TABLES \[695\]](#) privilege, along with any other privileges required for temporary table operations done by that user.

- The `CREATE USER` [696] privilege enables use of `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES`.
- The `CREATE VIEW` [696] privilege enables use of `CREATE VIEW`.
- The `DELETE` [696] privilege enables rows to be deleted from tables in a database.
- The `DROP` [696] privilege enables you to drop (remove) existing databases, tables, and views. The `DROP` [696] privilege is required in order to use the statement `ALTER TABLE ... DROP PARTITION` on a partitioned table. The `DROP` [696] privilege is also required for `TRUNCATE TABLE`. *If you grant the `DROP` [696] privilege for the `mysql` database to a user, that user can drop the database in which the MySQL access privileges are stored.*
- The `EVENT` [696] privilege is required to create, alter, drop, or see events for the Event Scheduler.
- The `EXECUTE` [696] privilege is required to execute stored routines (procedures and functions).
- The `FILE` [696] privilege gives you permission to read and write files on the server host using the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` [1105] function. A user who has the `FILE` [696] privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.) The `FILE` [696] privilege also enables the user to create new files in any directory where the MySQL server has write access. This includes the server's data directory containing the files that implement the privilege tables. As a security measure, the server will not overwrite existing files.

To limit the location in which files can be read and written, set the `secure_file_priv` [550] system to a specific directory. See [Section 5.1.4, “Server System Variables”](#).

- The `GRANT OPTION` [696] privilege enables you to give to other users or remove from other users those privileges that you yourself possess.
- The `INDEX` [696] privilege enables you to create or drop (remove) indexes. `INDEX` [696] applies to existing tables. If you have the `CREATE` [695] privilege for a table, you can include index definitions in the `CREATE TABLE` statement.
- The `INSERT` [696] privilege enables rows to be inserted into tables in a database. `INSERT` [696] is also required for the `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` table-maintenance statements.
- The `LOCK TABLES` [696] privilege enables the use of explicit `LOCK TABLES` statements to lock tables for which you have the `SELECT` [697] privilege. This includes the use of write locks, which prevents other sessions from reading the locked table.
- The `PROCESS` [696] privilege pertains to display of information about the threads executing within the server (that is, information about the statements being executed by sessions). The privilege enables use of `SHOW PROCESSLIST` or `mysqladmin processlist` to see threads belonging to other accounts; you can always see your own threads.
- The `PROXY` [696] privilege enables a user to impersonate or become known as another user. See [Section 6.3.7, “Proxy Users”](#). This privilege was added in MySQL 5.5.7.
- The `REFERENCES` [696] privilege currently is unused.
- The `RELOAD` [696] privilege enables use of the `FLUSH` statement. It also enables `mysqladmin` commands that are equivalent to `FLUSH` operations: `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, `flush-tables`, `flush-threads`, `refresh`, and `reload`.

The `reload` command tells the server to reload the grant tables into memory. `flush-privileges` is a synonym for `reload`. The `refresh` command closes and reopens the log files and flushes all tables. The other `flush-xxx` commands perform functions similar to `refresh`, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, `flush-logs` is a better choice than `refresh`.

- The `REPLICATION CLIENT` [697] privilege enables the use of `SHOW MASTER STATUS` and `SHOW SLAVE STATUS`. In MySQL 5.5.25 and later, it also enables the use of the `SHOW BINARY LOGS` statement.
- The `REPLICATION SLAVE` [697] privilege should be granted to accounts that are used by slave servers to connect to the current server as their master. Without this privilege, the slave cannot request updates that have been made to databases on the master server.
- The `SELECT` [697] privilege enables you to select rows from tables in a database. `SELECT` statements require the `SELECT` [697] privilege only if they actually retrieve rows from a table. Some `SELECT` statements do not access tables and can be executed without permission for any database. For example, you can use `SELECT` as a simple calculator to evaluate expressions that make no reference to tables:

```
SELECT 1+1;  
SELECT PI()*2;
```

The `SELECT` [697] privilege is also needed for other statements that read column values. For example, `SELECT` [697] is needed for columns referenced on the right hand side of `col_name=expr` assignment in `UPDATE` statements or for columns named in the `WHERE` clause of `DELETE` or `UPDATE` statements.

- The `SHOW DATABASES` [697] privilege enables the account to see database names by issuing the `SHOW DATABASE` statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the `--skip-show-database` [463] option. Note that *any* global privilege is a privilege for the database.
- The `SHOW VIEW` [697] privilege enables use of `SHOW CREATE VIEW`.
- The `SHUTDOWN` [697] privilege enables use of the `mysqladmin shutdown` command. There is no corresponding SQL statement.
- The `SUPER` [697] privilege enables an account to use `CHANGE MASTER TO`, `KILL` or `mysqladmin kill` to kill threads belonging to other accounts (you can always kill your own threads), `PURGE BINARY LOGS`, configuration changes using `SET GLOBAL` to modify global system variables, the `mysqladmin debug` command, enabling or disabling logging, performing updates even if the `read_only` [544] system variable is enabled, starting and stopping replication on slave servers, specification of any account in the `DEFINER` attribute of stored programs and views, and enables you to connect (once) even if the connection limit controlled by the `max_connections` [519] system variable is reached.

To create or alter stored functions if binary logging is enabled, you may also need the `SUPER` [697] privilege, as described in [Section 19.7, “Binary Logging of Stored Programs”](#).

- The `TRIGGER` [697] privilege enables trigger operations. You must have this privilege for a table to create, drop, or execute triggers for that table.
- The `UPDATE` [697] privilege enables rows to be updated in tables in a database.
- The `USAGE` [697] privilege specifier stands for “no privileges.” It is used at the global level with `GRANT` to modify account attributes such as resource limits or SSL characteristics without affecting existing account privileges.

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the [FILE \[696\]](#) and administrative privileges:

- The [FILE \[696\]](#) privilege can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using [SELECT](#) to transfer its contents to the client host.
- The [GRANT OPTION \[696\]](#) privilege enables users to give their privileges to other users. Two users that have different privileges and with the [GRANT OPTION \[696\]](#) privilege are able to combine privileges.
- The [ALTER \[695\]](#) privilege may be used to subvert the privilege system by renaming tables.
- The [SHUTDOWN \[697\]](#) privilege can be abused to deny service to other users entirely by terminating the server.
- The [PROCESS \[696\]](#) privilege can be used to view the plain text of currently executing statements, including statements that set or change passwords.
- The [SUPER \[697\]](#) privilege can be used to terminate other sessions or change how the server operates.
- Privileges granted for the `mysql` database itself can be used to change passwords and other access privilege information. Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the `user` table `Password` column can change an account's password, and then connect to the MySQL server using that account.

6.2.2. Privilege System Grant Tables

Normally, you manipulate the contents of the grant tables in the `mysql` database indirectly by using statements such as [GRANT](#) and [REVOKE](#) to set up accounts and control the privileges available to each one. See [Section 13.7.1, "Account Management Statements"](#). The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients.

These `mysql` database tables contain grant information:

- `user`: Contains user accounts, global privileges, and other non-privilege columns.
- `db`: Contains database-level privileges.
- `host`: Obsolete.
- `tables_priv`: Contains table-level privileges.
- `columns_priv`: Contains column-level privileges.
- `procs_priv`: Contains stored procedure and function privileges.
- `proxies_priv`: Contains proxy-user privileges.

Other tables in the `mysql` database do not hold grant information and are discussed elsewhere:

- `event`: Contains information about Event Scheduler events: See [Section 19.4, "Using the Event Scheduler"](#).
- `func`: Contains information about user-defined functions: See [Section 23.3, "Adding New Functions to MySQL"](#).
- `help_xxx`: These tables are used for server-side help: See [Section 5.1.10, "Server-Side Help"](#).

- `plugin`: Contains information about server plugins: See [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#), and [Section 23.2, “The MySQL Plugin API”](#).
- `proc`: Contains information about stored procedures and functions: See [Section 19.2, “Using Stored Routines \(Procedures and Functions\)”](#).
- `servers`: Used by the `FEDERATED` storage engine: See [Section 14.11.2.2, “Creating a FEDERATED Table Using CREATE SERVER”](#).
- `time_zone_xxx`: These tables contain time zone information: See [Section 10.6, “MySQL Server Time Zone Support”](#).
- Tables with `_log` in their name are used for logging: See [Section 5.2, “MySQL Server Logs”](#).

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row (entry) in the tables; that is, the context in which the row applies. For example, a `user` table row with `Host` and `User` values of `'thomas.loc.gov'` and `'bob'` would be used for authenticating connections made to the server from the host `thomas.loc.gov` by a client that specifies a user name of `bob`. Similarly, a `db` table row with `Host`, `User`, and `Db` column values of `'thomas.loc.gov'`, `'bob'` and `'reports'` would be used when `bob` connects from the host `thomas.loc.gov` to access the `reports` database. The `tables_priv` and `columns_priv` tables contain scope columns indicating tables or table/column combinations to which each row applies. The `procs_priv` scope columns indicate the stored routine to which each row applies.
- Privilege columns indicate which privileges are granted by a table row; that is, what operations can be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. [Section 6.2.5, “Access Control, Stage 2: Request Verification”](#), describes the rules that are used to do this.

The server uses the grant tables in the following manner:

- The `user` table scope columns determine whether to reject or permit incoming connections. For permitted connections, any privileges granted in the `user` table indicate the user's global privileges. Any privilege granted in this table applies to *all* databases on the server.



Note

Because any global privilege is considered a privilege for all databases, any global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`.

- The `db` table scope columns determine which users can access which databases from which hosts. The privilege columns determine which operations are permitted. A privilege granted at the database level applies to the database and to all objects in the database, such as tables and stored programs.
- The `host` table is used in conjunction with the `db` table when you want a given `db` table row to apply to several hosts. For example, if you want a user to be able to use a database from several hosts in your network, leave the `Host` value empty in the user's `db` table row, then populate the `host` table with a row for each of those hosts. This mechanism is described more detail in [Section 6.2.5, “Access Control, Stage 2: Request Verification”](#).



Note

The `host` table must be modified directly with statements such as `INSERT`, `UPDATE`, and `DELETE`. It is not affected by statements such as `GRANT` and

REVOKE that modify the grant tables indirectly. Most MySQL installations need not use this table at all.

- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.
- The `procs_priv` table applies to stored routines. A privilege granted at the routine level applies only to a single routine.
- The `proxies_priv` table indicates which users can act as proxies for other users and whether proxy users can grant the `PROXY [696]` privilege to other users.

The server uses the `user`, `db`, and `host` tables in the `mysql` database at both the first and second stages of access control (see [Section 6.2, “The MySQL Access Privilege System”](#)). The columns in the `user` and `db` tables are shown here. The `host` table is similar to the `db` table but has a specialized use as described in [Section 6.2.5, “Access Control, Stage 2: Request Verification”](#).

Table 6.3. `user` and `db` Table Columns

Table Name	<code>user</code>	<code>db</code>
Scope columns	Host	Host
	User	Db
	Password	User
Privilege columns	Select_priv	Select_priv
	Insert_priv	Insert_priv
	Update_priv	Update_priv
	Delete_priv	Delete_priv
	Index_priv	Index_priv
	Alter_priv	Alter_priv
	Create_priv	Create_priv
	Drop_priv	Drop_priv
	Grant_priv	Grant_priv
	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv
	Alter_routine_priv	Alter_routine_priv
	Execute_priv	Execute_priv
	Trigger_priv	Trigger_priv
	Event_priv	Event_priv
	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv
	References_priv	References_priv
	Reload_priv	
	Shutdown_priv	

Table Name	user	db
	Process_priv	
	File_priv	
	Show_db_priv	
	Super_priv	
	Repl_slave_priv	
	Repl_client_priv	
	Create_user_priv	
	Create_tablespace_priv	
Security columns	ssl_type	
	ssl_cipher	
	x509_issuer	
	x509_subject	
	plugin	
	authentication_string	
Resource control columns	max_questions	
	max_updates	
	max_connections	
	max_user_connections	

As of MySQL 5.5.7, the `mysql.user` table has `plugin` and `authentication_string` columns for storing authentication plugin information.

If the `plugin` column for an account row is empty, the server uses native authentication for connection attempts for the account: Clients must match the password in the `Password` column of the account row.

If an account row names a plugin in the `plugin` column, the server uses it to authenticate connection attempts for the account. Whether the plugin uses the value in the `Password` column is up to the plugin.

Prior to MySQL 5.5.11, the length of the `plugin` column was 60 characters. This was increased to 64 characters in MySQL 5.5.11 for compatibility with the `mysql.plugin` table's `name` column. (Bug #11766610, Bug #59752)

During the second stage of access control, the server performs request verification to make sure that each client has sufficient privileges for each request that it issues. In addition to the `user`, `db`, and `host` grant tables, the server may also consult the `tables_priv` and `columns_priv` tables for requests that involve tables. The latter tables provide finer privilege control at the table and column levels. They have the columns shown in the following table.

Table 6.4. `tables_priv` and `columns_priv` Table Columns

Table Name	tables_priv	columns_priv
Scope columns	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name

Table Name	<code>tables_priv</code>	<code>columns_priv</code>
Privilege columns	<code>Table_priv</code>	<code>Column_priv</code>
	<code>Column_priv</code>	
Other columns	<code>Timestamp</code>	<code>Timestamp</code>
	<code>Grantor</code>	

The `Timestamp` and `Grantor` columns currently are unused and are discussed no further here.

For verification of requests that involve stored routines, the server may consult the `procs_priv` table, which has the columns shown in the following table.

Table 6.5. `procs_priv` Table Columns

Table Name	<code>procs_priv</code>
Scope columns	<code>Host</code>
	<code>Db</code>
	<code>User</code>
	<code>Routine_name</code>
	<code>Routine_type</code>
Privilege columns	<code>Proc_priv</code>
Other columns	<code>Timestamp</code>
	<code>Grantor</code>

The `Routine_type` column is an `ENUM` column with values of `'FUNCTION'` or `'PROCEDURE'` to indicate the type of routine the row refers to. This column enables privileges to be granted separately for a function and a procedure with the same name.

The `Timestamp` and `Grantor` columns currently are unused and are discussed no further here.

The `proxies_priv` table was added in MySQL 5.5.7 and records information about proxy users. It has these columns:

- `Host`, `User`: These columns indicate the user account that has the `PROXY [696]` privilege for the proxied account.
- `Proxied_host`, `Proxied_user`: These columns indicate the account of the proxied user.
- `Grantor`: Currently unused.
- `Timestamp`: Currently unused.
- `With_grant`: This column indicates whether the proxy account can grant the `PROXY [696]` privilege to other accounts.

Scope columns in the grant tables contain strings. They are declared as shown here; the default value for each is the empty string.

Table 6.6. Grant Table Scope Column Types

Column Name	Type
<code>Host</code> , <code>Proxied_host</code>	<code>CHAR(60)</code>
<code>User</code> , <code>Proxied_user</code>	<code>CHAR(16)</code>
<code>Password</code>	<code>CHAR(41)</code>

Column Name	Type
Db	CHAR(64)
Table_name	CHAR(64)
Column_name	CHAR(64)
Routine_name	CHAR(64)

For access-checking purposes, comparisons of `User`, `Proxied_user`, `Password`, `Db`, and `Table_name` values are case sensitive. Comparisons of `Host`, `Proxied_host`, `Column_name`, and `Routine_name` values are not case sensitive.

In the `user`, `db`, and `host` tables, each privilege is listed in a separate column that is declared as `ENUM('N','Y') DEFAULT 'N'`. In other words, each privilege can be disabled or enabled, with the default being disabled.

In the `tables_priv`, `columns_priv`, and `procs_priv` tables, the privilege columns are declared as `SET` columns. Values in these columns can contain any combination of the privileges controlled by the table. Only those privileges listed in the column value are enabled.

Table 6.7. Set-Type Privilege Column Values

Table Name	Column Name	Possible Set Elements
<code>tables_priv</code>	<code>Table_priv</code>	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view', 'Trigger'
<code>tables_priv</code>	<code>Column_priv</code>	'Select', 'Insert', 'Update', 'References'
<code>columns_priv</code>	<code>Column_priv</code>	'Select', 'Insert', 'Update', 'References'
<code>procs_priv</code>	<code>Proc_priv</code>	'Execute', 'Alter Routine', 'Grant'

Administrative privileges (such as `RELOAD` [696] or `SHUTDOWN` [697]) are specified only in the `user` table. Administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. Consequently, to determine whether you can perform an administrative operation, the server need consult only the `user` table.

The `FILE` [696] privilege also is specified only in the `user` table. It is not an administrative privilege as such, but your ability to read or write files on the server host is independent of the database you are accessing.

The `mysqld` server reads the contents of the grant tables into memory when it starts. You can tell it to reload the tables by issuing a `FLUSH PRIVILEGES` statement or executing a `mysqladmin flush-privileges` or `mysqladmin reload` command. Changes to the grant tables take effect as indicated in Section 6.2.6, “When Privilege Changes Take Effect”.

When you modify an account's privileges, it is a good idea to verify that the changes set up privileges the way you want. To check the privileges for a given account, use the `SHOW GRANTS` statement (see Section 13.7.5.22, “`SHOW GRANTS` Syntax”). For example, to determine the privileges that are granted to an account with user name and host name values of `bob` and `pc84.example.com`, use this statement:

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

6.2.3. Specifying Account Names

MySQL account names consist of a user name and a host name. This enables creation of accounts for users with the same name who can connect from different hosts. This section describes how to write account names, including special values and wildcard rules.

In SQL statements such as `CREATE USER`, `GRANT`, and `SET PASSWORD`, write account names using the following rules:

- Syntax for account names is `'user_name'@'host_name'`.
- An account name consisting only of a user name is equivalent to `'user_name'@'%'`. For example, `'me'` is equivalent to `'me'@'%'`.
- The user name and host name need not be quoted if they are legal as unquoted identifiers. Quotes are necessary to specify a `user_name` string containing special characters (such as “-”), or a `host_name` string containing special characters or wildcard characters (such as “%”); for example, `'test-user'@'%.com'`.
- Quote user names and host names as identifiers or as strings, using either backticks (“`”), single quotation marks (“'”), or double quotation marks (“”).
- The user name and host name parts, if quoted, must be quoted separately. That is, write `'me'@'localhost'`, not `'me@localhost'`; the latter is interpreted as `'me@localhost'@'%'`.
- A reference to the `CURRENT_USER` [1192] or `CURRENT_USER()` [1192] function is equivalent to specifying the current client’s user name and host name literally.

MySQL stores account names in grant tables in the `mysql` database using separate columns for the user name and host name parts:

- The `user` table contains one row for each account. The `User` and `Host` columns store the user name and host name. This table also indicates which global privileges the account has.
- Other grant tables indicate privileges an account has for databases and objects within databases. These tables have `User` and `Host` columns to store the account name. Each row in these tables associates with the account in the `user` table that has the same `User` and `Host` values.

For additional detail about grant table structure, see [Section 6.2.2, “Privilege System Grant Tables”](#).

User names and host names have certain special values or wildcard conventions, as described following.

A user name is either a nonblank value that literally matches the user name for incoming connection attempts, or a blank value (empty string) that matches any user name. An account with a blank user name is an anonymous user. To specify an anonymous user in SQL statements, use a quoted empty user name part, such as `'@'localhost'`.

The host name part of an account name can take many forms, and wildcards are permitted:

- A host value can be a host name or an IP address (IPv4 or IPv6). The name `'localhost'` indicates the local host. The IP address `'127.0.0.1'` indicates the IPv4 loopback interface. The IP address `:::1'` indicates the IPv6 loopback interface.
- You can use the wildcard characters “%” and “_” in host name or IP address values. These have the same meaning as for pattern-matching operations performed with the `LIKE` [1111] operator. For example, a host value of `'%'` matches any host name, whereas a value of `'%.mysql.com'` matches any host in the `mysql.com` domain. `'192.168.1.%'` matches any host in the 192.168.1 class C network.

Because you can use IP wildcard values in host values (for example, `'192.168.1.%'` to match every host on a subnet), someone could try to exploit this capability by naming a host

`192.168.1.somewhere.com`. To foil such attempts, MySQL disallows matching on host names that start with digits and a dot. Thus, if you have a host named something like `1.2.example.com`, its name never matches the host part of account names. An IP wildcard value can match only IP addresses, not host names.

- For a host value specified as an IPv4 address, you can specify a netmask indicating how many address bits to use for the network number. Netmask notation cannot be used for IPv6 addresses.

The syntax is `host_ip/netmask`. For example:

```
CREATE USER 'david'@'192.58.197.0/255.255.255.0';
```

This enables `david` to connect from any client host having an IP address `client_ip` for which the following condition is true:

```
client_ip & netmask = host_ip
```

That is, for the `CREATE USER` statement just shown:

```
client_ip & 255.255.255.0 = 192.58.197.0
```

IP addresses that satisfy this condition and can connect to the MySQL server are those in the range from `192.58.197.0` to `192.58.197.255`.

The netmask can only be used to tell the server to use 8, 16, 24, or 32 bits of the address. Examples:

- `192.0.0.0/255.0.0.0`: Any host on the 192 class A network
- `192.168.0.0/255.255.0.0`: Any host on the 192.168 class B network
- `192.168.1.0/255.255.255.0`: Any host on the 192.168.1 class C network
- `192.168.1.1`: Only the host with this specific IP address

The following netmask will not work because it masks 28 bits, and 28 is not a multiple of 8:

```
192.168.0.1/255.255.255.240
```

The server performs matching of host values in account names against the client host using the value returned by the system DNS resolver for the client host name or IP address. Except in the case that the account host value is specified using netmask notation, this comparison is performed as a string match, even for an account host value given as an IP address. This means that you should specify account host values in the same format used by DNS. Here are examples of problems to watch out for:

- Suppose that a host on the local network has a fully qualified name of `host1.example.com`. If DNS returns name lookups for this host as `host1.example.com`, use that name in account host values. But if DNS returns just `host1`, use `host1` instead.
- If DNS returns the IP address for a given host as `192.168.1.2`, that will match an account host value of `192.168.1.2` but not `192.168.01.2`. Similarly, it will match an account host pattern like `192.168.1.%` but not `192.168.01.%`.

To avoid problems like this, it is advisable to check the format in which your DNS returns host names and addresses, and use values in the same format in MySQL account names.

6.2.4. Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password. If not, the server

denies access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL user name

Identity checking is performed using the three `user` table scope columns (`Host`, `User`, and `Password`). The server accepts the connection only if the `Host` and `User` columns in some `user` table row match the client host name and user name and the client supplies the password specified in that row. The rules for permissible `Host` and `User` values are given in [Section 6.2.3, “Specifying Account Names”](#).

If the `User` column value is nonblank, the user name in an incoming connection must match exactly. If the `User` value is blank, it matches any user name. If the `user` table row that matches an incoming connection has a blank user name, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank user name is used for all further access checking for the duration of the connection (that is, during Stage 2).

The `Password` column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password. If the server authenticates a client using a plugin, the authentication method that the plugin implements may or may not use the password in the `Password` column. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

Nonblank `Password` values in the `user` table represent encrypted passwords. MySQL does not store passwords in plaintext form for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the `PASSWORD()` [1188] function). The encrypted password then is used during the connection process when checking whether the password is correct. This is done without the encrypted password ever traveling over the connection. See [Section 6.3.1, “User Names and Passwords”](#).

From MySQL's point of view, the encrypted password is the *real* password, so you should never give anyone access to it. In particular, *do not give nonadministrative users read access to tables in the `mysql` database*.

The following table shows how various combinations of `Host` and `User` values in the `user` table apply to incoming connections.

Host Value	User Value	Permissible Connections
'thomas.loc.gov'	'fred'	fred, connecting from thomas.loc.gov
'thomas.loc.gov'	''	Any user, connecting from thomas.loc.gov
'%'	'fred'	fred, connecting from any host
'%'	''	Any user, connecting from any host
'%.loc.gov'	'fred'	fred, connecting from any host in the loc.gov domain
'x.y.%'	'fred'	fred, connecting from x.y.net, x.y.com, x.y.edu, and so on; this is probably not useful
'144.155.166.177'	'fred'	fred, connecting from the host with IP address 144.155.166.177
'144.155.166.%'	'fred'	fred, connecting from any host in the 144.155.166 class C subnet
'144.155.166.0/255.255.255.0'	'fred'	Same as previous example

It is possible for the client host name and user name of an incoming connection to match more than one row in the `user` table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from `thomas.loc.gov` by `fred`.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the `user` table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client host name and user name.

The server uses sorting rules that order rows with the most-specific `Host` values first. Literal host names and IP addresses are the most specific. (The specificity of a literal IP address is not affected by whether it has a netmask, so `192.168.1.13` and `192.168.1.0/255.255.255.0` are considered equally specific.) The pattern `'%'` means “any host” and is least specific. The empty string `' '` also means “any host” but sorts after `'%'`. Rows with the same `Host` value are ordered with the most-specific `User` values first (a blank `User` value means “any user” and is least specific).

To see how this works, suppose that the `user` table looks like this:

```
+-----+-----+
| Host      | User    | ...
+-----+-----+
| %         | root    | ...
| %         | jeffrey | ...
| localhost | root    | ...
| localhost |         | ...
+-----+-----+
```

When the server reads the table into memory, it sorts the rows using the rules just described. The result after sorting looks like this:

```
+-----+-----+
| Host      | User    | ...
+-----+-----+
| localhost | root    | ...
| localhost |         | ...
| %         | jeffrey | ...
| %         | root    | ...
+-----+-----+
```

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the rows from the table match: the one with `Host` and `User` values of `'localhost'` and `' '`, and the one with values of `'%'` and `'jeffrey'`. The `'localhost'` row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the `user` table looks like this:

```
+-----+-----+
| Host      | User    | ...
+-----+-----+
| %         | jeffrey | ...
| thomas.loc.gov |         | ...
+-----+-----+
```

The sorted table looks like this:

```
+-----+-----+
| Host      | User    | ...
+-----+-----+
```

```

+-----+-----+
| thomas.loc.gov |   | ...
| %              | jeffrey | ...
+-----+-----+

```

A connection by `jeffrey` from `thomas.loc.gov` is matched by the first row, whereas a connection by `jeffrey` from any host is matched by the second.



Note

It is a common misconception to think that, for a given user name, all rows that explicitly name that user are used first when the server attempts to find a match for the connection. This is not true. The preceding example illustrates this, where a connection from `thomas.loc.gov` by `jeffrey` is first matched not by the row containing `'jeffrey'` as the `User` column value, but by the row with no user name. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a user name when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` [1192] function. (See [Section 12.14, “Information Functions”](#).) It returns a value in `user_name@host_name` format that indicates the `User` and `Host` values from the matching `user` table row. Suppose that `jeffrey` connects and issues the following query:

```

mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| @localhost     |
+-----+

```

The result shown here indicates that the matching `user` table row had a blank `User` column value. In other words, the server is treating `jeffrey` as an anonymous user.

Another way to diagnose authentication problems is to print out the `user` table and sort it by hand to see where the first match is being made.

6.2.5. Access Control, Stage 2: Request Verification

After you establish a connection, the server enters Stage 2 of access control. For each request that you issue through that connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the `user`, `db`, `host`, `tables_priv`, `columns_priv`, or `procs_priv` tables. (You may find it helpful to refer to [Section 6.2.2, “Privilege System Grant Tables”](#), which lists the columns present in each of the grant tables.)

The `user` table grants privileges that are assigned to you on a global basis and that apply no matter what the default database is. For example, if the `user` table grants you the `DELETE` [696] privilege, you can delete rows from any table in any database on the server host! It is wise to grant privileges in the `user` table only to people who need them, such as database administrators. For other users, you should leave all privileges in the `user` table set to `'N'` and grant privileges at more specific levels only. You can grant privileges for particular databases, tables, columns, or routines.

The `db` and `host` tables grant database-specific privileges. Values in the scope columns of these tables can take the following forms:

- A blank `User` value in the `db` table matches the anonymous user. A nonblank value matches literally; there are no wildcards in user names.

- The wildcard characters “%” and “_” can be used in the `Host` and `Db` columns of either table. These have the same meaning as for pattern-matching operations performed with the `LIKE [1111]` operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include the underscore character (“_”) as part of a database name, specify it as “_” in the `GRANT` statement.
- A ‘%’ `Host` value in the `db` table means “any host.” A blank `Host` value in the `db` table means “consult the `host` table for further information” (a process that is described later in this section).
- A ‘%’ or blank `Host` value in the `host` table means “any host.”
- A ‘%’ or blank `Db` value in either table means “any database.”

The server reads the `db` and `host` tables into memory and sorts them at the same time that it reads the `user` table. The server sorts the `db` table based on the `Host`, `Db`, and `User` scope columns, and sorts the `host` table based on the `Host` and `Db` scope columns. As with the `user` table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching entries, it uses the first match that it finds.

The `tables_priv`, `columns_priv`, and `procs_priv` tables grant table-specific, column-specific, and routine-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters “%” and “_” can be used in the `Host` column. These have the same meaning as for pattern-matching operations performed with the `LIKE [1111]` operator.
- A ‘%’ or blank `Host` value means “any host.”
- The `Db`, `Table_name`, `Column_name`, and `Routine_name` columns cannot contain wildcards or be blank.

The server sorts the `tables_priv`, `columns_priv`, and `procs_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as `SHUTDOWN [697]` or `RELOAD [696]`, the server checks only the `user` table row because that is the only table that specifies administrative privileges. The server grants access if the row permits the requested operation and denies access otherwise. For example, if you want to execute `mysqladmin shutdown` but your `user` table row does not grant the `SHUTDOWN [697]` privilege to you, the server denies access without even checking the `db` or `host` tables. (They contain no `Shutdown_priv` column, so there is no need to do so.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user's global privileges by looking in the `user` table row. If the row permits the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user's database-specific privileges by checking the `db` and `host` tables:

1. The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns. The `Host` and `User` columns are matched to the connecting user's host name and MySQL user name. The `Db` column is matched to the database that the user wants to access. If there is no row for the `Host` and `User`, access is denied.
2. If there is a matching `db` table row and its `Host` column is not blank, that row defines the user's database-specific privileges.
3. If the matching `db` table row's `Host` column is blank, it signifies that the `host` table enumerates which hosts should be permitted access to the database. In this case, a further lookup is done in the `host`

table to find a match on the `Host` and `Db` columns. If no `host` table row matches, access is denied. If there is a match, the user's database-specific privileges are computed as the intersection (*not* the union!) of the privileges in the `db` and `host` table entries; that is, the privileges that are 'Y' in both entries. (This way you can grant general privileges in the `db` table row and then selectively restrict them on a host-by-host basis using the `host` table entries.)

After determining the database-specific privileges granted by the `db` and `host` table entries, the server adds them to the global privileges granted by the `user` table. If the result permits the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the `tables_priv` and `columns_priv` tables, adds those to the user's privileges, and permits or denies access based on the result. For stored-routine operations, the server uses the `procs_priv` table rather than `tables_priv` and `columns_priv`.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

It may not be apparent why, if the global `user` row privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT INTO ... SELECT` statement, you need both the `INSERT` [696] and the `SELECT` [697] privileges. Your privileges might be such that the `user` table row grants one privilege and the `db` table row grants the other. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either table by itself; the privileges granted by the entries in both tables must be combined.

The `host` table is not affected by the `GRANT` or `REVOKE` statements, so it is unused in most MySQL installations. If you modify it directly, you can use it for some specialized purposes, such as to maintain a list of secure servers on the local network that are granted all privileges.

You can also use the `host` table to indicate hosts that are *not* secure. Suppose that you have a machine `public.your.domain` that is located in a public area that you do not consider secure. You can enable access to all hosts on your network except that machine by using `host` table entries like this:

```
+-----+-----+
| Host          | Db | ...
+-----+-----+
| public.your.domain | % | ... (all privileges set to 'N')
| %.your.domain   | % | ... (all privileges set to 'Y')
+-----+-----+
```

6.2.6. When Privilege Changes Take Effect

When `mysqld` starts, it reads all grant table contents into memory. The in-memory tables become effective for access control at that point.

If you modify the grant tables indirectly using account-management statements such as `GRANT`, `REVOKE`, `SET PASSWORD`, or `RENAME USER`, the server notices these changes and loads the grant tables into memory again immediately.

If you modify the grant tables directly using statements such as `INSERT`, `UPDATE`, or `DELETE`, your changes have no effect on privilege checking until you either restart the server or tell it to reload the tables.

If you change the grant tables directly but forget to reload them, your changes have *no effect* until you restart the server. This may leave you wondering why your changes seem to make no difference!

To tell the server to reload the grant tables, perform a flush-privileges operation. This can be done by issuing a `FLUSH PRIVILEGES` statement or by executing a `mysqladmin flush-privileges` or `mysqladmin reload` command.

A grant table reload affects privileges for each existing client connection as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect the next time the client executes a `USE db_name` statement.



Note

Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database or flushing the privileges.

- Global privileges and passwords are unaffected for a connected client. These changes take effect only for subsequent connections.

If the server is started with the `--skip-grant-tables` [461] option, it does not read the grant tables or implement any access control. Anyone can connect and do anything, *which is insecure*. To cause a server thus started to read the tables and enable access checking, flush the privileges.

6.2.7. Causes of Access-Denied Errors

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not, clients cannot connect to it. For example, if an attempt to connect to the server fails with a message such as one of those following, one cause might be that the server is not running:

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

- It might be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a `--port` [239] option to indicate the proper port number, or a `--socket` [240] option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```
shell> netstat -ln | grep mysql
```

- Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with `--skip-networking` [461], it will not accept TCP/IP connections at all. If the server was started with `--bind-address=127.0.0.1` [435], it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.
- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that

the port has not been blocked. Under Windows, applications such as ZoneAlarm or the Windows XP personal firewall may need to be configured not to block the MySQL port.

- The grant tables must be properly set up so that the server can use them for access control. For some distribution types (such as binary distributions on Windows, or RPM distributions on Linux), the installation process initializes the `mysql` database containing the grant tables. For distributions that do not do this, you must initialize the grant tables manually by running the `mysql_install_db` script. For details, see [Section 2.11.1, “Unix Postinstallation Procedures”](#).

To determine whether you need to initialize the grant tables, look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If not, execute the `mysql_install_db` script. After running this script and starting the server, test the initial privileges by executing this command:

```
shell> mysql -u root test
```

The server should let you connect without error.

- After a fresh installation, you should connect to the server and set up your users and their access permissions:

```
shell> mysql -u root mysql
```

The server should let you connect because the MySQL `root` user has no password initially. That is also a security risk, so setting the password for the `root` accounts is something you should do while you're setting up your other MySQL accounts. For instructions on setting the initial passwords, see [Section 2.11.2, “Securing the Initial MySQL Accounts”](#).

- If you have updated an existing MySQL installation to a newer version, did you run the `mysql_upgrade` script? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).
- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

For information on how to deal with this, see [Section 6.1.2.4, “Password Hashing in MySQL”](#), and [Section C.5.2.4, “Client does not support authentication protocol”](#).

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you have not specified them on the command line, check any applicable option files and your environment. For example, if you get `Access denied` when you run a client without any options, make sure that you have not specified an old password in any of your option files!

You can suppress the use of option files by a client program by invoking it with the `--no-defaults` [248] option. For example:

```
shell> mysqladmin --no-defaults -u root version
```

The option files that clients use are listed in [Section 4.2.3.3, “Using Option Files”](#). Environment variables are listed in [Section 2.13, “Environment Variables”](#).

- If you get the following error, it means that you are using an incorrect `root` password:

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

If the preceding error occurs even when you have not specified a password, it means that you have an incorrect password listed in some option file. Try the `--no-defaults` [248] option as described in the previous item.

For information on changing passwords, see [Section 6.3.5, “Assigning Account Passwords”](#).

If you have lost or forgotten the `root` password, see [Section C.5.4.1, “How to Reset the Root Password”](#).

- If you change a password by using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must encrypt the password using the `PASSWORD()` [1188] function. If you do not use `PASSWORD()` [1188] for these statements, the password will not work. For example, the following statement assigns a password, but fails to encrypt it, so the user is not able to connect afterward:

```
SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

Instead, set the password like this:

```
SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

The `PASSWORD()` [1188] function is unnecessary when you specify a password using the `CREATE USER` or `GRANT` statements or the `mysqladmin password` command. Each of those automatically uses `PASSWORD()` [1188] to encrypt the password. See [Section 6.3.5, “Assigning Account Passwords”](#), and [Section 13.7.1.1, “CREATE USER Syntax”](#).

- `localhost` is a synonym for your local host name, and is also the default host to which clients try to connect if you specify no host explicitly.

To avoid this problem on such systems, you can use a `--host=127.0.0.1` [239] option to name the server host explicitly. This will make a TCP/IP connection to the local `mysqld` server. You can also use TCP/IP by specifying a `--host` [239] option that uses the actual host name of the local host. In this case, the host name must be specified in a `user` table row on the server host, even though you are running the client program on the same host as the server.

- The `Access denied` error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the `user` table that exactly matches the host name and user name that were given in the error message. For example, if you get an error message that contains `using password: NO`, it means that you tried to log in without a password.
- If you get an `Access denied` error when trying to connect to the database with `mysql -u user_name`, you may have a problem with the `user` table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

```
SELECT * FROM user;
```

The result should include a row with the `Host` and `User` columns matching your client's host name and your MySQL user name.

- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the `user` table with a `Host` value that matches the client host:

```
Host ... is not allowed to connect to this MySQL server
```

You can fix this by setting up an account for the combination of client host name and user name that you are using when trying to connect.

If you do not know the IP address or host name of the machine from which you are connecting, you should put a row with '%' as the `Host` column value in the `user` table. After trying to connect from the client machine, use a `SELECT USER()` query to see how you really did connect. Then change the '%' in the `user` table row to the actual host name that shows up in the log. Otherwise, your system is left insecure because it permits connections from any host for the given user name.

On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the `glibc` library than the one you are using. In this case, you should either upgrade your operating system or `glibc`, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

- If you specify a host name when trying to connect, but get an error message where the host name is not shown or is an IP address, it means that the MySQL server got an error when trying to resolve the IP address of the client host to a name:

```
shell> mysqladmin -u root -pXXXX -h some_hostname ver
Access denied for user 'root'@'' (using password: YES)
```

If you try to connect as `root` and get the following error, it means that you do not have a row in the `user` table with a `User` column value of '`root`' and that `mysqld` cannot resolve the host name for your client:

```
Access denied for user ''@'unknown'
```

These errors indicate a DNS problem. To fix it, execute `mysqladmin flush-hosts` to reset the internal DNS host cache. See [Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”](#).

Some permanent solutions are:

- Determine what is wrong with your DNS server and fix it.
- Specify IP addresses rather than host names in the MySQL grant tables.
- Put an entry for the client machine name in `/etc/hosts` on Unix or `\windows\hosts` on Windows.
- Start `mysqld` with the `--skip-name-resolve` [461] option.
- Start `mysqld` with the `--skip-host-cache` [461] option.
- On Unix, if you are running the server and the client on the same machine, connect to `localhost`. Unix connections to `localhost` use a Unix socket file rather than TCP/IP.
- On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the host name `.` (period). Connections to `.` use a named pipe rather than TCP/IP.
- If `mysql -u root test` works but `mysql -h your_hostname -u root test` results in `Access denied` (where `your_hostname` is the actual host name of the local host), you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the `user` table row specifies an unqualified host name, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have an entry with host '`pluto`' in the

`user` table, but your DNS tells MySQL that your host name is `'pluto.example.com'`, the entry does not work. Try adding an entry to the `user` table that contains the IP address of your host as the `Host` column value. (Alternatively, you could add an entry to the `user` table with a `Host` value that contains a wildcard; for example, `'pluto.%'`. However, use of `Host` values ending with `"%"` is *insecure* and is *not* recommended!)

- If `mysql -u user_name test` works but `mysql -u user_name other_db` does not, you have not granted access to the given user for the database named `other_db`.
- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` does not work when executed on a remote client host, you have not enabled access to the server for the given user name from the remote host.
- If you cannot figure out why you get `Access denied`, remove from the `user` table all entries that have `Host` values containing wildcards (entries that contain `'%'` or `'_'` characters). A very common error is to insert a new entry with `Host='%'` and `User='some_user'`, thinking that this enables you to specify `localhost` to connect from the same machine. The reason that this does not work is that the default privileges include an entry with `Host='localhost'` and `User=''`. Because that entry has a `Host` value `'localhost'` that is more specific than `'%'`, it is used in preference to the new entry when connecting from `localhost`! The correct procedure is to insert a second entry with `Host='localhost'` and `User='some_user'`, or to delete the entry with `Host='localhost'` and `User=''`. After deleting the entry, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables. See also [Section 6.2.4, "Access Control, Stage 1: Connection Verification"](#).
- If you are able to connect to the MySQL server, but get an `Access denied` message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA INFILE` statement, your entry in the `user` table does not have the `FILE [696]` privilege enabled.
- If you change the grant tables directly (for example, by using `INSERT`, `UPDATE`, or `DELETE` statements) and your changes seem to be ignored, remember that you must execute a `FLUSH PRIVILEGES` statement or a `mysqladmin flush-privileges` command to cause the server to reload the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the `root` password with an `UPDATE` statement, you will not need to specify the new password until after you flush the privileges, because the server will not know you've changed the password yet!
- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in [Section 6.2.6, "When Privilege Changes Take Effect"](#).
- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -p your_pass db_name`. If you are able to connect using the `mysql` client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=your_pass [239]` syntax to specify the password. If you use the `-p` or `--password [239]` option with no password value, MySQL prompts you for the password.)
- For testing purposes, start the `mysqld` server with the `--skip-grant-tables [461]` option. Then you can change the MySQL grant tables and use the `mysqlaccess` script to check whether your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to reload the privileges. This enables you to begin using the new grant table contents without stopping and restarting the server.
- If you get the following error, you may have a problem with the `db` or `host` table:

```
Access to database denied
```

If the entry selected from the `db` table has an empty value in the `Host` column, make sure that there are one or more corresponding entries in the `host` table specifying which hosts the `db` table entry applies to. This problem occurs infrequently because the `host` table is rarely used.

- If everything else fails, start the `mysqld` server with a debugging option (for example, `--debug=d,general,query` [439]). This prints host and user information about attempted connections, as well as information about each command issued. See [MySQL Internals: Porting to Other Systems](#).
- If you have any other problems with the MySQL grant tables and feel you must post the problem to the mailing list, always provide a dump of the MySQL grant tables. You can dump the tables with the `mysqldump mysql` command. To file a bug report, see the instructions at [Section 1.7, “How to Report Bugs or Problems”](#). In some cases, you may need to restart `mysqld` with `--skip-grant-tables` [461] to run `mysqldump`.

6.3. MySQL User Account Management

This section describes how to set up accounts for clients of your MySQL server. It discusses the following topics:

- The meaning of account names and passwords as used in MySQL and how that compares to names and passwords used by your operating system
- How to set up new accounts and remove existing accounts
- How to change passwords
- Guidelines for using passwords securely
- How to use secure connections with SSL

See also [Section 13.7.1, “Account Management Statements”](#), which describes the syntax and use for all user-management SQL statements.

6.3.1. User Names and Passwords

MySQL stores accounts in the `user` table of the `mysql` database. An account is defined in terms of a user name and the client host or hosts from which the user can connect to the server. The account may also have a password. For information about account representation in the `user` table, see [Section 6.2.2, “Privilege System Grant Tables”](#). MySQL 5.5 supports authentication plugins, so it is possible that an account authenticates using some external authentication method. See [Section 6.3.6, “Pluggable Authentication”](#).

There are several distinctions between the way user names and passwords are used by MySQL and the way they are used by your operating system:

- User names, as used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix user name as the MySQL user name, but that is for convenience only. The default can be overridden easily, because client programs permit any user name to be specified with a `-u` or `--user` option. Because this means that anyone can attempt to connect to the server using any user name, you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a user name for an account that has no password is able to connect successfully to the server.
- MySQL user names can be up to 16 characters long. Operating system user names, because they are completely unrelated to MySQL user names, may be of a different maximum length. For example, Unix user names typically are limited to eight characters.



Warning

The limit on MySQL user name length is hard-coded in the MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database *does not work*.

You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure that is described in [Section 4.4.7](#), “`mysql_upgrade` — Check and Upgrade MySQL Tables”. Attempting to redefine MySQL’s system tables in any other fashion results in undefined (and unsupported!) behavior.

- The server uses MySQL passwords stored in the `user` table to authenticate client connections using MySQL native authentication (against passwords stored in the `mysql.user` table). These passwords have nothing to do with passwords for logging in to your operating system. There is no necessary connection between the “external” password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.

If the server authenticates a client using a plugin, the authentication method that the plugin implements may or may not use the password in the `user` table. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

- MySQL encrypts passwords stored in the `user` table using its own algorithm. This encryption is the same as that implemented by the `PASSWORD() [1188]` SQL function but differs from that used during the Unix login process. Unix password encryption is the same as that implemented by the `ENCRYPT() [1187]` SQL function. See the descriptions of the `PASSWORD() [1188]` and `ENCRYPT() [1187]` functions in [Section 12.13](#), “Encryption and Compression Functions”.

From version 4.1 on, MySQL employs a stronger authentication method that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. (In earlier versions, even though passwords are stored in encrypted form in the `user` table, knowledge of the encrypted password value could be used to connect to the MySQL server.) [Section 6.1.2.4](#), “Password Hashing in MySQL”, discusses password encryption further.

- It is possible to connect to the server regardless of character set settings if the user name and password contain only ASCII characters. To connect when the user name or password contain non-ASCII characters, the client should call the `mysql_options()` C API function with the `MYSQL_SET_CHARSET_NAME` option and appropriate character set name as arguments. This causes authentication to take place using the specified character set. Otherwise, authentication will fail unless the server default character set is the same as the encoding in the authentication defaults.

Standard MySQL client programs support a `--default-character-set` option that causes `mysql_options()` to be called as just described. In addition, character set autodetection is supported as described in [Section 10.1.4](#), “Connection Character Sets and Collations”. For programs that use a connector that is not based on the C API, the connector may provide an equivalent to `mysql_options()` that can be used instead. Check the connector documentation.

The preceding notes do not apply for `ucs2`, `utf16`, and `utf32`, which are not permitted as client character sets.

When you install MySQL, the grant tables are populated with an initial set of accounts. The names and access privileges for these accounts are described in [Section 2.11.2](#), “Securing the Initial MySQL Accounts”, which also discusses how to assign passwords to them. Thereafter, you normally set up,

modify, and remove MySQL accounts using statements such as `CREATE USER`, `GRANT`, and `REVOKE`. See [Section 13.7.1, “Account Management Statements”](#).

When you connect to a MySQL server with a command-line client, specify the user name and password as necessary for the account that you want to use:

```
shell> mysql --user=monty --password=password db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u monty -ppassword db_name
```

There must be *no space* between the `-p` option and the following password value.

If you omit the `password` value following the `--password [239]` or `-p` option on the command line, the client prompts for one.

Specifying a password on the command line should be considered insecure. See [Section 6.1.2.1, “End-User Guidelines for Password Security”](#). You can use an option file to avoid giving the password on the command line.

For additional information about specifying user names, passwords, and other connection parameters, see [Section 4.2.2, “Connecting to the MySQL Server”](#).

6.3.2. Adding User Accounts

You can create MySQL accounts in two ways:

- By using statements intended for creating accounts, such as `CREATE USER` or `GRANT`. These statements cause the server to make appropriate modifications to the grant tables.
- By manipulating the MySQL grant tables directly with statements such as `INSERT`, `UPDATE`, or `DELETE`.

The preferred method is to use account-creation statements because they are more concise and less error-prone than manipulating the grant tables directly. `CREATE USER` and `GRANT` are described in [Section 13.7.1, “Account Management Statements”](#).

Another option for creating accounts is to use the GUI tool MySQL Workbench. Or one of several available third-party programs that offer capabilities for MySQL account administration. `phpMyAdmin` is one such program.

The following examples show how to use the `mysql` client program to set up new accounts. These examples assume that privileges have been set up according to the defaults described in [Section 2.11.2, “Securing the Initial MySQL Accounts”](#). This means that to make changes, you must connect to the MySQL server as the MySQL `root` user, and the `root` account must have the `INSERT [696]` privilege for the `mysql` database and the `RELOAD [696]` administrative privilege.

As noted in the examples where appropriate, some of the statements will fail if the server's SQL mode has been set to enable certain restrictions. In particular, strict mode (`STRICT_TRANS_TABLES [617]`, `STRICT_ALL_TABLES [616]`) and `NO_AUTO_CREATE_USER [613]` will prevent the server from accepting some of the statements. Workarounds are indicated for these cases. For more information about SQL modes and their effect on grant table manipulation, see [Section 5.1.7, “Server SQL Modes”](#), and [Section 13.7.1.3, “GRANT Syntax”](#).

First, use the `mysql` program to connect to the server as the MySQL `root` user:

```
shell> mysql --user=root mysql
```

If you have assigned a password to the `root` account, you will also need to supply a `--password` or `-p` option, both for this `mysql` command and for those later in this section.

After connecting to the server as `root`, you can add new accounts. The following statements use `GRANT` to set up four new accounts:

```
mysql> CREATE USER 'monty'@'localhost' IDENTIFIED BY 'some_pass';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'
-> WITH GRANT OPTION;
mysql> CREATE USER 'monty'@'%' IDENTIFIED BY 'some_pass';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'
-> WITH GRANT OPTION;
mysql> CREATE USER 'admin'@'localhost';
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
mysql> CREATE USER 'dummy'@'localhost';
```

The accounts created by these statements have the following properties:

- Two of the accounts have a user name of `monty` and a password of `some_pass`. Both accounts are superuser accounts with full privileges to do anything. The `'monty'@'localhost'` account can be used only when connecting from the local host. The `'monty'@'%'` account uses the `'%'` wildcard for the host part, so it can be used to connect from any host.

It is necessary to have both accounts for `monty` to be able to connect from anywhere as `monty`. Without the `localhost` account, the anonymous-user account for `localhost` that is created by `mysql_install_db` would take precedence when `monty` connects from the local host. As a result, `monty` would be treated as an anonymous user. The reason for this is that the anonymous-user account has a more specific `Host` column value than the `'monty'@'%'` account and thus comes earlier in the `user` table sort order. (`user` table sorting is discussed in [Section 6.2.4, “Access Control, Stage 1: Connection Verification”](#).)

- The `'admin'@'localhost'` account has no password. This account can be used only by `admin` to connect from the local host. It is granted the `RELOAD` [696] and `PROCESS` [696] administrative privileges. These privileges enable the `admin` user to execute the `mysqladmin reload`, `mysqladmin refresh`, and `mysqladmin flush-xxx` commands, as well as `mysqladmin processlist`. No privileges are granted for accessing any databases. You could add such privileges later by issuing other `GRANT` statements.
- The `'dummy'@'localhost'` account has no password. This account can be used only to connect from the local host. No privileges are granted. It is assumed that you will grant specific privileges to the account later.

The statements that create accounts with no password will fail if the `NO_AUTO_CREATE_USER` [613] SQL mode is enabled. To deal with this, use an `IDENTIFIED BY` clause that specifies a nonempty password.

To check the privileges for an account, use `SHOW GRANTS`:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----+
| Grants for admin@localhost |
+-----+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----+
```

As an alternative to `CREATE USER` and `GRANT`, you can create the same accounts directly by issuing `INSERT` statements and then telling the server to reload the grant tables using `FLUSH PRIVILEGES`:

```
shell> mysql --user=root mysql
```

```
mysql> INSERT INTO user
-> VALUES('localhost','monty',PASSWORD('some_pass'),
-> 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
-> VALUES('%','monty',PASSWORD('some_pass'),
-> 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y',
-> 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y',
-> '','','','','0,0,0,0);
mysql> INSERT INTO user SET Host='localhost',User='admin',
-> Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User>Password)
-> VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

When you create accounts with `INSERT`, it is necessary to use `FLUSH PRIVILEGES` to tell the server to reload the grant tables. Otherwise, the changes go unnoticed until you restart the server. With `CREATE USER`, `FLUSH PRIVILEGES` is unnecessary.

The reason for using the `PASSWORD()` [1188] function with `INSERT` is to encrypt the password. The `CREATE USER` statement encrypts the password for you, so `PASSWORD()` [1188] is unnecessary.

The 'Y' values enable privileges for the accounts. Depending on your MySQL version, you may have to use a different number of 'Y' values in the first two `INSERT` statements. The `INSERT` statement for the `admin` account employs the more readable extended `INSERT` syntax using `SET`.

In the `INSERT` statement for the `dummy` account, only the `Host`, `User`, and `Password` columns in the `user` table row are assigned values. None of the privilege columns are set explicitly, so MySQL assigns them all the default value of 'N'. This is equivalent to what `CREATE USER` does.

If strict SQL mode is enabled, all columns that have no default value must have a value specified. In this case, `INSERT` statements must explicitly specify values for the `ssl_cipher`, `x509_issuer`, and `x509_subject` columns.

To set up a superuser account, it is necessary only to insert a `user` table row with all privilege columns set to 'Y'. The `user` table privileges are global, so no entries in any of the other grant tables are needed.

The next examples create three accounts and give them access to specific databases. Each of them has a user name of `custom` and password of `obscure`.

To create the accounts with `CREATE USER` and `GRANT`, use the following statements:

```
shell> mysql --user=root mysql
mysql> CREATE USER 'custom'@'localhost' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON bankaccount.*
-> TO 'custom'@'localhost';
mysql> CREATE USER 'custom'@'host47.example.com' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON expenses.*
-> TO 'custom'@'host47.example.com';
mysql> CREATE USER 'custom'@'server.domain' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON customer.*
-> TO 'custom'@'server.domain';
```

The three accounts can be used as follows:

- The first account can access the `bankaccount` database, but only from the local host.
- The second account can access the `expenses` database, but only from the host `host47.example.com`.

- The third account can access the `customer` database, but only from the host `server.domain`.

To set up the `custom` accounts without `GRANT`, use `INSERT` statements as follows to modify the grant tables directly:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
-> VALUES ('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
-> VALUES ('host47.example.com','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
-> VALUES ('server.domain','custom',PASSWORD('obscure'));
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv>Delete_priv>Create_priv>Drop_priv)
-> VALUES ('localhost','bankaccount','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv>Delete_priv>Create_priv>Drop_priv)
-> VALUES ('host47.example.com','expenses','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv>Delete_priv>Create_priv>Drop_priv)
-> VALUES ('server.domain','customer','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

The first three `INSERT` statements add `user` table entries that permit the user `custom` to connect from the various hosts with the given password, but grant no global privileges (all privileges are set to the default value of 'N'). The next three `INSERT` statements add `db` table entries that grant privileges to `custom` for the `bankaccount`, `expenses`, and `customer` databases, but only when accessed from the proper hosts. As usual when you modify the grant tables directly, you must tell the server to reload them with `FLUSH PRIVILEGES` so that the privilege changes take effect.

To create a user who has access from all machines in a given domain (for example, `mydomain.com`), you can use the “%” wildcard character in the host part of the account name:

```
mysql> CREATE USER 'myname'@'%.mydomain.com' IDENTIFIED BY 'mypass';
```

To do the same thing by modifying the grant tables directly, do this:

```
mysql> INSERT INTO user (Host,User,Password,...)
-> VALUES ('%.mydomain.com','myname',PASSWORD('mypass'),...);
mysql> FLUSH PRIVILEGES;
```

6.3.3. Removing User Accounts

To remove an account, use the `DROP USER` statement, which is described in [Section 13.7.1.2, “DROP USER Syntax”](#).

6.3.4. Setting Account Resource Limits

One means of limiting use of MySQL server resources is to set the global `max_user_connections` [\[525\]](#) system variable to a nonzero value. This limits the number of simultaneous connections that can be made by any given account, but places no limits on what a client can do once connected. In addition, setting `max_user_connections` [\[525\]](#) does not enable management of individual accounts. Both types of control are of interest to many MySQL administrators, particularly those working for Internet Service Providers.

In MySQL 5.5, you can limit use of the following server resources for individual accounts:

- The number of queries that an account can issue per hour
- The number of updates that an account can issue per hour
- The number of times an account can connect to the server per hour
- The number of simultaneous connections to the server by an account

Any statement that a client can issue counts against the query limit (unless its results are served from the query cache). Only statements that modify databases or tables count against the update limit.

An “account” in this context corresponds to a row in the `mysql.user` table. That is, a connection is assessed against the `User` and `Host` values in the `user` table row that applies to the connection. For example, an account `'usera'@'%.example.com'` corresponds to a row in the `user` table that has `User` and `Host` values of `usera` and `%.example.com`, to permit `usera` to connect from any host in the `example.com` domain. In this case, the server applies resource limits in this row collectively to all connections by `usera` from any host in the `example.com` domain because all such connections use the same account.

Before MySQL 5.0.3, an “account” was assessed against the actual host from which a user connects. This older method accounting may be selected by starting the server with the `--old-style-user-limits` [455] option. In this case, if `usera` connects simultaneously from `host1.example.com` and `host2.example.com`, the server applies the account resource limits separately to each connection. If `usera` connects again from `host1.example.com`, the server applies the limits for that connection together with the existing connection from that host.

To set resource limits for an account, use the `GRANT` statement (see Section 13.7.1.3, “`GRANT Syntax`”). Provide a `WITH` clause that names each resource to be limited. The default value for each limit is zero (no limit). For example, to create a new account that can access the `customer` database, but only in a limited fashion, issue these statements:

```
mysql> CREATE USER 'francis'@'localhost' IDENTIFIED BY 'frank';
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
->     WITH MAX_QUERIES_PER_HOUR 20
->         MAX_UPDATES_PER_HOUR 10
->         MAX_CONNECTIONS_PER_HOUR 5
->         MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the `WITH` clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. For `MAX_USER_CONNECTIONS`, the limit is an integer representing the maximum number of simultaneous connections by the account. If this limit is set to zero, the global `max_user_connections` [525] system variable value determines the number of simultaneous connections. If `max_user_connections` [525] is also zero, there is no limit for the account.

To modify existing limits for an account, use a `GRANT USAGE` statement at the global level (`ON *.*`). The following statement changes the query limit for `francis` to 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
->     WITH MAX_QUERIES_PER_HOUR 100;
```

The statement modifies only the limit value specified and leaves the account otherwise unchanged.

To remove a limit, set its value to zero. For example, to remove the limit on how many times per hour `francis` can connect, use this statement:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'  
-> WITH MAX_CONNECTIONS_PER_HOUR 0;
```

As mentioned previously, the simultaneous-connection limit for an account is determined from the `MAX_USER_CONNECTIONS` limit and the `max_user_connections` [525] system variable. Suppose that the global `max_user_connections` [525] value is 10 and three accounts have resource limits specified with `GRANT`:

```
GRANT ... TO 'user1'@'localhost' WITH MAX_USER_CONNECTIONS 0;  
GRANT ... TO 'user2'@'localhost' WITH MAX_USER_CONNECTIONS 5;  
GRANT ... TO 'user3'@'localhost' WITH MAX_USER_CONNECTIONS 20;
```

`user1` has a connection limit of 10 (the global `max_user_connections` [525] value) because it has a zero `MAX_USER_CONNECTIONS` limit). `user2` and `user3` have connection limits of 5 and 20, respectively, because they have nonzero `MAX_USER_CONNECTIONS` limits.

The server stores resource limits for an account in the `user` table row corresponding to the account. The `max_questions`, `max_updates`, and `max_connections` columns store the per-hour limits, and the `max_user_connections` column stores the `MAX_USER_CONNECTIONS` limit. (See [Section 6.2.2, “Privilege System Grant Tables”](#).)

Resource-use counting takes place when any account has a nonzero limit placed on its use of any of the resources.

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, further connections for the account are rejected until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, further queries or updates are rejected until the hour is up. In all such cases, an appropriate error message is issued.

Resource counting is done per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a `mysqladmin reload` command).
- The counts for an individual account can be set to zero by re-granting it any of its limits. To do this, use `GRANT USAGE` as described earlier and specify a limit value equal to the value that the account currently has.

Counter resets do not affect the `MAX_USER_CONNECTIONS` limit.

All counts begin at zero when the server starts; counts are not carried over through a restart.

For the `MAX_USER_CONNECTIONS` limit, an edge case can occur if the account currently has open the maximum number of connections permitted to it: A disconnect followed quickly by a connect can result in an error (`ER_TOO_MANY_USER_CONNECTIONS` [3993] or `ER_USER_LIMIT_REACHED` [3994]) if the server has not fully processed the disconnect by the time the connect occurs. When the server finishes disconnect processing, another connection will once more be permitted.

6.3.5. Assigning Account Passwords

Required credentials for clients that connect to the MySQL server can include a password. This section describes how to assign passwords for MySQL accounts. In MySQL 5.5, it is also possible for clients to authenticate using plugins. For information, see [Section 6.3.6, “Pluggable Authentication”](#).

To assign a password when you create a new account with `CREATE USER`, include an `IDENTIFIED BY` clause:

```
mysql> CREATE USER 'jeffrey'@'localhost'  
-> IDENTIFIED BY 'mypass';
```

To assign or change a password for an existing account, one way is to issue a `SET PASSWORD` statement:

```
mysql> SET PASSWORD FOR  
-> 'jeffrey'@'localhost' = PASSWORD('mypass');
```

MySQL stores passwords in the `user` table in the `mysql` database. Only users such as `root` that have update access to the `mysql` database can change the password for other users. If you are not connected as an anonymous user, you can change your own password by omitting the `FOR` clause:

```
mysql> SET PASSWORD = PASSWORD('mypass');
```

The `old_passwords` [534] system variable value determines the hashing method used by `PASSWORD()` [1188]. If you specify the password using that function and `SET PASSWORD` rejects the password as not being in the correct format, it may be necessary to set `old_passwords` [534] to change the hashing method. For descriptions of the permitted values, see [Section 5.1.4, “Server System Variables”](#).

In MySQL 5.5, enabling the `read_only` [544] system variable prevents the use of the `SET PASSWORD` statement by any user not having the `SUPER` [697] privilege.

You can also use a `GRANT USAGE` statement at the global level (`ON *.*`) to assign a password to an account without affecting the account's current privileges:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'localhost'  
-> IDENTIFIED BY 'mypass';
```

To assign a password from the command line, use the `mysqladmin` command:

```
shell> mysqladmin -u user_name -h host_name password "newpwd"
```

The account for which this command sets the password is the one with a `user` table row that matches `user_name` in the `User` column and the client host *from which you connect* in the `Host` column.

During authentication when a client connects to the server, MySQL treats the password in the `user` table as an encrypted hash value (the value that `PASSWORD()` [1188] would return for the password). When assigning a password to an account, it is important to store an encrypted value, not the plaintext password. Use the following guidelines:

- When you assign a password using `CREATE USER`, `GRANT` with an `IDENTIFIED BY` clause, or the `mysqladmin password` command, they encrypt the password for you. Specify the literal plaintext password:

```
mysql> CREATE USER 'jeffrey'@'localhost'  
-> IDENTIFIED BY 'mypass';
```

- For `CREATE USER` or `GRANT`, you can avoid sending the plaintext password if you know the hash value that `PASSWORD()` [1188] would return for the password. Specify the hash value preceded by the keyword `PASSWORD`:

```
mysql> CREATE USER 'jeffrey'@'localhost'  
-> IDENTIFIED BY PASSWORD '*90E462C37378CED12064BB3388827D2BA3A9B689';
```

- When you assign an account a nonempty password using `SET PASSWORD`, you must use the `PASSWORD()` [1188] function to encrypt the password, otherwise the password is stored as plaintext. Suppose that you assign a password like this:

```
mysql> SET PASSWORD FOR
-> 'jeffrey'@'localhost' = 'mypass';
```

The result is that the literal value `'mypass'` is stored as the password in the `user` table, not the encrypted value. When `jeffrey` attempts to connect to the server using this password, the value is encrypted and compared to the value stored in the `user` table. However, the stored value is the literal string `'mypass'`, so the comparison fails and the server rejects the connection with an `Access denied` error.



Note

`PASSWORD()` [1188] encryption differs from Unix password encryption. See [Section 6.3.1, “User Names and Passwords”](#).

It is preferable to assign passwords using `SET PASSWORD`, `GRANT`, or `mysqladmin`, but it is also possible to modify the `user` table directly. In this case, you must also use `FLUSH PRIVILEGES` to cause the server to reread the grant tables. Otherwise, the change remains unnoticed by the server until you restart it.

- To establish a password for a new account, provide a value for the `Password` column:

```
mysql> INSERT INTO mysql.user (Host,User>Password)
-> VALUES('localhost','jeffrey',PASSWORD('mypass'));
mysql> FLUSH PRIVILEGES;
```

- To change the password for an existing account, use `UPDATE` to set the `Password` column value:

```
mysql> UPDATE mysql.user SET Password = PASSWORD('bagel')
-> WHERE Host = 'localhost' AND User = 'francis';
mysql> FLUSH PRIVILEGES;
```

6.3.6. Pluggable Authentication

When a client connects to the MySQL server, the server uses the user name provided by the client and the client host to select the appropriate account row from the `mysql.user` table. It then uses this row to authenticate the client.

Before MySQL 5.5.7, the server authenticates the password provided by the client against the `Password` column of the account row.

As of MySQL 5.5.7, the server authenticates clients using plugins. Selection of the proper account row from the `mysql.user` table is based on the user name and client host, as before, but the server authenticates the client credentials as follows:

- The server determines from the account row which authentication plugin applies for the client:
 - If the account row specifies no plugin name, the server uses native authentication; that is, authentication against the password stored in the `Password` column of the account row. This is the same authentication method provided by MySQL servers older than 5.5.7, before pluggable authentication was implemented, but now is implemented using two plugins that are built in and cannot be disabled.
 - If the account row specifies a plugin, the server invokes it to authenticate the user. If the server cannot find the plugin, an error occurs.
- The plugin returns a status to the server indicating whether the user is permitted to connect.

Pluggable authentication enables two important capabilities:

- **External authentication:** Pluggable authentication makes it possible for clients to connect to the MySQL server with credentials that are appropriate for authentication methods other than native authentication based on passwords stored in the `mysql.user` table. For example, plugins can be created to use external authentication methods such as PAM, Windows login IDs, LDAP, or Kerberos.
- **Proxy users:** If a user is permitted to connect, an authentication plugin can return to the server a user name different from the name of the connecting user, to indicate that the connecting user is a proxy for another user. While the connection lasts, the proxy user is treated, for purposes of access control, as having the privileges of a different user. In effect, one user impersonates another. For more information, see [Section 6.3.7, “Proxy Users”](#).

Several authentication plugins are available in MySQL. The following sections provide details about specific plugins.

- Plugins that perform native authentication that matches the password against the `Password` column of the account row. See [Section 6.3.6.1, “The Native Authentication Plugins”](#). Native authentication is the default for accounts that have no plugin named explicitly in their account row.
- A plugin that performs external authentication against PAM (Pluggable Authentication Modules), enabling MySQL Server to use PAM to authenticate MySQL users. This plugin supports proxy users as well. See [Section 6.3.6.2, “The PAM Authentication Plugin”](#).
- A plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. This plugin supports proxy users as well. See [Section 6.3.6.3, “The Windows Native Authentication Plugin”](#).
- A client-side plugin that sends the password to the server without hashing or encryption. This plugin can be used by server-side plugins that require access to the password exactly as provided by the client user. See [Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”](#).
- A plugin that authenticates clients that connect from the local host through the Unix socket file. See [Section 6.3.6.5, “The Socket Peer-Credential Authentication Plugin”](#).
- A test plugin that authenticates using MySQL native authentication. This plugin is intended for testing and development purposes, and as an example of how to write an authentication plugin. See [Section 6.3.6.6, “The Test Authentication Plugin”](#).

**Note**

For information about current restrictions on the use of pluggable authentication, including which connectors support which plugins, see [Section E.9, “Restrictions on Pluggable Authentication”](#).

Third-party connector developers should read that section to determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

If you are interested in writing your own authentication plugins, see [Section 23.2.4.9, “Writing Authentication Plugins”](#).

In general, pluggable authentication uses corresponding plugins on the server and client sides, so you use a given authentication method like this:

- On the server host, install the appropriate library containing the server plugin, if necessary, so that the server can use it to authenticate client connections. Similarly, on each client host, install the appropriate library containing the client plugin for use by client programs.
- Create MySQL accounts that specify use of the plugin for authentication.
- When a client connects, the server plugin tells the client program which client plugin to use for authentication.

The remainder of this section provides general instructions for installing and using authentication plugins. The instructions use an example authentication plugin included in MySQL distributions (see [Section 6.3.6.6, “The Test Authentication Plugin”](#)). The procedure is similar for other authentication plugins; substitute the appropriate plugin and file names.

The example authentication plugin has these characteristics:

- The server-side plugin name is `test_plugin_server`.
- The client-side plugin name is `auth_test_plugin`.
- Both plugins are located in the shared library object file named `auth_test_plugin.so` in the plugin directory (the directory named by the `plugin_dir` [537] system variable). The file name suffix might differ on your system.

Install and use the example authentication plugin as follows:

1. Make sure that the plugin library is installed on the server and client hosts.
2. Install the server-side test plugin at server startup or at runtime:
 - To install the plugin at startup, use the `--plugin-load` [457] option. For example, use these lines in a `my.cnf` option file:

```
[mysqld]
plugin-load=test_plugin_server=auth_test_plugin.so
```

With this plugin-loading method, the option must be given each time you start the server. The plugin is not installed if you omit the option.

- To install the plugin at runtime, use the `INSTALL PLUGIN` statement:

```
mysql> INSTALL PLUGIN test_plugin_server SONAME 'auth_test_plugin.so';
```

This installs the plugin permanently and need be done only once.

3. Verify that the plugin is installed. For example, use `SHOW PLUGINS`:

```
mysql> SHOW PLUGINS\G
...
***** 21. row *****
  Name: test_plugin_server
  Status: ACTIVE
  Type: AUTHENTICATION
  Library: auth_test_plugin.so
  License: GPL
```

For other ways to check the plugin, see [Section 5.1.8.2, “Obtaining Server Plugin Information”](#).

4. To specify that a MySQL user must be authenticated using the plugin, name it in the `IDENTIFIED WITH` clause of the `CREATE USER` statement that creates the user:

```
CREATE USER 'testuser'@'localhost' IDENTIFIED WITH test_plugin_server;
```

5. Connect to the server using a client program. The test plugin authenticates the same way as native MySQL authentication, so provide the usual `--user` [240] and `--password` [239] options that you normally use to connect to the server. For example:

```
shell> mysql --user=your_name --password=your_pass
```

For connections by `testuser`, the server sees that the account must be authenticated using the server-side plugin named `test_plugin_server` and communicates to the client program which client-side plugin it must use—in this case, `auth_test_plugin`.

In the case that the account uses the authentication method that is the default for both the server and the client program, the server need not communicate to the client which plugin to use, and a round trip in client/server negotiation can be avoided. Currently this is true for accounts that use native MySQL authentication (`mysql_native_password`).

The `--default-auth=plugin_name` [277] option can be specified on the `mysql` command line to make explicit which client-side plugin the program can expect to use, although the server will override this if the user account requires a different plugin.

If `mysql` does not find the plugin, specify a `--plugin-dir=dir_name` [280] option to indicate where the plugin is located.



Note

If you start the server with the `--skip-grant-tables` [461] option, authentication plugins are not used even if loaded because the server performs no client authentication and permits any client to connect. Because this is insecure, you might want to use `--skip-grant-tables` [461] in conjunction with `--skip-networking` [461] to prevent remote clients from connecting.

6.3.6.1. The Native Authentication Plugins

MySQL includes two plugins that implement the same kind of native authentication that older servers provide; that is, authentication against passwords stored in the `Password` column of the `mysql.user` table:

- The `mysql_native_password` authentication plugin implements the same default authentication against the `mysql.user` table as used prior to the implementation of pluggable authentication.
- The `mysql_old_password` plugin implements authentication as used before MySQL 4.1.1 that is based on shorter password hash values. For information about this authentication method, see [Section 6.1.2.4, “Password Hashing in MySQL”](#).

The native authentication plugins are backward compatible. Clients older than MySQL 5.5.7 do not support authentication plugins but use native authentication, so they can connect to servers from 5.5.7 and up.

The following tables show the plugin names. Both are considered to implement native authentication even though only one has “native” in the name.

Table 6.8. MySQL Native Password Authentication Plugin

Server-side plugin name	<code>mysql_native_password</code>
Client-side plugin name	<code>mysql_native_password</code>

Library object file name	None (plugins are built in)
--------------------------	-----------------------------

Table 6.9. MySQL Native Old-Password Authentication Plugin

Server-side plugin name	<code>mysql_old_password</code>
Client-side plugin name	<code>mysql_old_password</code>
Library object file name	None (plugins are built in)

Each plugin exists in both client and server form. MySQL client programs use `mysql_native_password` by default. The `--default-auth` [277] option can be used to specify either plugin explicitly:

```
shell> mysql --default-auth=mysql_native_password ...
shell> mysql --default-auth=mysql_old_password ...
```

The server-side plugins are built into the server, need not be loaded explicitly, and cannot be disabled by unloading them. The client-side plugins are built into the `libmysql` client library as of MySQL 5.5.7 and available to any program linked against `libmysql` from that version or newer.

For general information about pluggable authentication in MySQL, see [Section 6.3.6, “Pluggable Authentication”](#).

6.3.6.2. The PAM Authentication Plugin



Note

The PAM authentication plugin is a commercial extension. To learn more about commercial products (MySQL Enterprise Edition), see <http://www.mysql.com/products/>.

As of MySQL 5.5.16, commercial distributions of MySQL include an authentication plugin that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as Unix passwords or an LDAP directory.

The PAM plugin uses the information passed to it by MySQL Server (such as user name, host name, password, and authentication string), plus whatever method is available for PAM lookup. The plugin checks the user credentials against PAM and returns `'Authentication succeeded, Username is user_name'` or `'Authentication failed'`.

The PAM authentication plugin provides these capabilities:

- **External authentication:** The plugin enables MySQL Server to accept connections from users defined outside the MySQL grant tables.
- **Proxy user support:** The plugin can return to MySQL a user name different from the login user, based on the groups the external user is in and the authentication string provided. This means that the plugin can return the MySQL user that defines the privileges the external PAM-authenticated user should have. For example, a PAM user named `joe` can connect and have the privileges of the MySQL user named `developer`.

The following table shows the plugin and library file names. The file name suffix might be different on your system. The file location must be the directory named by the `plugin_dir` [537] system variable. For installation information, see [Section 6.3.6.2.1, “Installing the PAM Authentication Plugin”](#).

Table 6.10. MySQL PAM Authentication Plugin

Server-side plugin name	<code>authentication_pam</code>
Client-side plugin name	<code>mysql_clear_password</code>
Library object file name	<code>authentication_pam.so</code>

The library file includes only the server-side plugin. As of MySQL 5.5.10, the client-side plugin is built into the `libmysql` client library. See [Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”](#).

The server-side PAM authentication plugin is included only in commercial distributions. It is not included in MySQL community distributions. The client-side clear-text plugin that communicates with the server-side plugin is built into the MySQL client library and is included in all distributions, including community distributions. This permits clients from any 5.5.10 or newer distribution to connect to a server that has the server-side plugin loaded.

The PAM authentication plugin has been tested on Linux and Mac OS X. It requires MySQL Server 5.5.16 or newer.

For general information about pluggable authentication in MySQL, see [Section 6.3.6, “Pluggable Authentication”](#). For proxy user information, see [Section 6.3.7, “Proxy Users”](#).

6.3.6.2.1. Installing the PAM Authentication Plugin

The PAM authentication plugin must be located in the MySQL plugin directory (the directory named by the `plugin_dir` [537] system variable). If necessary, set the value of `plugin_dir` [537] at server startup to tell the server the location of the plugin directory.

To enable the plugin, start the server with the `--plugin-load` [457] option. For example, put the following lines in your `my.cnf` file. If object files have a suffix different from `.so` on your system, substitute the correct suffix.

```
[mysqld]
plugin-load=authentication_pam.so
```

Use the plugin name `authentication_pam` in the `IDENTIFIED WITH` clause of `CREATE USER` or `GRANT` statements for MySQL accounts that should be authenticated with this plugin.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement. See [Section 5.1.8.2, “Obtaining Server Plugin Information”](#).

6.3.6.2.2. Using the PAM Authentication Plugin

This section describes how to use the PAM authentication plugin to connect from MySQL client programs to the server. It is assumed that the server-side plugin is enabled and that client programs are recent enough to include the client-side plugin.



Note

The client-side plugin with which the PAM plugin communicates simply sends the password to the server in clear text so it can be passed to PAM. This may be a security problem in some configurations, but is necessary to use the server-side PAM library. To avoid problems if there is any possibility that the password would be intercepted, clients should connect to MySQL Server using SSL. See [Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”](#).

To refer to the PAM authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` or `GRANT` statement, use the name `authentication_pam`. For example:

```
CREATE USER user
  IDENTIFIED WITH authentication_pam
  AS 'authentication_string';
```

The authentication string specifies the following types of information:

- PAM supports the notion of “service name,” which is a name that the system administrator can use to configure the authentication method for a particular application. There can be several such “applications” associated with a single database server instance, so the choice of service name is left to the SQL application developer. When you define an account that should authenticate using PAM, specify the service name in the authentication string.
- PAM provides a way for a PAM module to return to the server a MySQL user name other than the login name supplied at login time. Use the authentication string to control the mapping between login name and MySQL user name. If you want to take advantage of proxy user capabilities, the authentication string must include this kind of mapping.

For example, if the service name is `mysql` and users in the `root` and `users` PAM groups should be mapped to the `developer` and `data_entry` users, respectively, use a statement like this:

```
CREATE USER user
  IDENTIFIED WITH authentication_pam
  AS 'mysql, root=developer, users=data_entry';
```

Authentication string syntax for the PAM authentication plugin follows these rules:

- The string consists of a PAM service name, optionally followed by a group mapping list consisting of one or more keyword/value pairs each specifying a group name and a SQL user name:

```
pam_service_name[,group_name=sql_user_name]...
```

- Each `group_name=sql_user_name` pair must be preceded by a comma.
- Leading and trailing spaces not inside double quotation marks are ignored.
- Unquoted `pam_service_name`, `group_name`, and `sql_user_name` values can contain anything except equal sign, comma, or space.
- If a `pam_service_name`, `group_name`, or `sql_user_name` value is quoted with double quotation marks, everything between the quotation marks is part of the value. This is necessary, for example, if the value contains space characters. All characters are legal except double quotation mark and backslash (`'\"`). To include either character, escape it with a backslash.

The plugin parses the authentication string on each login check. To minimize overhead, keep the string as short as possible.

If the plugin successfully authenticates a login name, it looks for a group mapping list in the authentication string and uses it to return a different user name to the MySQL server based on the groups the external user is a member of:

- If the authentication string contains no group mapping list, the plugin returns the login name.
- If the authentication string does contain a group mapping list, the plugin examines each `group_name=sql_user_name` pair in the list from left to right and tries to find a match for the `group_name` value in a non-MySQL directory of the groups assigned to the authenticated user and

returns `sql_user_name` for the first match it finds. If the plugin finds no match for any group, it returns the login name. If the plugin is not capable of looking up a group in a directory, it ignores the group mapping list and returns the login name.

The following sections describe how to set up several authentication scenarios that use the PAM authentication plugin:

- No proxy users. This uses PAM only to check login names and passwords. Every external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use external PAM authentication. Authentication can be performed by various PAM-supported methods. The discussion shows how to use traditional Unix passwords and LDAP.

PAM authentication, when not done through proxy users or groups, requires the MySQL account to have the same user name as the Unix account. Because MySQL user names are limited to 16 characters (see [Section 6.2.2, “Privilege System Grant Tables”](#)), this limits PAM nonproxy authentication to Unix accounts with names of at most 16 characters.

- Proxy login only and group mapping. For this scenario, create a few MySQL accounts that define different sets of privileges. (Ideally, nobody should log in through these directly.) Then define a default user authenticating through PAM that uses some mapping scheme (usually by the external groups the users are in) to map all the external logins to the few MySQL accounts holding the privilege sets. Any user that logs in is mapped to one of the MySQL accounts and uses its privileges. The discussion shows how to set this up using Unix passwords, but other PAM methods such as LDAP could be used instead.

Variations on these scenarios are possible. For example, you can permit some users to log in directly but require others to connect through proxy users.

The examples make the following assumptions. You might need to make some adjustments if your system is set up differently.

- The PAM configuration directory is `/etc/pam.d`.
- The PAM service name is `mysql`, which means that you must set up a PAM file named `mysql` in the PAM configuration directory (creating the file if it does not exist). If you use a different service name, the file name will be different and you must use a different name in the `AS` clause of `CREATE USER` and `GRANT` statements.
- The examples use a login name of `antonio` and password of `verysecret`. Change these to correspond to the users you want to authenticate.

The PAM authentication plugin checks at initialization time whether the `AUTHENTICATION_PAM_LOG` environment value is set. If so, the plugin enables logging of diagnostic messages to the standard output. These messages may be helpful for debugging PAM-related problems that occur when the plugin performs authentication. For more information, see [Section 6.3.6.2.3, “PAM Authentication Plugin Debugging”](#).

6.3.6.2.2.1. Unix Password Authentication without Proxy Users

This authentication scenario uses PAM only to check Unix user login names and passwords. Every external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use external PAM authentication.

1. Verify that Unix authentication in PAM permits you to log in as `antonio` with password `verysecret`.
2. Set up PAM to authenticate the `mysql` service. Put the following in `/etc/pam.d/mysql`:

```
##PAM-1.0
auth            include            password-auth
```

```
account      include      password-auth
```

3. Create a MySQL account with the same user name as the Unix login name and define it to authenticate using the PAM plugin:

```
CREATE USER 'antonio'@'localhost'
  IDENTIFIED WITH authentication_pam AS 'mysql';
GRANT ALL PRIVILEGES ON mydb.* TO 'antonio'@'localhost';
```

4. Try to connect to the MySQL server using the `mysql` command-line client. For example:

```
mysql --user=antonio --password=verysecret mydb
```

The server should permit the connection and the following query should return output as shown:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER()          | CURRENT_USER() | @@proxy_user |
+-----+-----+-----+
| antonio@localhost | antonio@localhost | NULL         |
+-----+-----+-----+
```

This shows that `antonio` uses the privileges granted to the `antonio` MySQL account, and that no proxying has occurred.

6.3.6.2.2.2. LDAP Authentication without Proxy Users

This authentication scenario uses PAM only to check LDAP user login names and passwords. Every external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use external PAM authentication.

1. Verify that LDAP authentication in PAM permits you to log in as `antonio` with password `verysecret`.
2. Set up PAM to authenticate the `mysql` service through LDAP. Put the following in `/etc/pam.d/mysql`:

```
##PAM-1.0
auth      required      pam_ldap.so
account   required      pam_ldap.so
```

If PAM object files have a suffix different from `.so` on your system, substitute the correct suffix.

3. MySQL account creation and connecting to the server is the same as previously described in [Section 6.3.6.2.2.1, “Unix Password Authentication without Proxy Users”](#).

6.3.6.2.2.3. Unix Password Authentication with Proxy Users and Group Mapping

This authentication scheme uses proxying and group mapping to map users who connect to the MySQL server through PAM onto a few MySQL accounts that define different sets of privileges. Users do not connect directly through the accounts that define the privileges. Instead, they connect through a default proxy user authenticating through PAM that uses a mapping scheme to map all the external logins to the few MySQL accounts holding the privileges. Any user who connects is mapped to one of the MySQL accounts and uses its privileges.

The procedure shown here uses Unix password authentication. To use LDAP instead, see the early steps of [Section 6.3.6.2.2.2, “LDAP Authentication without Proxy Users”](#).

1. Verify that Unix authentication in PAM permits you to log in as `antonio` with password `verysecret` and that `antonio` is a member of the `root` or `users` group.
2. Set up PAM to authenticate the `mysql` service. Put the following in `/etc/pam.d/mysql`:

```

#%PAM-1.0
auth            include            password-auth
account         include            password-auth

```

3. Create the default proxy user that maps the external PAM users to the proxied accounts. It maps external users from the `root` PAM group to the `developer` MySQL account and the external users from the `users` PAM group to the `data_entry` MySQL account:

```

CREATE USER ''@'
  IDENTIFIED WITH authentication_pam
  AS 'mysql, root=developer, users=data_entry';

```

The mapping list following the service name is required when you set up proxy users. Otherwise, the plugin cannot tell how to map the name of PAM groups to the proper proxied user name.

4. Create the proxied accounts that will be used to access the databases:

```

CREATE USER 'developer'@'localhost' IDENTIFIED BY 'very secret password';
GRANT ALL PRIVILEGES ON mydevdb.* TO 'developer'@'localhost';
CREATE USER 'data_entry'@'localhost' IDENTIFIED BY 'very secret password';
GRANT ALL PRIVILEGES ON mydb.* TO 'data_entry'@'localhost';

```

If you do not let anyone know the passwords for these accounts, other users cannot use them to connect directly to the MySQL server. Instead, it is expected that users will authenticate using PAM and that they will use the `developer` or `data_entry` account by proxy based on their PAM group.

5. Grant the `PROXY` [696] privilege to the proxy account for the proxied accounts:

```

GRANT PROXY ON 'developer'@'localhost' TO ''@';
GRANT PROXY ON 'data_entry'@'localhost' TO ''@';

```

6. Try to connect to the MySQL server using the `mysql` command-line client. For example:

```
mysql --user=antonio --password=verysecret mydb
```

The server authenticates the connection using the `''@'` account. The privileges `antonio` will have depends on what PAM groups he is a member of. If `antonio` is a member of the `root` PAM group, the PAM plugin maps `root` to the `developer` MySQL user name and returns that name to the server. The server verifies that `''@'` has the `PROXY` [696] privilege for `developer` and permits the connection. the following query should return output as shown:

```

mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER()          | CURRENT_USER()  | @@proxy_user |
+-----+-----+-----+
| antonio@localhost | developer@localhost | ''@'         |
+-----+-----+-----+

```

This shows that `antonio` uses the privileges granted to the `developer` MySQL account, and that proxying occurred through the default proxy user account.

If `antonio` is not a member of the `root` PAM group but is a member of the `users` group, a similar process occurs, but the plugin maps `user` group membership to the `data_entry` MySQL user name and returns that name to the server. In this case, `antonio` uses the privileges of the `data_entry` MySQL account:

```

mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER()          | CURRENT_USER()  | @@proxy_user |
+-----+-----+-----+
| antonio@localhost | data_entry@localhost | ''@'         |
+-----+-----+-----+

```

6.3.6.2.3. PAM Authentication Plugin Debugging

The PAM authentication plugin checks at initialization time whether the `AUTHENTICATION_PAM_LOG` environment value is set (the value does not matter). If so, the plugin enables logging of diagnostic messages to the standard output. These messages may be helpful for debugging PAM-related problems that occur when the plugin performs authentication.

Some messages include reference to PAM plugin source files and line numbers, which enables plugin actions to be tied more closely to the location in the code where they occur.

The following transcript demonstrates the kind of information produced by enabling logging. It resulted from a successful proxy authentication attempt.

```
entering auth_pam_server
entering auth_pam_next_token
auth_pam_next_token:reading at [cups,admin=writer,everyone=reader], sep=[,]
auth_pam_next_token:state=PRESPACE, ptr=[cups,admin=writer,everyone=reader],
out=[]
auth_pam_next_token:state=IDENT, ptr=[cups,admin=writer,everyone=reader],
out=[]
auth_pam_next_token:state=AFTERSPACE, ptr=[,admin=writer,everyone=reader],
out=[cups]
auth_pam_next_token:state=DELIMITER, ptr=[,admin=writer,everyone=reader],
out=[cups]
auth_pam_next_token:state=DONE, ptr=[,admin=writer,everyone=reader],
out=[cups]
leaving auth_pam_next_token on
/Users/gkodinov/mysql/work/x-5.5.16-release-basket/release/plugin/pam-authentication-plugin/src/parser.c:1
auth_pam_server:password 12345qq received
auth_pam_server:pam_start rc=0
auth_pam_server:pam_set_item(PAM_RUSER,gkodinov) rc=0
auth_pam_server:pam_set_item(PAM_RHOST,localhost) rc=0
entering auth_pam_server_conv
auth_pam_server_conv:PAM_PROMPT_ECHO_OFF [Password:] received
leaving auth_pam_server_conv on
/Users/gkodinov/mysql/work/x-5.5.16-release-basket/release/plugin/pam-authentication-plugin/src/authentication.c:1
auth_pam_server:pam_authenticate rc=0
auth_pam_server:pam_acct_mgmt rc=0
auth_pam_server:pam_setcred(PAM_ESTABLISH_CRED) rc=0
auth_pam_server:pam_get_item rc=0
auth_pam_server:pam_setcred(PAM_DELETE_CRED) rc=0
entering auth_pam_map_groups
entering auth_pam_walk_namevalue_list
auth_pam_walk_namevalue_list:reading at: [admin=writer,everyone=reader]
entering auth_pam_next_token
auth_pam_next_token:reading at [admin=writer,everyone=reader], sep=[=]
auth_pam_next_token:state=PRESPACE, ptr=[admin=writer,everyone=reader], out=[]
auth_pam_next_token:state=IDENT, ptr=[admin=writer,everyone=reader], out=[]
auth_pam_next_token:state=AFTERSPACE, ptr=[=writer,everyone=reader],
out=[admin]
auth_pam_next_token:state=DELIMITER, ptr=[=writer,everyone=reader],
out=[admin]
auth_pam_next_token:state=DONE, ptr=[=writer,everyone=reader], out=[admin]
leaving auth_pam_next_token on
/Users/gkodinov/mysql/work/x-5.5.16-release-basket/release/plugin/pam-authentication-plugin/src/parser.c:1
auth_pam_walk_namevalue_list:name=[admin]
entering auth_pam_next_token
auth_pam_next_token:reading at [writer,everyone=reader], sep=[,]
auth_pam_next_token:state=PRESPACE, ptr=[writer,everyone=reader], out=[]
auth_pam_next_token:state=IDENT, ptr=[writer,everyone=reader], out=[]
auth_pam_next_token:state=AFTERSPACE, ptr=[,everyone=reader], out=[writer]
auth_pam_next_token:state=DELIMITER, ptr=[,everyone=reader], out=[writer]
auth_pam_next_token:state=DONE, ptr=[,everyone=reader], out=[writer]
```

```

leaving auth_pam_next_token on
/Users/gkodinov/mysql/work/x-5.5.16-release-basket/release/plugin/pam-authentication-plugin/src/parser.c:191
walk, &error_namevalue_list:value=[writer]
entering auth_pam_map_group_to_user
auth_pam_map_group_to_user:pam_user=gkodinov, name=admin, value=writer
examining member root
examining member gkodinov
substitution was made to mysql user writer
leaving auth_pam_map_group_to_user on
/Users/gkodinov/mysql/work/x-5.5.16-release-basket/release/plugin/pam-authentication-plugin/src/authentication
auth_pam_walk_namevalue_list:found mapping
leaving auth_pam_walk_namevalue_list on
/Users/gkodinov/mysql/work/x-5.5.16-release-basket/release/plugin/pam-authentication-plugin/src/parser.c:270
auth_pam_walk_namevalue_list returned 0
leaving auth_pam_map_groups on
/Users/gkodinov/mysql/work/x-5.5.16-release-basket/release/plugin/pam-authentication-plugin/src/authentication
auth_pam_server:authenticated_as=writer
auth_pam_server: rc=0
leaving auth_pam_server on
/Users/gkodinov/mysql/work/x-5.5.16-release-basket/release/plugin/pam-authentication-plugin/src/authentication
    
```

6.3.6.3. The Windows Native Authentication Plugin



Note

The Windows authentication plugin is a commercial extension. To learn more about commercial products (MySQL Enterprise Edition), see <http://www.mysql.com/products/>.

As of MySQL 5.5.16, commercial distributions of MySQL for Windows include an authentication plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password.

The client and server exchange data packets in the authentication handshake. As a result of this exchange, the server creates a security context object that represents the identity of the client in the Windows OS. This identity includes the name of the client account. The Windows authentication plugin uses the identity of the client to check whether it is a given account or a member of a group. By default, negotiation uses Kerberos to authenticate, then NTLM if Kerberos is unavailable.

The Windows authentication plugin provides these capabilities:

- External authentication: The plugin enables MySQL Server to accept connections from users defined outside the MySQL grant tables.
- Proxy user support: The plugin can return to MySQL a user name different from the client user. This means that the plugin can return the MySQL user that defines the privileges the external Windows-authenticated user should have. For example, a Windows user named `joe` can connect and have the privileges of the MySQL user named `developer`.

The following table shows the plugin and library file names. The file location must be the directory named by the `plugin_dir` [537] system variable. For installation information, see [Section 6.3.6.3.1, “Installing the Windows Authentication Plugin”](#).

Table 6.11. MySQL Windows Authentication Plugin

Server-side plugin name	<code>authentication_windows</code>
Client-side plugin name	<code>authentication_windows_client</code>

Library object file name	<code>authentication_windows.dll</code>
--------------------------	---

The library file includes only the server-side plugin. As of MySQL 5.5.13, the client-side plugin is built into the `libmysql` client library.

The server-side Windows authentication plugin is included only in commercial distributions. It is not included in MySQL community distributions. The client-side plugin is included in all distributions, including community distributions. This permits clients from any 5.5.13 or newer distribution to connect to a server that has the server-side plugin loaded.

The Windows authentication plugin should work on Windows 2000 Professional and up. It requires MySQL Server 5.5.16 or newer.

For general information about pluggable authentication in MySQL, see [Section 6.3.6, “Pluggable Authentication”](#). For proxy user information, see [Section 6.3.7, “Proxy Users”](#).

6.3.6.3.1. Installing the Windows Authentication Plugin

The Windows authentication plugin must be installed in the MySQL plugin directory (the directory named by the `plugin_dir` [537] system variable). If necessary, set the value of `plugin_dir` [537] at server startup to tell the server the location of the plugin directory.

To enable the plugin, start the server with the `--plugin-load` [457] option. For example, put these lines in your `my.ini` file:

```
[mysqld]
plugin-load=authentication_windows.dll
```

Use the plugin name `authentication_windows` in the `IDENTIFIED WITH` clause of `CREATE USER` or `GRANT` statements for MySQL accounts that should be authenticated with this plugin.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement. See [Section 5.1.8.2, “Obtaining Server Plugin Information”](#).

6.3.6.3.2. Using the Windows Authentication Plugin

The Windows authentication plugin supports the use of MySQL accounts such that users who have logged in to Windows can connect to the MySQL server without having to specify an additional password. It is assumed that the server-side plugin is enabled and that client programs are recent enough to include the client-side plugin built into `libmysql` (MySQL 5.5.13 or higher). Once the DBA has enabled the server-side plugin and set up accounts to use it, clients can connect using those accounts with no other setup required on their part.

To refer to the Windows authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` or `GRANT` statement, use the name `authentication_windows`. Suppose that the Windows users `Rafal` and `Tasha` should be permitted to connect to MySQL, as well as any users in the `Administrators` or `Power Users` group. To set this up, create a MySQL account named `sql_admin` that uses the Windows plugin for authentication:

```
CREATE USER sql_admin
  IDENTIFIED WITH authentication_windows
  AS 'Rafal, Tasha, Administrators, "Power Users"';
```

The plugin name is `authentication_windows`. The string following the `AS` keyword is the authentication string. It specifies that the Windows users named `Rafal` or `Tasha` are permitted to authenticate to the server as the MySQL user `sql_admin`, as are any Windows users in the `Administrators` or `Power Users` group. The latter group name contains a space, so it must be quoted with double quote characters.

After you create the `sql_admin` account, a user who has logged in to Windows can attempt to connect to the server using that account:

```
C:\> mysql --user=sql_admin
```

No password is required here. The `authentication_windows` plugin uses the Windows security API to check which Windows user is connecting. If that user is named `Rafal` or `Tasha`, or is in the `Administrators` or `Power Users` group, the server grants access and the client is authenticated as `sql_admin` and has whatever privileges are granted to the `sql_admin` account. Otherwise, the server denies access.

Authentication string syntax for the Windows authentication plugin follows these rules:

- The string consists of one or more user mappings separated by commas.
- Each user mapping associates a Windows user or group name with a MySQL user name:

```
win_user_or_group_name=sql_user_name
win_user_or_group_name
```

For the latter syntax, with no `sql_user_name` value given, the implicit value is the MySQL user created by the `CREATE USER` statement. Thus, these statements are equivalent:

```
CREATE USER sql_admin
  IDENTIFIED WITH authentication_windows
  AS 'Rafal, Tasha, Administrators, "Power Users"';

CREATE USER sql_admin
  IDENTIFIED WITH authentication_windows
  AS 'Rafal=sql_admin, Tasha=sql_admin, Administrators=sql_admin,
     "Power Users"=sql_admin';
```

- Each backslash (`'\'`) in a value must be doubled because backslash is the escape character in MySQL strings.
- Leading and trailing spaces not inside double quotation marks are ignored.
- Unquoted `win_user_or_group_name` and `sql_user_name` values can contain anything except equal sign, comma, or space.
- If a `win_user_or_group_name` and or `sql_user_name` value is quoted with double quotation marks, everything between the quotation marks is part of the value. This is necessary, for example, if the name contains space characters. All characters within double quotes are legal except double quotation mark and backslash. To include either character, escape it with a backslash.
- `win_user_or_group_name` values use conventional syntax for Windows principals, either local or in a domain. Examples (note the doubling of backslashes):

```
domain\\user
.\user
domain\\group
.\group
BUILTIN\WellKnownGroup
```

When invoked by the server to authenticate a client, the plugin scans the authentication string left to right for a user or group match to the Windows user. If there is a match, the plugin returns the corresponding `sql_user_name` to the MySQL server. If there is no match, authentication fails.

A user name match takes preference over a group name match. Suppose that the Windows user named `win_user` is a member of `win_group` and the authentication string looks like this:

```
'win_group = sql_user1, win_user = sql_user2'
```

When `win_user` connects to the MySQL server, there is a match both to `win_group` and to `win_user`. The plugin authenticates the user as `sql_user2` because the more-specific user match takes precedence over the group match, even though the group is listed first in the authentication string.

Windows authentication always works for connections from the same computer on which the server is running. For cross-computer connections, both computers must be registered with Windows Active Directory. If they are in the same Windows domain, it is unnecessary to specify a domain name. It is also possible to permit connections from a different domain, as in this example:

```
CREATE USER sql_accounting
  IDENTIFIED WITH authentication_windows
  AS 'SomeDomain\\Accounting';
```

Here `SomeDomain` is the name of the other domain. The backslash character is doubled because it is the MySQL escape character within strings.

MySQL supports the concept of proxy users whereby a client can connect and authenticate to the MySQL server using one account but while connected has the privileges of another account (see [Section 6.3.7, “Proxy Users”](#)). Suppose that you want Windows users to connect using a single user name but be mapped based on their Windows user and group names onto specific MySQL accounts as follows:

- The `local_user` and `MyDomain\domain_user` local and domain Windows users should map to the `local_wlad` MySQL account.
- Users in the `MyDomain\Developers` domain group should map to the `local_dev` MySQL account.
- Local machine administrators should map to the `local_admin` MySQL account.

To set this up, create a proxy account for Windows users to connect to, and configure this account so that users and groups map to the appropriate MySQL accounts (`local_wlad`, `local_dev`, `local_admin`). In addition, grant the MySQL accounts the privileges appropriate to the operations they need to perform. The following instructions use `win_proxy` as the proxy account, and `local_wlad`, `local_dev`, and `local_admin` as the proxied accounts.

1. Create the proxy MySQL account:

```
CREATE USER win_proxy
  IDENTIFIED WITH authentication_windows
  AS 'local_user = local_wlad,
  MyDomain\\domain_user = local_wlad,
  MyDomain\\Developers = local_dev,
  BUILTIN\Administrators = local_admin';
```

2. For proxying to work, the proxied accounts must exist, so create them:

```
CREATE USER local_wlad IDENTIFIED BY 'wlad_pass';
CREATE USER local_dev IDENTIFIED BY 'dev_pass';
CREATE USER local_admin IDENTIFIED BY 'admin_pass';
```

If you do not let anyone know the passwords for these accounts, other users cannot use them to connect directly to the MySQL server.

You should also issue `GRANT` statements (not shown) that grant each proxied account the privileges it needs.

3. The proxy account must have the `PROXY [696]` privilege for each of the proxied accounts:

```
GRANT PROXY ON local_wlad TO win_proxy;
```

```
GRANT PROXY ON local_dev TO win_proxy;
GRANT PROXY ON local_admin TO win_proxy;
```

Now the Windows users `local_user` and `MyDomain\domain_user` can connect to the MySQL server as `win_proxy` and when authenticated have the privileges of the account given in the authentication string—in this case, `local_wlad`. A user in the `MyDomain\Developers` group who connects as `win_proxy` has the privileges of the `local_dev` account. A user in the `BUILTIN\Administrators` group has the privileges of the `local_admin` account.

To configure authentication so that all Windows users who do not have their own MySQL account go through a proxy account, substitute the default proxy user (`'@'`) for `win_proxy` in the preceding instructions. For information about the default proxy user, see [Section 6.3.7, “Proxy Users”](#).

To use the Windows authentication plugin with Connector/Net connection strings in Connector/Net 6.4.4 and higher, see [Section 22.2.5.5, “Using the Windows Native Authentication Plugin”](#).

Additional control over the Windows authentication plugin is provided by the `authentication_windows_use_principal_name` [483] and `authentication_windows_log_level` [482] system variables. See [Section 5.1.4, “Server System Variables”](#).

6.3.6.4. The Cleartext Client-Side Authentication Plugin

As of MySQL 5.5.10, a client-side authentication plugin is available that sends the password to the server without hashing or encryption. This plugin is built into the MySQL client library.

The following table shows the plugin name.

Table 6.12. MySQL Cleartext Authentication Plugin

Server-side plugin name	None, see discussion
Client-side plugin name	<code>mysql_clear_password</code>
Library object file name	None (plugin is built in)

With native MySQL authentication, the client performs one-way hashing on the password before sending it to the server. This enables the client to avoid sending the password in clear text. See [Section 6.1.2.4, “Password Hashing in MySQL”](#). However, because the hash algorithm is one way, the original password cannot be recovered on the server side.

One-way hashing cannot be done for authentication schemes that require the server to receive the password as entered on the client side. In such cases, the `mysql_clear_password` client-side plugin can be used to send the password to the server in clear text. There is no corresponding server-side plugin. Rather, the client-side plugin can be used by any server-side plugin that needs a clear text password.

For general information about pluggable authentication in MySQL, see [Section 6.3.6, “Pluggable Authentication”](#).



Note

Sending passwords in clear text may be a security problem in some configurations. To avoid problems if there is any possibility that the password would be intercepted, clients should connect to MySQL Server using a method that protects the password. Possibilities include SSL (see [Section 6.3.8, “Using SSL for Secure Connections”](#)), IPsec, or a private network.

As of MySQL 5.5.27, to make inadvertent use of this plugin less likely, it is required that clients explicitly enable it. This can be done several ways:

- Set the `LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN` environment variable to a value that begins with `1`, `Y`, or `y`. This enables the plugin for all client connections.
- The `mysql`, `mysqladmin`, and `mysqlslap` client programs support an `--enable-cleartext-plugin` option that enables the plugin on a per-invocation basis.
- The `mysql_options()` C API function supports a `MYSQL_ENABLE_CLEARTEXT_PLUGIN` option that enables the plugin on a per-connection basis. Also, any program that uses `libmysqlclient` and reads option files can enable the plugin by including an `enable-cleartext-plugin` option in an option group read by the client library.

6.3.6.5. The Socket Peer-Credential Authentication Plugin

As of MySQL 5.5.10, a server-side authentication plugin is available that authenticates clients that connect from the local host through the Unix socket file. This plugin works only on Linux systems.

The source code for this plugin can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file location is the directory named by the `plugin_dir` [537] system variable. For installation information, see [Section 6.3.6, “Pluggable Authentication”](#).

Table 6.13. MySQL Socket Peer-Credential Authentication Plugin

Server-side plugin name	<code>auth_socket</code>
Client-side plugin name	None, see discussion
Library object file name	<code>auth_socket.so</code>

The `auth_socket` authentication plugin authenticates clients that connect from the local host through the Unix socket file. The plugin uses the `SO_PEERCREC` socket option to obtain information about the user running the client program. The plugin checks whether the user name matches the MySQL user name specified by the client program to the server, and permits the connection only if the names match. The plugin can be built only on systems that support the `SO_PEERCREC` option, such as Linux.

Suppose that a MySQL account is created for a user named `valerie` who is to be authenticated by the `auth_socket` plugin for connections from the local host through the socket file:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
```

If a user on the local host with a login name of `stefanie` invokes `mysql` with the option `--user=valerie` to connect through the socket file, the server uses `auth_socket` to authenticate the client. The plugin determines that the `--user` option value (`valerie`) differs from the client user's name (`stefanie`) and refuses the connection. If a user named `valerie` tries the same thing, the plugin finds that the user name and the MySQL user name are both `valerie` and permits the connection. However, the plugin refuses the connection even for `valerie` if the connection is made using a different protocol, such as TCP/IP.

For general information about pluggable authentication in MySQL, see [Section 6.3.6, “Pluggable Authentication”](#).

6.3.6.6. The Test Authentication Plugin

MySQL includes a test plugin that authenticates using MySQL native authentication, but is a loadable plugin (not built in) and must be installed prior to use. It can authenticate against either normal or older (shorter) password hash values.

This plugin is intended for testing and development purposes, and not for use in production environments. The test plugin source code is separate from the server source, unlike the built-in native plugin, so it can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file location is the directory named by the `plugin_dir` [537] system variable. For installation information, see [Section 6.3.6, “Pluggable Authentication”](#).

Table 6.14. MySQL Test Authentication Plugin

Server-side plugin name	<code>test_plugin_server</code>
Client-side plugin name	<code>auth_test_plugin</code>
Library object file name	<code>auth_test_plugin.so</code>

Because the test plugin authenticates the same way as native MySQL authentication, provide the usual `--user` [240] and `--password` [239] options that you normally use for accounts that use native authentication when you connect to the server. For example:

```
shell> mysql --user=your_name --password=your_pass
```

For general information about pluggable authentication in MySQL, see [Section 6.3.6, “Pluggable Authentication”](#).

6.3.7. Proxy Users

When authentication to the MySQL server occurs by means of an authentication plugin, the plugin may request that the connecting (external) user be treated as a different user for privilege-checking purposes. This enables the external user to be a proxy for the second user; that is, to have the privileges of the second user. In other words, the external user is a “proxy user” (a user who can impersonate or become known as another user) and the second user is a “proxied user” (a user whose identity can be taken on by a proxy user).

This section describes how the proxy user capability works. For general information about authentication plugins, see [Section 6.3.6, “Pluggable Authentication”](#). If you are interested in writing your own authentication plugins that support proxy users, see [Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”](#).

For proxying to occur, these conditions must be satisfied:

- When a connecting client should be treated as a proxy user, the plugin must return a different name, to indicate the proxied user name.
- A proxy user account must be set up to be authenticated by the plugin. Use the `CREATE USER` or `GRANT` statement to associate an account with a plugin.
- A proxy user account must have the `PROXY` [696] privilege for the proxied account. Use the `GRANT` statement for this.

Consider the following definitions:

```
CREATE USER 'empl_external'@'localhost'
  IDENTIFIED WITH auth_plugin AS 'auth_string';
CREATE USER 'employee'@'localhost'
  IDENTIFIED BY 'employee_pass';
GRANT PROXY
  ON 'employee'@'localhost'
```

```
TO 'empl_external'@'localhost';
```

When a client connects as `empl_external` from the local host, MySQL uses `auth_plugin` to perform authentication. If `auth_plugin` returns the `employee` user name to the server (based on the content of `'auth_string'` and perhaps by consulting some external authentication system), that serves as a request to the server to treat this client, for purposes of privilege checking, as the `employee` local user.

In this case, `empl_external` is the proxy user and `employee` is the proxied user.

The server verifies that proxy authentication for `employee` is possible for the `empl_external` user by checking whether `empl_external` has the `PROXY [696]` privilege for `employee`. (If this privilege had not been granted, an error would occur.)

When proxying occurs, the `USER()` and `CURRENT_USER()` functions can be used to see the difference between the connecting user and the account whose privileges apply during the current session. For the example just described, those functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER()          | CURRENT_USER() |
+-----+-----+
| empl_external@localhost | employee@localhost |
+-----+-----+
```

The `IDENTIFIED WITH` clause that names the authentication plugin may be followed by an `AS` clause specifying a string that the server passes to the plugin when the user connects. It is up to each plugin whether the `AS` clause is required. If it is required, the format of the authentication string depends on how the plugin intends to use it. Consult the documentation for a given plugin for information about the authentication string values it accepts.

Granting the Proxy Privilege

A special `PROXY [696]` privilege is needed to enable an external user to connect as and have the privileges of another user. To grant this privilege, use the `GRANT` statement. For example:

```
GRANT PROXY ON 'proxied_user' TO 'proxy_user';
```

`proxy_user` must represent a valid externally authenticated MySQL user at connection time or connection attempts fail. `proxied_user` must represent a valid locally authenticated user at connection time or connection attempts fail.

The corresponding `REVOKE` syntax is:

```
REVOKE PROXY ON 'proxied_user' FROM 'proxy_user';
```

MySQL `GRANT` and `REVOKE` syntax extensions work as usual. For example:

```
GRANT PROXY ON 'a' TO 'b', 'c', 'd';
GRANT PROXY ON 'a' TO 'd' IDENTIFIED BY ...;
GRANT PROXY ON 'a' TO 'd' WITH GRANT OPTION;
GRANT PROXY ON 'a' TO '@';
REVOKE PROXY ON 'a' FROM 'b', 'c', 'd';
```

In the preceding example, `'@'` is the default proxy user and means “any user.” The default proxy user is discussed later in this section.

The `PROXY [696]` privilege can be granted in these cases:

- By `proxied_user` for itself: The value of `USER()` must exactly match `CURRENT_USER()` and `proxied_user`, for both the user name and host name parts of the account name.

- By a user that has `GRANT PROXY ... WITH GRANT OPTION` for *proxied_user*.

The `root` account created by default during MySQL installation has the `PROXY ... WITH GRANT OPTION` [696] privilege for `'@'`, that is, for all users. This enables `root` to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. For example, `root` can do this:

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'test';
GRANT PROXY ON '@' TO 'admin'@'localhost' WITH GRANT OPTION;
```

Now the `admin` user can manage all the specific `GRANT PROXY` mappings. For example, `admin` can do this:

```
GRANT PROXY ON sally TO joe;
```

Default Proxy Users

To specify that some or all users should connect using a given external plugin, create a “blank” MySQL user, set it up to use that plugin for authentication, and let the plugin return the real authenticated user name (if different from the blank user). For example, suppose that there exists a hypothetical plugin named `ldap_auth` that implements LDAP authentication:

```
CREATE USER '@' IDENTIFIED WITH ldap_auth AS 'O=Oracle, OU=MySQL';
CREATE USER 'developer'@'localhost' IDENTIFIED BY 'developer_pass';
CREATE USER 'manager'@'localhost' IDENTIFIED BY 'manager_pass';
GRANT PROXY ON 'manager'@'localhost' TO '@';
GRANT PROXY ON 'developer'@'localhost' TO '@';
```

Now assume that a client tries to connect as follows:

```
mysql --user=myuser --password='myuser_pass' ...
```

The server will not find `myuser` defined as a MySQL user. But because there is a blank user account (`'@'`), that matches the client user name and host name, the server authenticates the client against that account: The server invokes `ldap_auth`, passing it `myuser` and `myuser_pass` as the user name and password.

If the `ldap_auth` plugin finds in the LDAP directory that `myuser_pass` is not the correct password for `myuser`, authentication fails and the server rejects the connection.

If the password is correct and `ldap_auth` finds that `myuser` is a developer, it returns the user name `developer` to the MySQL server, rather than `myuser`. The server verifies that `'@'` can authenticate as `developer` (because it has the `PROXY` [696] privilege to do so) and accepts the connection. The session proceeds with `myuser` having the privileges of `developer`. (These privileges should be set up by the DBA using `GRANT` statements, not shown.) The `USER()` [1197] and `CURRENT_USER()` [1192] functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER() | CURRENT_USER() |
+-----+-----+
| myuser@localhost | developer@localhost |
+-----+-----+
```

If the plugin instead finds in the LDAP directory that `myuser` is a manager, it returns `manager` as the user name and the session proceeds with `myuser` having the privileges of `manager`.

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER() | CURRENT_USER() |
+-----+-----+
```

```
+-----+-----+
| myuser@localhost | manager@localhost |
+-----+-----+
```

For simplicity, external authentication cannot be multilevel: Neither the credentials for `developer` nor those for `manager` are taken into account in the preceding example. However, they are still used if a client tries to authenticate directly against the `developer` or `manager` account, which is why those accounts should be assigned passwords.

The default proxy account uses `'` in the host part, which matches any host. If you set up a default proxy user, take care to also check for accounts with `'%` in the host part, because that also matches any host, but has precedence over `'` by the rules that the server uses to sort account rows internally (see [Section 6.2.4, “Access Control, Stage 1: Connection Verification”](#)).

Suppose that a MySQL installation includes these two accounts:

```
CREATE USER ''@' IDENTIFIED WITH some_plugin;
CREATE USER ''@%' IDENTIFIED BY 'some_password';
```

The intent of the first account is to serve as the default proxy user, to be used to authenticate connections for users who do not otherwise match a more-specific account. The second account might have been created, for example, to enable users without their own account as the anonymous user.

However, in this configuration, the first account will never be used because the matching rules sort `'@%'` ahead of `'@'`. For accounts that do not match any more-specific account, the server will attempt to authenticate them against `'@%'` rather than `'@'`.

If you intend to create a default proxy user, check for other existing “match any user” accounts that will take precedence over the default proxy user and thus prevent that user from working as intended. It may be necessary to remove any such accounts.

Proxy User System Variables

Two system variables help trace the proxy login process:

- `proxy_user` [539]: This value is `NULL` if proxying is not used. Otherwise, it indicates the proxy user account. For example, if a client authenticates through the default proxy account, this variable will be set as follows:

```
mysql> SELECT @@proxy_user;
+-----+
| @@proxy_user |
+-----+
| ''@'         |
+-----+
```

- `external_user` [499]: Sometimes the authentication plugin may use an external user to authenticate to the MySQL server. For example, when using Windows native authentication, a plugin that authenticates using the windows API does not need the login ID passed to it. However, it still uses an Windows user ID to authenticate. The plugin may return this external user ID (or the first 512 UTF-8 bytes of it) to the server using the `external_user` read-only session variable. If the plugin does not set this variable, its value is `NULL`.

6.3.8. Using SSL for Secure Connections

MySQL supports secure (encrypted) connections between MySQL clients and the server using the Secure Sockets Layer (SSL) protocol. This section discusses how to use SSL connections. For information on

how to require users to use SSL connections, see the discussion of the [REQUIRE](#) clause of the [GRANT](#) statement in [Section 13.7.1.3, “GRANT Syntax”](#).

The standard configuration of MySQL is intended to be as fast as possible, so encrypted connections are not used by default. For applications that require the security provided by encrypted connections, the extra computation to encrypt the data is worthwhile.

MySQL enables encryption on a per-connection basis. You can choose an unencrypted connection or a secure encrypted SSL connection according to the requirements of individual applications.

Secure connections are based on the OpenSSL API and are available through the MySQL C API. Replication uses the C API, so secure connections can be used between master and slave servers. See [Section 16.3.7, “Setting Up Replication Using SSL”](#).

Another way to connect securely is from within an SSH connection to the MySQL server host. For an example, see [Section 6.3.9, “Connecting to MySQL Remotely from Windows with SSH”](#).

6.3.8.1. Basic SSL Concepts

To understand how MySQL uses SSL, it is necessary to explain some basic SSL and X509 concepts. People who are familiar with these concepts can skip this part of the discussion.

By default, MySQL uses unencrypted connections between the client and the server. This means that someone with access to the network could watch all your traffic and look at the data being sent or received. They could even change the data while it is in transit between client and server. To improve security a little, you can compress client/server traffic by using the `--compress` option when invoking client programs. However, this does not foil a determined attacker.

When you need to move information over a network in a secure fashion, an unencrypted connection is unacceptable. Encryption is the way to make any kind of data unreadable. Encryption algorithms must include security elements to resist many kinds of known attacks such as changing the order of encrypted messages or replaying data twice.

SSL is a protocol that uses different encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect any data change, loss, or replay. SSL also incorporates algorithms that provide identity verification using the X509 standard.

X509 makes it possible to identify someone on the Internet. It is most commonly used in e-commerce applications. In basic terms, there should be some entity called a “Certificate Authority” (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can show the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted with this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

For more information about SSL, X509, encryption, or public-key cryptography, perform an Internet search for the keywords in which you are interested.

6.3.8.2. Configuring MySQL for SSL

To use SSL connections between the MySQL server and client programs, your system must support either OpenSSL or yaSSL, and your version of MySQL must be built with SSL support. To make it easier to use secure connections, MySQL is bundled with yaSSL, which uses the same licensing model as MySQL. (OpenSSL uses an Apache-style license.) yaSSL support is available on all MySQL platforms supported by Oracle Corporation.

To get secure connections to work with MySQL and SSL, you must do the following:

1. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, and you are going to use OpenSSL rather than the bundled yaSSL library, install OpenSSL if it has not already been installed. We have tested MySQL with OpenSSL 0.9.6. To obtain OpenSSL, visit <http://www.openssl.org>.

Building MySQL using OpenSSL requires a shared OpenSSL library, otherwise linker errors occur. Alternatively, build MySQL using yaSSL.

2. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, configure a MySQL source distribution to use SSL. When you configure MySQL, invoke `CMake` like this:

```
shell> cmake . -DWITH_SSL=bundled
```

That command configures the distribution to use the bundled yaSSL library. To use the system SSL library instead, specify the option like this instead:

```
shell> cmake . -DWITH_SSL=system
```

See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

Then compile and install the distribution.

On Unix platforms, yaSSL retrieves true random numbers from either either `/dev/urandom` or `/dev/random`. [Bug#13164](#) lists workarounds for some very old platforms which do not support these devices.

3. To check whether a `mysqld` server supports SSL, examine the value of the `have_ssl` [505] system variable:

```
mysql> SHOW VARIABLES LIKE 'have_ssl';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_ssl      | YES   |
+-----+-----+
```

If the value is `YES`, the server supports SSL connections. If the value is `DISABLED`, the server is capable of supporting SSL connections but was not started with the appropriate `--ssl-xxx` options to enable them to be used; see [Section 6.3.8.3, “Using SSL Connections”](#).

6.3.8.3. Using SSL Connections

To enable SSL connections, your MySQL distribution must be built with SSL support, as described in [Section 6.3.8.2, “Configuring MySQL for SSL”](#). In addition, the proper SSL-related options must be used to specify the appropriate certificate and key files. For a complete list of SSL options, see [Section 6.3.8.4, “SSL Command Options”](#).

To start the MySQL server so that it permits clients to connect using SSL, use the options that identify the certificate and key files the server uses when establishing a secure connection:

- `--ssl-ca` [750] identifies the Certificate Authority (CA) certificate.
- `--ssl-cert` [750] identifies the server public key certificate. This can be sent to the client and authenticated against the CA certificate that it has.
- `--ssl-key` [751] identifies the server private key.

For example, start the server like this:

```
shell> mysqld --ssl-ca=ca-cert.pem \
```

```
--ssl-cert=server-cert.pem \  
--ssl-key=server-key.pem
```

Each option names a file in PEM format. For instructions on generating the required SSL certificate and key files, see [Section 6.3.8.5, “Setting Up SSL Certificates and Keys for MySQL”](#). If you have a MySQL source distribution, you can also test your setup using the demonstration certificate and key files in the `mysql-test/std_data` directory of the distribution.

Similar options are used on the client side, although in this case, `--ssl-cert` [750] and `--ssl-key` [751] identify the client public and private key. Note that the Certificate Authority certificate, if specified, must be the same as used by the server.

To establish a secure connection to a MySQL server with SSL support, the options that a client must specify depend on the SSL requirements of the MySQL account used by the client. (See the discussion of the `REQUIRE` clause in [Section 13.7.1.3, “GRANT Syntax”](#).)

Suppose that you want to connect using an account that has no special SSL requirements or was created using a `GRANT` statement that includes the `REQUIRE SSL` option. As a recommended set of SSL options, start the server with at least `--ssl-cert` [750] and `--ssl-key` [751], and invoke the client with `--ssl-ca` [750]. A client can connect securely like this:

```
shell> mysql --ssl-ca=ca-cert.pem
```

To require that a client certificate also be specified, create the account using the `REQUIRE X509` option. Then the client must also specify the proper client key and certificate files or the server will reject the connection:

```
shell> mysql --ssl-ca=ca-cert.pem \  
--ssl-cert=client-cert.pem \  
--ssl-key=client-key.pem
```

A client can determine whether the current connection with the server uses SSL by checking the value of the `Ssl_cipher` [608] status variable. The value of `Ssl_cipher` [608] is nonempty if SSL is used, and empty otherwise. For example:

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';  
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| Ssl_cipher    | DHE-RSA-AES256-SHA |  
+-----+-----+
```

For the `mysql` client, an alternative is to use the `STATUS` or `\s` command and check the `SSL` line:

```
mysql> \s  
...  
SSL:                Not in use  
...
```

Or:

```
mysql> \s  
...  
SSL:                Cipher in use is DHE-RSA-AES256-SHA  
...
```

The C API enables application programs to use SSL:

- To establish a secure connection, program, use the `mysql_ssl_set()` C API function to set the appropriate certificate options before calling `mysql_real_connect()`. See [Section 22.8.3.67, “mysql_ssl_set\(\)”](#).

- To determine whether SSL is in use after the connection is established, use `mysql_get_ssl_cipher()`. A non-NULL return value indicates a secure connection and names the SSL cipher used for encryption. A NULL return value indicates that SSL is not being used. See [Section 22.8.3.33, “mysql_get_ssl_cipher\(\)”](#).

Replication uses the C API, so secure connections can be used between master and slave servers. See [Section 16.3.7, “Setting Up Replication Using SSL”](#).

6.3.8.4. SSL Command Options

This section describes options that are used to specify whether to use SSL and the names of SSL certificate and key files. These options can be given on the command line or in an option file. They are not available unless MySQL has been built with SSL support. See [Section 6.3.8.2, “Configuring MySQL for SSL”](#).

Table 6.15. SSL Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
have_openssl [504]			Yes		Global	No
have_ssl [505]			Yes		Global	No
skip-ssl [749]	Yes	Yes				
ssl [749]	Yes	Yes				
ssl-ca [750]	Yes	Yes			Global	No
- Variable: <code>ssl_ca</code>			Yes		Global	No
ssl-capath [750]	Yes	Yes			Global	No
- Variable: <code>ssl_capath</code>			Yes		Global	No
ssl-cert [750]	Yes	Yes			Global	No
- Variable: <code>ssl_cert</code>			Yes		Global	No
ssl-cipher [750]	Yes	Yes			Global	No
- Variable: <code>ssl_cipher</code>			Yes		Global	No
ssl-key [751]	Yes	Yes			Global	No
- Variable: <code>ssl_key</code>			Yes		Global	No
ssl-verify-server-cert [751]	Yes	Yes				

- `--ssl [749]`

For the server, this option specifies that the server permits SSL connections. For a client program, it permits the client to connect to the server using SSL, but this option is not sufficient in itself to cause an SSL connection to be used. As a recommended set of options to enable SSL connections, use at least `--ssl-cert [750]` and `--ssl-key [751]` on the server side and `--ssl-ca [750]` on the client side.

`--ssl [749]` is implied by other `--ssl-xxx` options as indicated in the descriptions for those options. For this reason, `--ssl [749]` is not usually specified explicitly. It is more often used explicitly in its opposite form to override other SSL options and indicate that SSL should *not* be used. To do this,

specify the option as `--skip-ssl` [749] or `--ssl=0` [749]. For example, you might have SSL options specified in the `[client]` group of your option file to use SSL connections by default when you invoke MySQL client programs. To use an unencrypted connection instead, invoke the client program with `--skip-ssl` [749] on the command line to override the options in the option file.

Use of `--ssl` [749] does not *require* an SSL connection to be used, it only permits it. For example, if you specify this option for a client program but the server has not been configured to permit SSL connections, an unencrypted connection is used.

The secure way to require use of an SSL connection is to create a MySQL account that includes at least a `REQUIRE SSL` clause in the `GRANT` statement. In this case, connections for that account will be rejected unless MySQL supports SSL connections and the server and client have been started with the proper SSL options.

The `REQUIRE` clause permits other SSL-related restrictions as well. These can be used for stricter requirements than `REQUIRE SSL`. The description of `REQUIRE` in [Section 13.7.1.3, “GRANT Syntax”](#), provides additional detail about which SSL command options may or must be specified by clients that connect using accounts that are created using the various `REQUIRE` options.

- `--ssl-ca=file_name` [750]

The path to a file in PEM format that contains a list of trusted SSL certificate authorities. This option implies `--ssl` [749].

If you use SSL when establishing a client connection, you can tell the client not to authenticate the server certificate by specifying neither `--ssl-ca` [750] nor `--ssl-capath` [750]. The server still verifies the client according to any applicable requirements established using `GRANT` statements for the client, and it still uses any `--ssl-ca` [750]/`--ssl-capath` [750] values that were passed to server at startup.

- `--ssl-capath=directory_name` [750]

The path to a directory that contains trusted SSL certificate authority certificates in PEM format. This option implies `--ssl` [749].

If you use SSL when establishing a client connection, you can tell the client not to authenticate the server certificate by specifying neither `--ssl-ca` [750] nor `--ssl-capath` [750]. The server still verifies the client according to any applicable requirements established using `GRANT` statements for the client, and it still uses any `--ssl-ca` [750]/`--ssl-capath` [750] values that were passed to server at startup.

MySQL distributions built with OpenSSL support the `--ssl-capath` [750] option. Distributions built with yaSSL do not because yaSSL does not look in any directory and does not follow a chained certificate tree. yaSSL requires that all components of the CA certificate tree be contained within a single CA certificate tree and that each certificate in the file has a unique SubjectName value. To work around this yaSSL limitation, concatenate the individual certificate files comprising the certificate tree into a new file. Then specify the new file as the value of the `--ssl-capath` [750] option.

- `--ssl-cert=file_name` [750]

The name of the SSL certificate file in PEM format to use for establishing a secure connection. This option implies `--ssl` [749].

- `--ssl-cipher=cipher_list` [750]

A list of permissible ciphers to use for SSL encryption. If no cipher in the list is supported, SSL connections will not work. This option implies `--ssl` [749].

For greatest portability, *cipher_list* should be a list of one or more cipher names, separated by colons. This format is understood both by OpenSSL and yaSSL. Examples:

```
--ssl-cipher=AES128-SHA
--ssl-cipher=DHE-RSA-AES256-SHA:AES128-SHA
```

OpenSSL supports a more flexible syntax for specifying ciphers, as described in the OpenSSL documentation at <http://www.openssl.org/docs/apps/ciphers.html>. However, yaSSL does not, so attempts to use that extended syntax fails for a MySQL distribution built with yaSSL.

- `--ssl-key=file_name` [751]

The name of the SSL key file in PEM format to use for establishing a secure connection.

If the MySQL distribution was built using OpenSSL and the key file is protected by a passphrase, the program will prompt the user for the passphrase. The password must be given interactively; it cannot be stored in a file. If the passphrase is incorrect, the program continues as if it could not read the key. If the MySQL distribution was built using yaSSL and the key file is protected by a passphrase, an error occurs.

- `--ssl-verify-server-cert` [751]

This option is available for client programs only, not the server. It causes the client to check the server's Common Name value in the certificate that the server sends to the client. The client verifies that name against the host name the client uses for connecting to the server, and the connection fails if there is a mismatch. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default.

6.3.8.5. Setting Up SSL Certificates and Keys for MySQL

This section demonstrates how to set up SSL certificate and key files for use by MySQL servers and clients. The first example shows a simplified procedure such as you might use from the command line. The second shows a script that contains more detail. The first two examples are intended for use on Unix and both use the `openssl` command that is part of OpenSSL. The third example describes how to set up SSL files on Windows.

Example 1: Creating SSL Files from the Command Line on Unix

The following example shows a set of commands to create MySQL server and client certificate and key files. You will need to respond to several prompts by the `openssl` commands. To generate test files, you can press Enter to all prompts. To generate files for production use, you should provide nonempty responses.

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts

# Create CA certificate
shell> openssl genrsa 2048 > ca-key.pem
shell> openssl req -new -x509 -nodes -days 3600 \
    -key ca-key.pem -out ca-cert.pem

# Create server certificate, remove passphrase, and sign it
# server-cert.pem = public key, server-key.pem = private key
shell> openssl req -newkey rsa:2048 -days 3600 \
    -nodes -keyout server-key.pem -out server-req.pem
shell> openssl rsa -in server-key.pem -out server-key.pem
shell> openssl x509 -req -in server-req.pem -days 3600 \
    -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem

# Create client certificate, remove passphrase, and sign it
```

```
# client-cert.pem = public key, client-key.pem = private key
shell> openssl req -newkey rsa:2048 -days 3600 \
    -nodes -keyout client-key.pem -out client-req.pem
shell> openssl rsa -in client-key.pem -out client-key.pem
shell> openssl x509 -req -in client-req.pem -days 3600 \
    -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

After generating the certificates, verify them:

```
shell> openssl verify -CAfile ca-cert.pem server-cert.pem client-cert.pem
server-cert.pem: OK
client-cert.pem: OK
```

Now you have a set of files that can be used as follows:

- `ca-cert.pem`: Use this as the argument to `--ssl-ca` [750] on the server and client sides. (The CA certificate, if used, must be the same on both sides.)
- `server-cert.pem`, `server-key.pem`: Use these as the arguments to `--ssl-cert` [750] and `--ssl-key` [751] on the server side.
- `client-cert.pem`, `client-key.pem`: Use these as the arguments to `--ssl-cert` [750] and `--ssl-key` [751] on the client side.

To use the files to test SSL connections, see [Section 6.3.8.3, “Using SSL Connections”](#).

Example 2: Creating SSL Files Using a Script on Unix

Here is an example script that shows how to set up SSL certificate and key files for MySQL. After executing the script, use the files to test SSL connections as described in [Section 6.3.8.3, “Using SSL Connections”](#).

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/ca-cert.pem \
    -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
```

```
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..+++++
# .....+++++
# writing new private key to '/home/monty/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca -cert $DIR/ca-cert.pem -policy policy_anything \
    -out $DIR/server-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName          :PRINTABLE:'FI'
```

```

# organizationName      :PRINTABLE:'MySQL AB'
# commonName           :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
    $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#
openssl ca -cert $DIR/ca-cert.pem -policy policy_anything \
    -out $DIR/client-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName      :PRINTABLE:'FI'

```

```
# organizationName      :PRINTABLE:'MySQL AB'
# commonName           :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cat <<EOF > $DIR/my.cnf
[client]
ssl-ca=$DIR/ca-cert.pem
ssl-cert=$DIR/client-cert.pem
ssl-key=$DIR/client-key.pem
[mysqld]
ssl-ca=$DIR/ca-cert.pem
ssl-cert=$DIR/server-cert.pem
ssl-key=$DIR/server-key.pem
EOF
```

Example 3: Creating SSL Files on Windows

Download OpenSSL for Windows if it is not installed on your system. An overview of available packages can be seen here:

<http://www.sproweb.com/products/Win32OpenSSL.html>

Choose the Win32 OpenSSL Light or Win64 OpenSSL Light package, depending on your architecture (32-bit or 64-bit). The default installation location will be `C:\OpenSSL-Win32` or `C:\OpenSSL-Win64`, depending on which package you downloaded. The following instructions assume a default location of `C:\OpenSSL-Win32`. Modify this as necessary if you are using the 64-bit package.

If a message occurs during setup indicating '`...critical component is missing: Microsoft Visual C++ 2008 Redistributables`', cancel the setup and download one of the following packages as well, again depending on your architecture (32-bit or 64-bit):

- Visual C++ 2008 Redistributables (x86), available at:

<http://www.microsoft.com/downloads/details.aspx?familyid=9B2DA534-3E03-4391-8A4D-074B9F2BC1BF>

- Visual C++ 2008 Redistributables (x64), available at:

<http://www.microsoft.com/downloads/details.aspx?familyid=bd2a6171-e2d6-4230-b809-9a8d7548c1b6>

After installing the additional package, restart the OpenSSL setup procedure.

During installation, leave the default `C:\OpenSSL-Win32` as the install path, and also leave the default option '`Copy OpenSSL DLL files to the Windows system directory`' selected.

When the installation has finished, add `C:\OpenSSL-Win32\bin` to the Windows System Path variable of your server:

1. On the Windows desktop, right-click the My Computer icon, and select Properties.
2. Select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.

3. Under **System Variables**, select Path, then click the **Edit** button. The **Edit System Variable** dialogue should appear.
4. Add `';C:\OpenSSL-Win32\bin'` to the end (notice the semicolon).
5. Press OK 3 times.
6. Check that OpenSSL was correctly integrated into the Path variable by opening a new command console (**Start>Run>cmd.exe**) and verifying that OpenSSL is available:

```
Microsoft Windows [Version ...]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd \

C:\>openssl
OpenSSL> exit <<< If you see the OpenSSL prompt, installation was successful.

C:\>
```

Depending on your version of Windows, the preceding path-setting instructions might differ slightly.

After OpenSSL has been installed, use instructions similar to those from Example 1 (shown earlier in this section), with the following changes:

- Change the following Unix commands:

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts
```

On Windows, use these commands instead:

```
# Create clean environment
shell> md c:\newcerts
shell> cd c:\newcerts
```

- When a `'\'` character is shown at the end of a command line, this `'\'` character must be removed and the command lines entered all on a single line.

After generating the certificate and key files, to use them to test SSL connections, see [Section 6.3.8.3, "Using SSL Connections"](#).

6.3.9. Connecting to MySQL Remotely from Windows with SSH

This section describes how to get a secure connection to a remote MySQL server with SSH. The information was provided by David Carlson <dcarlson@mplcomm.com>.

1. Install an SSH client on your Windows machine. As a user, the best nonfree one I have found is from [SecureCRT](http://www.vandyke.com/) from <http://www.vandyke.com/>. Another option is [f-secure](http://www.f-secure.com/) from <http://www.f-secure.com/>. You can also find some free ones on [Google](http://directory.google.com/Top/Computers/Internet/Protocols/SSH/Clients/Windows/) at <http://directory.google.com/Top/Computers/Internet/Protocols/SSH/Clients/Windows/>.
2. Start your Windows SSH client. Set `Host_Name = yourmysqlserver_URL_or_IP`. Set `userid=your_userid` to log in to your server. This `userid` value might not be the same as the user name of your MySQL account.
3. Set up port forwarding. Either do a remote forward (Set `local_port: 3306, remote_host: yourmysqlservername_or_ip, remote_port: 3306`) or a local forward (Set `port: 3306, host: localhost, remote port: 3306`).

4. Save everything, otherwise you will have to redo it the next time.
5. Log in to your server with the SSH session you just created.
6. On your Windows machine, start some ODBC application (such as Access).
7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in `localhost` for the MySQL host server, not `yourmysqldservername`.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.

6.3.10. MySQL Enterprise Audit



Note

MySQL Enterprise Audit is a commercial extension. To learn more about commercial products (MySQL Enterprise Edition), see <http://www.mysql.com/products/>.

As of MySQL 5.5.28, MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin named `audit_log`. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

After you install the plugin (see [Section 6.3.10.1, “Installing the Audit Log Plugin”](#)), it writes an audit log file. By default, the file is named `audit.log` in the server data directory. To change the name of the file, set the `audit_log_file` [765] system variable at server startup.

The audit log file is written in XML format, with auditable events encoded as `<AUDIT_RECORD>` elements. For details on file format and contents, see [Section 6.3.10.2, “The Audit Log File”](#).

To control what information `audit_log` writes to its log file, set the `audit_log_policy` [766] system variable. By default, this variable is set to `ALL` (write all auditable events), but also permits values of `LOGINS` or `QUERIES` to log only login or query events, or `NONE` to disable logging.

For more information about controlling how logging occurs, see [Section 6.3.10.3, “Audit Log Plugin Logging Control”](#). For descriptions of all system variables provided by the audit log plugin, see [Section 6.3.10.4, “Audit Log Plugin Options and Variables”](#).

The Performance Schema (see [Chapter 21, MySQL Performance Schema](#)) has instrumentation for the audit log plugin. To identify the relevant instruments, use this query:

```
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE '%/alog/%';
```

6.3.10.1. Installing the Audit Log Plugin

The audit log plugin is named `audit_log`. To be usable by the server, the plugin library object file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` [537] system variable). If necessary, set the value of `plugin_dir` [537] at server startup to tell the server the location of the plugin directory.

To load the plugin at server startup, use the `--plugin-load` [457] option to name the object file that contains the plugin. For example, put the following lines in your `my.cnf` file:

```
[mysqld]
plugin-load=audit_log.so
```

If object files have a suffix different from `.so` on your system, substitute the correct suffix (for example, `.dll` on Windows).

With this plugin-loading method, the option must be given each time the server starts.

Alternatively, to register the plugin at runtime, use this statement (changing the extension as necessary):

```
mysql> INSTALL PLUGIN audit_log SONAME 'audit_log.so';
```

`INSTALL PLUGIN` loads the plugin, and also registers it in the `mysql.plugins` table to cause the plugin to be loaded for each subsequent normal server startup.

If the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load` [457], you can use the `--audit-log` option at server startup to control plugin activation. For example, to load the plugin and prevent it from being removed at runtime, use these options:

```
[mysqld]
plugin-load=audit_log.so
audit-log=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without the audit plugin, use `--audit-log` [764] with a value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

For general information about installing plugins, see [Section 5.1.8, "Server Plugins"](#). To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement. See [Section 5.1.8.2, "Obtaining Server Plugin Information"](#).

For security reasons, the audit log file produced by `audit_log` should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log. The default file is `audit.log` in the data directory. This can be changed by setting the `audit_log_file` [765] system variable at server startup.

For additional information about system variables that control operation of the `audit_log` plugin, see [Section 6.3.10.4, "Audit Log Plugin Options and Variables"](#).

6.3.10.2. The Audit Log File

The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The contents are not encrypted. The root element is `<AUDIT>`. The closing `</AUDIT>` tag of the root element is written when the plugin terminates, so the tag is not present in the file while the audit log plugin is active.

The root element contains `<AUDIT_RECORD>` elements. Each `<AUDIT_RECORD>` element has an empty body; all audit record fields are represented by element attributes.

Here is a sample log file, reformatted for readability:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUDIT>
  <AUDIT_RECORD
    TIMESTAMP="2012-08-02T14:52:12"
    NAME="Audit"
    SERVER_ID="150514"
    VERSION="1"
    STARTUP_OPTIONS="--port=3306"
```

```

OS_VERSION="i686-Linux"
MYSQL_VERSION="5.5.28-debug-log"/>
<AUDIT_RECORD
  TIMESTAMP="2012-08-02T14:52:41"
  NAME="Connect"
  CONNECTION_ID="1"
  STATUS="0"
  USER="root"
  PRIV_USER="root"
  OS_LOGIN=""
  PROXY_USER=""
  HOST="localhost"
  IP="127.0.0.1"
  DB="" />
<AUDIT_RECORD
  TIMESTAMP="2012-08-02T14:53:45"
  NAME="Query"
  CONNECTION_ID="1"
  STATUS="0"
  SQLTEXT="INSERT INTO t1 () VALUES()"/>
<AUDIT_RECORD
  TIMESTAMP="2012-08-02T14:53:51"
  NAME="Quit"
  CONNECTION_ID="1"
  STATUS="0"/>
<AUDIT_RECORD
  TIMESTAMP="2012-08-06T14:21:03"
  NAME="NoAudit"
  SERVER_ID="150514"/>
</AUDIT>

```

Attributes of `<AUDIT_RECORD>` elements have these characteristics:

- Some attributes appear in every element, but most are optional and do not necessarily appear in every element.
- Order of attributes within an element is not guaranteed.
- Attribute values are not fixed length. Long values may be truncated as indicated in the attribute descriptions given later.
- The '<', '>', '"', and '&' characters are encoded as `<`, `>`, `"`, and `&`, respectively. Null bytes (U+00) are encoded as the '?' character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

```
#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFF] | [#x10000-#x10FFFF]
```

The following attributes appear in every `<AUDIT_RECORD>` element:

- **NAME**

A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

Example: `NAME="Query"`

Some common **NAME** values:

```

"Audit"      When auditing starts, which may be server startup time
"Connect"    When a client connects, also known as logging in
"Query"      An SQL statement (executed directly)
"Prepare"    Preparation of an SQL statement; usually followed by Execute

```

```
"Execute" Execution of an SQL statement; usually follows Prepare
"Shutdown" Server shutdown
"Quit" When a client disconnects
"NoAudit" Auditing has been turned off
```

The possible values are "Audit", "Binlog Dump", "Change user", "Close stmt", "Connect Out", "Connect", "Create DB", "Daemon", "Debug", "Delayed insert", "Drop DB", "Execute", "Fetch", "Field List", "Init DB", "Kill", "Long Data", "NoAudit", "Ping", "Prepare", "Processlist", "Query", "Quit", "Refresh", "Register Slave", "Reset stmt", "Set option", "Shutdown", "Sleep", "Statistics", "Table Dump", "Time".

With the exception of "Audit" and "NoAudit", these values correspond to the `COM_xxx` command values listed in the `mysql_com.h` header file.

- **TIMESTAMP**

The date and time that the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `TIMESTAMP` value occurring after the statement finishes, not when it is received. The value is UTC, in the format `yyyy-mm-ddThh:mm:ss` (with 'T', no decimals).

Example: `TIMESTAMP="2012-08-09T12:55:16"`

The following attributes are optional in `<AUDIT_RECORD>` elements. Many of them occur only for elements with specific values of the `NAME` attribute.

- **CONNECTION_ID**

An unsigned integer representing the client connection identifier. This is the same as the `CONNECTION_ID() [1191]` function value within the connection.

Example: `CONNECTION_ID="1"`

- **DB**

A string representing the default database name. This attribute appears only if the `NAME` value is "Connect" or "Change user".

- **HOST**

A string representing the client host name. This attribute appears only if the `NAME` value is "Connect" or "Change user".

Example: `HOST="localhost"`

- **IP**

A string representing the client IP address. This attribute appears only if the `NAME` value is "Connect" or "Change user".

Example: `IP="127.0.0.1"`

- **MYSQL_VERSION**

A string representing the MySQL server version. This is the same as the value of the `VERSION() [1198]` function or `version [572]` system variable. This attribute appears only if the `NAME` value is "Audit".

Example: `MYSQL_VERSION="5.5.28-debug-log"`

- `OS_LOGIN`

A string representing the external user (empty if none). The value may differ from `USER`, for example, if the server authenticates the client using an external authentication method. This attribute appears only if the `NAME` value is `"Connect"` or `"Change user"`.

- `OS_VERSION`

A string representing the operating system on which the server was built or is running. This attribute appears only if the `NAME` value is `"Audit"`.

Example: `OS_VERSION="x86_64-Linux"`

- `PRIV_USER`

A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and may be different from the `USER` value. This attribute appears only if the `NAME` value is `"Connect"` or `"Change user"`.

- `PROXY_USER`

A string representing the proxy user. The value is empty if user proxying is not in effect. This attribute appears only if the `NAME` value is `"Connect"` or `"Change user"`.

- `SERVER_ID`

An unsigned integer representing the server ID. This is the same as the value of the `server_id` [551] system variable. This attribute appears only if the `NAME` value is `"Audit"` or `"NoAudit"`.

Example: `SERVER_ID="1"`

- `SQLTEXT`

A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. This attribute appears only if the `NAME` value is `"Query"` or `"Execute"`.

The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

Example: `SQLTEXT="DELETE FROM t1"`

- `STARTUP_OPTIONS`

A string representing the options that were given when the MySQL server was started, either on the command line or in option files. This attribute appears only if the `NAME` value is `"Audit"`.

Example: `STARTUP_OPTIONS="--log-output=FILE"`

- `STATUS`

An unsigned integer representing the command status. The value is zero for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function.

The audit log does not contain the `SQLSTATE` value or error message. To see the association between error codes, `SQLSTATE` values, and messages, see [Section C.3, "Server Error Codes and Messages"](#).

Warnings are not logged.

Example: `STATUS="1051"`

- `USER`

A string representing the user name sent by the client. This may be different from the `PRIV_USER` value. This attribute appears only if the `NAME` value is `"Connect"` or `"Change user"`.

- `VERSION`

An unsigned integer representing the version of the audit log file format. This attribute appears only if the `NAME` value is `"Audit"`.

Example: `VERSION="1"`

6.3.10.3. Audit Log Plugin Logging Control

This section describes how the `audit_log` plugin performs logging and the system variables that control how logging occurs. It assumes familiarity with the log file format, described in [Section 6.3.10.2, "The Audit Log File"](#).

When the audit log plugin opens its log file, it checks whether the XML declaration and opening `<AUDIT>` root element tag need to be written and writes them if so. When the audit log plugin terminates, it writes a closing `</AUDIT>` tag to the file.

If the log file exists at open time, the plugin checks whether the file ends with an `</AUDIT>` tag and truncates it if so before writing any `<AUDIT_RECORD>` elements. If the log file already exists but does not end with `</AUDIT>` or the `</AUDIT>` tag cannot be truncated, the plugin considers the file malformed and fails to initialize. This can occur if the server crashes or is killed with the audit log plugin running. No logging occurs until the problem is rectified. Check the error log for diagnostic information:

```
[ERROR] Plugin 'audit_log' init function returned error.
```

To deal with this problem, you must either remove or rename the malformed log file and restart the server.

The MySQL server calls the audit log plugin to write an `<AUDIT_RECORD>` element whenever an auditable event occurs, such as when it completes execution of an SQL statement received from a client. Typically the first `<AUDIT_RECORD>` element has the server description and startup options. Elements following that one represent events such as client connect and disconnect events, executed SQL statements, and so forth. Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures. Contents of files referenced by statements such as `LOAD DATA INFILE` are not logged.

To permit control over how logging occurs, the `audit_log` plugin provides several system variables, described in the following list. For more information, see [Section 6.3.10.4, "Audit Log Plugin Options and Variables"](#).

- `audit_log_file` [765]: The name of the log file. By default, the name is `audit.log` in the server data directory. For security reasons, the audit log file should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log.
- `audit_log_policy` [766]: What kinds of information the plugin writes. By default, this variable is set to `ALL` (write all auditable events), but also permits values of `LOGINS` or `QUERIES` to log only login or query events, or `NONE` to disable logging.
- `audit_log_strategy` [767]: The method used for log writes. By default, the strategy value is `ASYNCHRONOUS` and the plugin logs asynchronously to a buffer, waiting if the buffer is full. It's possible

to tell the plugin not to wait (`PERFORMANCE`) or to log synchronously, either using file system caching (`SEMISYNCHRONOUS`) or forcing output with a `sync()` call after each write request (`SYNCHRONOUS`).

Asynchronous logging strategy has these characteristics:

- Minimal impact on server performance and scalability.
- Blocking of threads that generate audit events for the shortest possible time; that is, time to allocate the buffer plus time to copy the event to the buffer.
- Output goes to the buffer. Writes from the buffer to the log file are handled by a separate thread.

A disadvantage of `PERFORMANCE` strategy is that it drops events when the buffer is full, so for a heavily loaded server it is more likely that the audit log will be missing events.

With asynchronous logging, the integrity of the log file may be compromised if a problem occurs during a write to the file or if the plugin does not shut down cleanly (for example, in the event that the server host crashes). To reduce this risk, set `audit_log_strategy` [767] use synchronous logging. Regardless of strategy, logging occurs on a best-effort basis, with no guarantee of consistency.

- `audit_log_buffer_size` [764]: The size of the buffer for asynchronous logging. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin allocates this buffer only if logging is asynchronous.
- `audit_log_rotate_on_size` [766], `audit_log_flush` [765]: These variables permit audit log file rotation and flushing. The audit log file has the potential to grow very large and consume a lot of disk space. To manage the space used, either enable automatic log rotation, or manually rename the audit file and flush the log to open a new file. The renamed file can be removed or backed up as desired.

By default, `audit_log_rotate_on_size=0` [766] and there is no log rotation. In this case, the audit log plugin closes and reopens the log file when the `audit_log_flush` [765] value changes from disabled to enabled. Log file renaming must be done externally to the server. Suppose that you want to maintain the three most recent log files, which cycle through the names `audit.log.1` through `audit.log.3`. On Unix, perform rotation manually like this:

1. From the command line, rename the current log files:

```
shell> mv audit.log.2 audit.log.3
shell> mv audit.log.1 audit.log.2
shell> mv audit.log audit.log.1
```

At this point, the plugin is still writing to the current log file, which has been renamed to `audit.log.1`.

2. Connect to the server and flush the log file so the plugin closes it and reopens a new `audit.log` file:

```
mysql> SET GLOBAL audit_log_flush = ON;
```

If `audit_log_rotate_on_size` [766] is greater than 0, setting `audit_log_flush` [765] has no effect. In this case, the audit log plugin closes and reopens its log file whenever a write to the file causes its size to exceed the `audit_log_rotate_on_size` [766] value. The original file is renamed to have a timestamp extension. For example, `audit.log` might be renamed to `audit.log.13440033615657730`. The last 7 digits are a fractional second part. The first 10 digits are a Unix timestamp value that can be interpreted using the `FROM_UNIXTIME()` [1140] function:

```
mysql> SELECT FROM_UNIXTIME(1344003361);
+-----+
```

```
| FROM_UNIXTIME(1344003361) |
+-----+
| 2012-08-03 09:16:01 |
+-----+
```

6.3.10.4. Audit Log Plugin Options and Variables

This section describes the command options and system variables that control operation of the audit log plugin. If values specified at startup time are incorrect, the plugin may fail to initialize properly and the server does not load it. In this case, the server may also produce error messages for other audit log settings because it will not recognize them.

To control the activation of the `audit_log` plugin, use this option:

- `--audit-log[=value]` [764]

Version Introduced	5.5.28	
Command-Line Format	<code>--audit-log[=value]</code>	
Option-File Format	<code>audit-log[=value]</code>	
	Permitted Values	
	Type	enumeration
	Default	ON
	Valid Values	ON OFF FORCE FORCE_PLUS_PERMANENT

This option controls how the server loads the `audit_log` plugin at startup. The value should be one of those available for plugin-loading options, as described in [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#). For example, `--audit-log=FORCE_PLUS_PERMANENT` [764] tells the server to load the plugin and prevent it from being removed while the server is running.

This option was added in MySQL 5.5.28. It is available only if the audit log plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load` [457]. See [Section 6.3.10.1, “Installing the Audit Log Plugin”](#).

If the `audit_log` plugin is installed, it exposes several system variables that permit control over logging:

```
mysql> SHOW VARIABLES LIKE 'audit_log%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| audit_log_buffer_size | 1048576 |
| audit_log_file | audit.log |
| audit_log_flush | OFF |
| audit_log_policy | ALL |
| audit_log_rotate_on_size | 0 |
| audit_log_strategy | ASYNCHRONOUS |
+-----+-----+
```

You can set any of these variables at server startup, and some of them at runtime.

- `audit_log_buffer_size` [764]

Version Introduced	5.5.28	
Variable Name	<code>audit_log_buffer_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Platform Bit Size	32
	Type	<code>numeric</code>
	Default	1048576
	Range	4096 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	<code>numeric</code>
	Default	1048576
	Range	4096 .. 18446744073709547520

When the audit log plugin writes events to the log asynchronously, it uses a buffer to store event contents prior to writing them. This variable controls the size of that buffer, in bytes. (The server adjusts the value to a multiple of 4096.) The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin allocates this buffer only if logging is asynchronous.

This variable was added in MySQL 5.5.28. It is available only if the `audit_log` plugin is enabled.

- [audit_log_file \[765\]](#)

Version Introduced	5.5.28	
Variable Name	<code>audit_log_file</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>audit.log</code>

The name of the file to which the audit log plugin writes events. The default value is `audit.log`. If the file name is a relative path, the server interprets it relative to the data directory. For security reasons, the audit log file should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log.

This variable was added in MySQL 5.5.28. It is available only if the `audit_log` plugin is enabled.

- [audit_log_flush \[765\]](#)

Version Introduced	5.5.28	
Variable Name	<code>audit_log_flush</code>	

Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	boolean
Default	OFF

When this variable is set to enabled (1 or ON), the audit log plugin closes and reopens its log file to flush it. (Note that the value remains OFF so that you need not disable it explicitly before enabling it again to perform another flush.) Enabling this variable has no effect unless [audit_log_rotate_on_size \[766\]](#) is 0.

This variable was added in MySQL 5.5.28. It is available only if the [audit_log](#) plugin is enabled.

- [audit_log_policy \[766\]](#)

Version Introduced	5.5.28
Variable Name	audit_log_policy
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	enumeration
Default	ALL
Valid Values	ALL NONE LOGINS QUERIES

The policy controlling what information the audit log plugin writes to its log file. The following table shows the permitted values.

Value	Description
ALL	Log all events
NONE	Log nothing (disable the audit stream)
LOGINS	Log only login events
QUERIES	Log only query events

This variable was added in MySQL 5.5.28. It is available only if the [audit_log](#) plugin is enabled.

- [audit_log_rotate_on_size \[766\]](#)

Version Introduced	5.5.28
Variable Name	audit_log_rotate_on_size
Variable Scope	Global
Dynamic Variable	Yes

	Permitted Values	
Type	numeric	
Default	0	

If the `audit_log_rotate_on_size` [766] value is greater than 0, the audit log plugin closes and reopens its log file if a write to the file causes its size to exceed this value. The original file is renamed to have a timestamp extension.

If the `audit_log_rotate_on_size` [766] value is 0, the plugin does not close and reopen its log based on size. Instead, use `audit_log_flush` [765] to close and reopen the log on demand. In this case, the file must be renamed externally to the server before flushing it.

For more information about audit log file rotation and timestamp interpretation, see [Section 6.3.10.3, “Audit Log Plugin Logging Control”](#).

If you set this variable to a value that is not a multiple of 4096, it is truncated to the nearest multiple.

This variable was added in MySQL 5.5.28. It is available only if the `audit_log` plugin is enabled.

- `audit_log_strategy` [767]

Version Introduced	5.5.28	
Variable Name	<code>audit_log_strategy</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
Type	enumeration	
Default	ASYNCHRONOUS	
Valid Values	ASYNCHRONOUS PERFORMANCE SEMI SYNCHRONOUS SYNCHRONOUS	

The logging method used by the audit log plugin. The following table describes the permitted values.

Table 6.16. Audit Log Strategies

Value	Meaning
ASYNCHRONOUS	Log asynchronously, wait for space in output buffer
PERFORMANCE	Log asynchronously, drop request if insufficient space in output buffer
SEMI SYNCHRONOUS	Log synchronously, permit caching by operating system
SYNCHRONOUS	Log synchronously, call <code>sync()</code> after each request

This variable was added in MySQL 5.5.28. It is available only if the `audit_log` plugin is enabled.

6.3.10.5. Audit Log Plugin Restrictions

The audit log plugin is subject to these restrictions:

- Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures.
- Contents of files referenced by statements such as `LOAD DATA INFILE` are not logged.

6.3.11. SQL-Based MySQL Account Activity Auditing

Applications can use the following guidelines to perform SQL-based auditing that ties database activity to MySQL accounts.

MySQL accounts correspond to rows in the `mysql.user` table. When a client connects successfully, the server authenticates the client to a particular row in this table. The `User` and `Host` column values in this row uniquely identify the account and correspond to the `'user_name'@'host_name'` format in which account names are written in SQL statements.

The account used to authenticate a client determines which privileges the client has. Normally, the `CURRENT_USER() [1192]` function can be invoked to determine which account this is for the client user. Its value is constructed from the `User` and `Host` columns of the `user` table row for the account.

However, there are circumstances under which the `CURRENT_USER() [1192]` value corresponds not to the client user but to a different account. This occurs in contexts when privilege checking is not based the client's account:

- Stored routines (procedures and functions) defined with the `SQL SECURITY DEFINER` characteristic
- Views defined with the `SQL SECURITY DEFINER` characteristic
- Triggers and events

In those contexts, privilege checking is done against the `DEFINER` account and `CURRENT_USER() [1192]` refers to that account, not to the account for the client who invoked the stored routine or view or who caused the trigger to activate. To determine the invoking user, you can call the `USER() [1197]` function, which returns a value indicating the actual user name provided by the client and the host from which the client connected. However, this value does not necessarily correspond directly to an account in the `user` table, because the `USER() [1197]` value never contains wildcards, whereas account values (as returned by `CURRENT_USER() [1192]`) may contain user name and host name wildcards.

For example, a blank user name matches any user, so an account of `'@'localhost` enables clients to connect as an anonymous user from the local host with any user name. If this case, if a client connects as `user1` from the local host, `USER() [1197]` and `CURRENT_USER() [1192]` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER() | CURRENT_USER() |
+-----+-----+
| user1@localhost | @localhost |
+-----+-----+
```

The host name part of an account can contain wildcards, too. If the host name contains a `'%'` or `'_'` pattern character or uses netmask notation, the account can be used for clients connecting from multiple hosts and the `CURRENT_USER() [1192]` value will not indicate which one. For example, the account `'user2'@'%.example.com'` can be used by `user2` to connect from any host in the `example.com` domain. If `user2` connects from `remote.example.com`, `USER() [1197]` and `CURRENT_USER() [1192]` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER() | CURRENT_USER() |
+-----+-----+
```

```

+-----+-----+
| user2@remote.example.com | user2@%.example.com |
+-----+-----+

```

If an application must invoke `USER()` [1197] for user auditing (for example, if it does auditing from within triggers) but must also be able to associate the `USER()` [1197] value with an account in the `user` table, it is necessary to avoid accounts that contain wildcards in the `User` or `Host` column. Specifically, do not permit `User` to be empty (which creates an anonymous-user account), and do not permit pattern characters or netmask notation in `Host` values. All accounts must have a nonempty `User` value and literal `Host` value.

With respect to the previous examples, the `'@'localhost'` and `'user2'@'%.example.com'` accounts should be changed not to use wildcards:

```

RENAME USER '@'localhost' TO 'user1'@'localhost';
RENAME USER 'user2'@'%.example.com' TO 'user2'@'remote.example.com';

```

If `user2` must be able to connect from several hosts in the `example.com` domain, there should be a separate account for each host.

To extract the user name or host name part from a `CURRENT_USER()` [1192] or `USER()` [1197] value, use the `SUBSTRING()` [1109] function:

```

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(), '@', 1);
+-----+
| SUBSTRING_INDEX(CURRENT_USER(), '@', 1) |
+-----+
| user1                                     |
+-----+

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(), '@', -1);
+-----+
| SUBSTRING_INDEX(CURRENT_USER(), '@', -1) |
+-----+
| localhost                                 |
+-----+

```

Chapter 7. Backup and Recovery

Table of Contents

7.1. Backup and Recovery Types	772
7.2. Database Backup Methods	775
7.3. Example Backup and Recovery Strategy	777
7.3.1. Establishing a Backup Policy	778
7.3.2. Using Backups for Recovery	780
7.3.3. Backup Strategy Summary	780
7.4. Using <code>mysqldump</code> for Backups	780
7.4.1. Dumping Data in SQL Format with <code>mysqldump</code>	781
7.4.2. Reloading SQL-Format Backups	782
7.4.3. Dumping Data in Delimited-Text Format with <code>mysqldump</code>	782
7.4.4. Reloading Delimited-Text Format Backups	784
7.4.5. <code>mysqldump</code> Tips	784
7.5. Point-in-Time (Incremental) Recovery Using the Binary Log	786
7.5.1. Point-in-Time Recovery Using Event Times	787
7.5.2. Point-in-Time Recovery Using Event Positions	788
7.6. <code>MyISAM</code> Table Maintenance and Crash Recovery	789
7.6.1. Using <code>myisamchk</code> for Crash Recovery	789
7.6.2. How to Check <code>MyISAM</code> Tables for Errors	790
7.6.3. How to Repair <code>MyISAM</code> Tables	791
7.6.4. <code>MyISAM</code> Table Optimization	793
7.6.5. Setting Up a <code>MyISAM</code> Table Maintenance Schedule	794

It is important to back up your databases so that you can recover your data and be up and running again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake. Backups are also essential as a safeguard before upgrading a MySQL installation, and they can be used to transfer a MySQL installation to another system or to set up replication slave servers.

MySQL offers a variety of backup strategies from which you can choose the methods that best suit the requirements for your installation. This chapter discusses several backup and recovery topics with which you should be familiar:

- Types of backups: Logical versus physical, full versus incremental, and so forth.
- Methods for creating backups.
- Recovery methods, including point-in-time recovery.
- Backup scheduling, compression, and encryption.
- Table maintenance, to enable recovery of corrupt tables.

Additional Resources

Resources related to backup or to maintaining data availability include the following:

- Customers of MySQL Enterprise Edition can use the MySQL Enterprise Backup product for backups. For an overview of the MySQL Enterprise Backup product, see [Section 24.2, “MySQL Enterprise Backup”](#).
- A forum dedicated to backup issues is available at <http://forums.mysql.com/list.php?93>.

- Details for `mysqldump`, `mysqlhotcopy`, and other MySQL backup programs can be found in [Chapter 4, MySQL Programs](#).
- The syntax of the SQL statements described here is given in [Chapter 13, SQL Statement Syntax](#).
- For additional information about `InnoDB` backup procedures, see [Section 14.3.7, “Backing Up and Recovering an InnoDB Database”](#).
- Replication enables you to maintain identical data on multiple servers. This has several benefits, such as enabling client query load to be distributed over servers, availability of data even if a given server is taken offline or fails, and the ability to make backups with no impact on the master by using a slave server. See [Chapter 16, Replication](#).
- MySQL Cluster provides a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. See [Chapter 17, MySQL Cluster NDB 7.2](#). For information specifically about MySQL Cluster backup, see [Section 17.5.3, “Online Backup of MySQL Cluster”](#).
- Distributed Replicated Block Device (DRBD) is another high-availability solution. It works by replicating a block device from a primary server to a secondary server at the block level. See [Chapter 15, High Availability and Scalability](#)

7.1. Backup and Recovery Types

This section describes the characteristics of different types of backups.

Physical (Raw) Versus Logical Backups

Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.

Logical backups save information represented as logical database structure (`CREATE DATABASE`, `CREATE TABLE` statements) and content (`INSERT` statements or delimited-text files). This type of backup is suitable for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture.

Physical backup methods have these characteristics:

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory.
- Physical backup methods are faster than logical because they involve only file copying without conversion.
- Output is more compact than for logical backup.
- Because backup speed and compactness are important for busy, important databases, the MySQL Enterprise Backup product performs physical backups. For an overview of the MySQL Enterprise Backup product, see [Section 24.2, “MySQL Enterprise Backup”](#).
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity, depending on storage engine. For example, `InnoDB` tables can each be in a separate file, or share file storage with other `InnoDB` tables; each `MyISAM` table corresponds uniquely to a set of files.
- In addition to databases, the backup can include any related files such as log or configuration files.

- Data from `MEMORY` tables is tricky to back up this way because their contents are not stored on disk. (The MySQL Enterprise Backup product has a feature where you can retrieve data from `MEMORY` tables during a backup.)
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup. MySQL Enterprise Backup does this locking automatically for tables that require it.
- Physical backup tools include the `mysqlbackup` of MySQL Enterprise Backup for `InnoDB` or any other tables, file system-level commands (such as `cp`, `scp`, `tar`, `rsync`), or `mysqlhotcopy` for `MyISAM` tables.
- For restore:
 - MySQL Enterprise Backup restores `InnoDB` and other tables that it backed up.
 - `ndb_restore` restores `NDB` tables.
 - Files copied at the file system level or with `mysqlhotcopy` can be copied back to their original locations with file system commands.

Logical backup methods have these characteristics:

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format. If the output is written on the client side, the server must also send it to the backup program.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running. The server is not taken offline.
- Logical backup tools include the `mysqldump` program and the `SELECT ... INTO OUTFILE` statement. These work for any storage engine, even `MEMORY`.
- To restore logical backups, SQL-format dump files can be processed using the `mysql` client. To load delimited-text files, use the `LOAD DATA INFILE` statement or the `mysqlimport` client.

Online Versus Offline Backups

Online backups take place while the MySQL server is running so that the database information can be obtained from the server. Offline backups take place while the server is stopped. This distinction can also be described as “hot” versus “cold” backups; a “warm” backup is one where the server remains running but locked against modifying data while you access database files externally.

Online backup methods have these characteristics:

- The backup is less intrusive to other clients, which can connect to the MySQL server during the backup and may be able to access data depending on what operations they need to perform.
- Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity. The MySQL Enterprise Backup product does such locking automatically.

Offline backup methods have these characteristics:

- Clients can be affected adversely because the server is unavailable during backup. For that reason, such backups are often taken from a replication slave server that can be taken offline without harming availability.
- The backup procedure is simpler because there is no possibility of interference from client activity.

A similar distinction between online and offline applies for recovery operations, and similar characteristics apply. However, it is more likely that clients will be affected for online recovery than for online backup because recovery requires stronger locking. During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

Local Versus Remote Backups

A local backup is performed on the same host where the MySQL server runs, whereas a remote backup is done from a different host. For some types of backups, the backup can be initiated from a remote host even if the output is written locally on the server host.

- `mysqldump` can connect to local or remote servers. For SQL output (`CREATE` and `INSERT` statements), local or remote dumps can be done and generate output on the client. For delimited-text output (with the `--tab [321]` option), data files are created on the server host.
- `mysqlhotcopy` performs only local backups: It connects to the server to lock it against data modifications and then copies local table files.
- `SELECT ... INTO OUTFILE` can be initiated from a local or remote client host, but the output file is created on the server host.
- Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for copied files might be remote.

Snapshot Backups

Some file system implementations enable “snapshots” to be taken. These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system. (For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied.) MySQL itself does not provide the capability for taking file system snapshots. It is available through third-party solutions such as Veritas, LVM, or ZFS.

Full Versus Incremental Backups

A full backup includes all data managed by a MySQL server at a given point in time. An incremental backup consists of the changes made to the data during a given time span (from one point in time to another). MySQL has different ways to perform full backups, such as those described earlier in this section. Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

Full Versus Point-in-Time (Incremental) Recovery

A full recovery restores all data from a full backup. This restores the server instance to the state that it had when the backup was made. If that state is not sufficiently current, a full recovery can be followed by recovery of incremental backups made since the full backup, to bring the server to a more up-to-date state.

Incremental recovery is recovery of changes made during a given time span. This is also called point-in-time recovery because it makes a server's state current up to a given time. Point-in-time recovery is based on the binary log and typically follows a full recovery from the backup files that restores the server to its state when the backup was made. Then the data changes written in the binary log files are applied as incremental recovery to redo data modifications and bring the server up to the desired point in time.

Table Maintenance

Data integrity can be compromised if tables become corrupt. For [InnoDB](#) tables, this is not a typical issue. For programs to check [MyISAM](#) tables and repair them if problems are found, see [Section 7.6, “MyISAM Table Maintenance and Crash Recovery”](#).

Backup Scheduling, Compression, and Encryption

Backup scheduling is valuable for automating backup procedures. Compression of backup output reduces space requirements, and encryption of the output provides better security against unauthorized access of backed-up data. MySQL itself does not provide these capabilities. The MySQL Enterprise Backup product can compress [InnoDB](#) backups, and compression or encryption of backup output can be achieved using file system utilities. Other third-party solutions may be available.

7.2. Database Backup Methods

This section summarizes some general methods for making backups.

Making a Hot Backup with MySQL Enterprise Backup

Customers of MySQL Enterprise Edition can use the [MySQL Enterprise Backup](#) product to do [physical](#) backups of entire instances or selected databases, tables, or both. This product includes features for [incremental](#) and [compressed](#) backups. Backing up the physical database files makes restore much faster than logical techniques such as the `mysqldump` command. [InnoDB](#) tables are copied using a [hot backup](#) mechanism. (Ideally, the [InnoDB](#) tables should represent a substantial majority of the data.) Tables from other storage engines are copied using a [warm backup](#) mechanism. For an overview of the MySQL Enterprise Backup product, see [Section 24.2, “MySQL Enterprise Backup”](#).

Making Backups with `mysqldump` or `mysqlhotcopy`

The `mysqldump` program and the `mysqlhotcopy` script can make backups. `mysqldump` is more general because it can back up all kinds of tables. `mysqlhotcopy` works only with some storage engines. (See [Section 7.4, “Using `mysqldump` for Backups”](#), and [Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”](#).)

For [InnoDB](#) tables, it is possible to perform an online backup that takes no locks on tables using the `--single-transaction` [321] option to `mysqldump`. See [Section 7.3.1, “Establishing a Backup Policy”](#).

Making Backups by Copying Table Files

For storage engines that represent each table using its own files, tables can be backed up by copying those files. For example, [MyISAM](#) tables are stored as files, so it is easy to do a backup by copying files

(* .frm, * .MYD, and * .MYI files). To get a consistent backup, stop the server or lock and flush the relevant tables:

```
FLUSH TABLES tbl_list WITH READ LOCK;
```

You need only a read lock; this enables other clients to continue to query the tables while you are making a copy of the files in the database directory. The flush is needed to ensure that the all active index pages are written to disk before you start the backup. See [Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”](#), and [Section 13.7.6.3, “FLUSH Syntax”](#).

You can also create a binary backup simply by copying all table files, as long as the server isn't updating anything. The `mysqlhotcopy` script uses this method. (But note that table file copying methods do not work if your database contains InnoDB tables. `mysqlhotcopy` does not work for InnoDB tables because InnoDB does not necessarily store table contents in database directories. Also, even if the server is not actively updating data, InnoDB may still have modified data cached in memory and not flushed to disk.)

Making Delimited-Text File Backups

To create a text file containing a table's data, you can use `SELECT * INTO OUTFILE 'file_name' FROM tbl_name`. The file is created on the MySQL server host, not the client host. For this statement, the output file cannot already exist because permitting files to be overwritten constitutes a security risk. See [Section 13.2.9, “SELECT Syntax”](#). This method works for any kind of data file, but saves only table data, not the table structure.

Another way to create text data files (along with files containing `CREATE TABLE` statements for the backed up tables) is to use `mysqldump` with the `--tab [321]` option. See [Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”](#).

To reload a delimited-text data file, use `LOAD DATA INFILE` or `mysqlimport`.

Making Incremental Backups by Enabling the Binary Log

MySQL supports incremental backups: You must start the server with the `--log-bin [1892]` option to enable binary logging; see [Section 5.2.4, “The Binary Log”](#). The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup. At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using `FLUSH LOGS`. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained in [Section 7.5, “Point-in-Time \(Incremental\) Recovery Using the Binary Log”](#). The next time you do a full backup, you should also rotate the binary log using `FLUSH LOGS`, `mysqldump --flush-logs`, or `mysqlhotcopy --flushlog`. See [Section 4.5.4, “mysqldump — A Database Backup Program”](#), and [Section 4.6.9, “mysqlhotcopy — A Database Backup Program”](#).

Making Backups Using Replication Slaves

If you have performance problems with your master server while making backups, one strategy that can help is to set up replication and perform backups on the slave rather than on the master. See [Section 16.3.1, “Using Replication for Backups”](#).

If you are backing up a slave replication server, you should back up its `master.info` and `relay-log.info` files when you back up the slave's databases, regardless of the backup method you choose. These information files are always needed to resume replication after you restore the slave's data. If your slave is replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication

of any interrupted `LOAD DATA INFILE` operations. The location of this directory is the value of the `--slave-load-tmpdir` [1882] option. If the server was not started with that option, the directory location is the value of the `tmpdir` [568] system variable.

Recovering Corrupt Tables

If you have to restore `MyISAM` tables that have become corrupt, try to recover them using `REPAIR TABLE` or `myisamchk -r` first. That should work in 99.9% of all cases. If `myisamchk` fails, see [Section 7.6, “MyISAM Table Maintenance and Crash Recovery”](#).

Making Backups Using a File System Snapshot

If you are using a Veritas file system, you can make a backup like this:

1. From a client program, execute `FLUSH TABLES WITH READ LOCK`.
2. From another shell, execute `mount vxfs snapshot`.
3. From the first client, execute `UNLOCK TABLES`.
4. Copy files from the snapshot.
5. Unmount the snapshot.

Similar snapshot capabilities may be available in other file systems, such as LVM or ZFS.

7.3. Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that enables you to recover data after several types of crashes:

- Operating system crash
- Power failure
- File system crash
- Hardware problem (hard drive, motherboard, and so forth)

The example commands do not include options such as `--user` [240] and `--password` [239] for the `mysqldump` and `mysql` client programs. You should include such options as necessary to enable client programs to connect to the MySQL server.

Assume that data is stored in the `InnoDB` storage engine, which has support for transactions and automatic crash recovery. Assume also that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The `InnoDB` data files might not contain consistent data due to the crash, but `InnoDB` reads its logs and finds in them the list of pending committed and noncommitted transactions that have not been flushed to the data files. `InnoDB` automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
```

```
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

For the cases of file system crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that backups must already have been made. To make sure that is the case, design and implement a backup policy.

7.3.1. Establishing a Backup Policy

To be useful, backups must be scheduled regularly. A full backup (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, [MySQL Enterprise Backup](#) can perform a [physical backup](#) of an entire instance, with optimizations to minimize overhead and avoid disruption when backing up [InnoDB](#) data files; [mysqldump](#) provides online [logical backup](#). This discussion uses [mysqldump](#).

Assume that we make a full backup of all our [InnoDB](#) tables in all databases using the following command on Sunday at 1 p.m., when load is low:

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

The resulting `.sql` file produced by [mysqldump](#) contains a set of SQL `INSERT` statements that can be used to reload the dumped tables at a later time.

This backup operation acquires a global read lock on all tables at the beginning of the dump (using `FLUSH TABLES WITH READ LOCK`). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the `FLUSH` statement is issued, the backup operation may stall until those statements finish. After that, the dump becomes lock-free and does not disturb reads and writes on the tables.

It was assumed earlier that the tables to back up are [InnoDB](#) tables, so `--single-transaction` [\[321\]](#) uses a consistent read and guarantees that data seen by [mysqldump](#) does not change. (Changes made by other clients to [InnoDB](#) tables are not seen by the [mysqldump](#) process.) If the backup operation includes nontransactional tables, consistency requires that they do not change during the backup. For example, for the [MyISAM](#) tables in the `mysql` database, there must be no administrative changes to MySQL accounts during the backup.

Full backups are necessary, but it is not always convenient to create them. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. It is more efficient to make an initial full backup, and then to make incremental backups. The incremental backups are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. In MySQL, these changes are represented in the binary log, so the MySQL server should always be started with the `--log-bin` [1892] option to enable that log. With binary logging enabled, the server writes each data change into a file while it updates data. Looking at the data directory of a MySQL server that was started with the `--log-bin` [1892] option and that has been running for some days, we find these MySQL binary log files:

```
-rw-rw---- 1 guilhem guilhem 1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem guilhem      4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem guilhem      79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem guilhem     508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem guilhem   998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem guilhem     361 Nov 14 10:07 gbichot2-bin.index
```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a `mysqladmin flush-logs` command. `mysqldump` also has an option to flush the logs. The `.index` file in the data directory contains the list of all MySQL binary logs in the directory.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, the binary log files created afterward contain all the data changes made since the backup. Let's modify the previous `mysqldump` command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
        --all-databases > backup_sunday_1_PM.sql
```

After executing this command, the data directory contains a new binary log file, `gbichot2-bin.000007`, because the `--flush-logs` [316] option causes the server to flush its logs. The `--master-data` [317] option causes `mysqldump` to write binary log information to its output, so the resulting `.sql` dump file includes these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

Because the `mysqldump` command made a full backup, those lines mean two things:

- The dump file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or newer.
- All data changes logged after the backup are not present in the dump file, but are present in the `gbichot2-bin.000007` binary log file or newer.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a `mysqladmin flush-logs` command creates `gbichot2-bin.000008`. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. will be in the `gbichot2-bin.000007` file. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another `mysqladmin flush-logs` command. All changes between Monday 1 p.m. and Tuesday 1 p.m. will be in the `gbichot2-bin.000008` file (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
        --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

**Note**

Deleting the MySQL binary logs with `mysqldump --delete-master-logs` can be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log. The description for the `PURGE BINARY LOGS` statement explains what should be verified before deleting the MySQL binary logs. See [Section 13.4.1.1, “PURGE BINARY LOGS Syntax”](#).

7.3.2. Using Backups for Recovery

Now, suppose that we have a catastrophic crash on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the `gbichot2-bin.000007` and `gbichot2-bin.000008` binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a `--log-bin [1892]` option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the `gbichot2-bin.000009` file (and any subsequent files) at hand, and we could apply them using `mysqlbinlog` and `mysql` to restore the most recent data changes with no loss up to the moment of the crash:

```
shell> mysqlbinlog gbichot2-bin.000009 ... | mysql
```

For more information about using `mysqlbinlog` to process binary log files, see [Section 7.5, “Point-in-Time \(Incremental\) Recovery Using the Binary Log”](#).

7.3.3. Backup Strategy Summary

In case of an operating system crash or power failure, `InnoDB` itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the `--log-bin [1892]` option, or even `--log-bin=log_name [1892]`, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).
- Make periodic full backups, using the `mysqldump` command shown earlier in [Section 7.3.1, “Establishing a Backup Policy”](#), that makes an online, nonblocking backup.
- Make periodic incremental backups by flushing the logs with `FLUSH LOGS` or `mysqladmin flush-logs`.

7.4. Using `mysqldump` for Backups

This section describes how to use `mysqldump` to produce dump files, and how to reload dump files. A dump file can be used in several ways:

- As a backup to enable data recovery in case of data loss.
- As a source of data for setting up replication slaves.
- As a source of data for experimentation:
 - To make a copy of a database that you can use without changing the original data.
 - To test potential upgrade incompatibilities.

`mysqldump` produces two types of output, depending on whether the `--tab` [321] option is given:

- Without `--tab` [321], `mysqldump` writes SQL statements to the standard output. This output consists of `CREATE` statements to create dumped objects (databases, tables, stored routines, and so forth), and `INSERT` statements to load data into tables. The output can be saved in a file and reloaded later using `mysql` to recreate the dumped objects. Options are available to modify the format of the SQL statements, and to control which objects are dumped.
- With `--tab` [321], `mysqldump` produces two output files for each dumped table. The server writes one file as tab-delimited text, one line per table row. This file is named `tbl_name.txt` in the output directory. The server also sends a `CREATE TABLE` statement for the table to `mysqldump`, which writes it as a file named `tbl_name.sql` in the output directory.

7.4.1. Dumping Data in SQL Format with `mysqldump`

This section describes how to use `mysqldump` to create SQL-format dump files. For information about reloading such dump files, see [Section 7.4.2, “Reloading SQL-Format Backups”](#).

By default, `mysqldump` writes information as SQL statements to the standard output. You can save the output in a file:

```
shell> mysqldump [arguments] > file_name
```

To dump all databases, invoke `mysqldump` with the `--all-databases` [313] option:

```
shell> mysqldump --all-databases > dump.sql
```

To dump only specific databases, name them on the command line and use the `--databases` [314] option:

```
shell> mysqldump --databases db1 db2 db3 > dump.sql
```

The `--databases` [314] option causes all names on the command line to be treated as database names. Without this option, `mysqldump` treats the first name as a database name and those following as table names.

With `--all-databases` [313] or `--databases` [314], `mysqldump` writes `CREATE DATABASE` and `USE` statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the `--add-drop-database` [313] option as well. In this case, `mysqldump` writes a `DROP DATABASE` statement preceding each `CREATE DATABASE` statement.

To dump a single database, name it on the command line:

```
shell> mysqldump --databases test > dump.sql
```

In the single-database case, it is permissible to omit the `--databases` [314] option:

```
shell> mysqldump test > dump.sql
```

The difference between the two preceding commands is that without `--databases` [314], the dump output contains no `CREATE DATABASE` or `USE` statements. This has several implications:

- When you reload the dump file, you must specify a default database name so that the server knows which database to reload.
- For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.
- If the database to be reloaded does not exist, you must create it first.
- Because the output will contain no `CREATE DATABASE` statement, the `--add-drop-database` [313] option has no effect. If you use it, it produces no `DROP DATABASE` statement.

To dump only specific tables from a database, name them on the command line following the database name:

```
shell> mysqldump test t1 t3 t7 > dump.sql
```

7.4.2. Reloading SQL-Format Backups

To reload a dump file written by `mysqldump` that consists of SQL statements, use it as input to the `mysql` client. If the dump file was created by `mysqldump` with the `--all-databases` [313] or `--databases` [314] option, it contains `CREATE DATABASE` and `USE` statements and it is not necessary to specify a default database into which to load the data:

```
shell> mysql < dump.sql
```

Alternatively, from within `mysql`, use a `source` command:

```
mysql> source dump.sql
```

If the file is a single-database dump not containing `CREATE DATABASE` and `USE` statements, create the database first (if necessary):

```
shell> mysqladmin create db1
```

Then specify the database name when you load the dump file:

```
shell> mysql db1 < dump.sql
```

Alternatively, from within `mysql`, create the database, select it as the default database, and load the dump file:

```
mysql> CREATE DATABASE IF NOT EXISTS db1;
mysql> USE db1;
mysql> source dump.sql
```

7.4.3. Dumping Data in Delimited-Text Format with `mysqldump`

This section describes how to use `mysqldump` to create delimited-text dump files. For information about reloading such dump files, see [Section 7.4.4, “Reloading Delimited-Text Format Backups”](#).

If you invoke `mysqldump` with the `--tab=dir_name` [321] option, it uses `dir_name` as the output directory and dumps tables individually in that directory using two files for each table. The table name is the

basename for these files. For a table named `t1`, the files are named `t1.sql` and `t1.txt`. The `.sql` file contains a `CREATE TABLE` statement for the table. The `.txt` file contains the table data, one line per table row.

The following command dumps the contents of the `db1` database to files in the `/tmp` database:

```
shell> mysqldump --tab=/tmp db1
```

The `.txt` files containing table data are written by the server, so they are owned by the system account used for running the server. The server uses `SELECT ... INTO OUTFILE` to write the files, so you must have the `FILE [696]` privilege to perform this operation, and an error occurs if a given `.txt` file already exists.

The server sends the `CREATE` definitions for dumped tables to `mysqldump`, which writes them to `.sql` files. These files therefore are owned by the user who executes `mysqldump`.

It is best that `--tab [321]` be used only for dumping a local server. If you use it with a remote server, the `--tab [321]` directory must exist on both the local and remote hosts, and the `.txt` files will be written by the server in the remote directory (on the server host), whereas the `.sql` files will be written by `mysqldump` in the local directory (on the client host).

For `mysqldump --tab`, the server by default writes table data to `.txt` files one line per row with tabs between column values, no quotation marks around column values, and newline as the line terminator. (These are the same defaults as for `SELECT ... INTO OUTFILE`.)

To enable data files to be written using a different format, `mysqldump` supports these options:

- `--fields-terminated-by=str [316]`
The string for separating column values (default: tab).
- `--fields-enclosed-by=char [316]`
The character within which to enclose column values (default: no character).
- `--fields-optionally-enclosed-by=char [316]`
The character within which to enclose non-numeric column values (default: no character).
- `--fields-escaped-by=char [316]`
The character for escaping special characters (default: no escaping).
- `--lines-terminated-by=str [317]`
The line-termination string (default: newline).

Depending on the value you specify for any of these options, it might be necessary on the command line to quote or escape the value appropriately for your command interpreter. Alternatively, specify the value using hex notation. Suppose that you want `mysqldump` to quote column values within double quotation marks. To do so, specify double quote as the value for the `--fields-enclosed-by [316]` option. But this character is often special to command interpreters and must be treated specially. For example, on Unix, you can quote the double quote like this:

```
--fields-enclosed-by='\"'
```

On any platform, you can specify the value in hex:

```
--fields-enclosed-by=0x22
```

It is common to use several of the data-formatting options together. For example, to dump tables in comma-separated values format with lines terminated by carriage-return/newline pairs (`\r\n`), use this command (enter it on a single line):

```
shell> mysqldump --tab=/tmp --fields-terminated-by=,
--fields-enclosed-by=''' --lines-terminated-by=0x0d0a db1
```

Should you use any of the data-formatting options to dump table data, you will need to specify the same format when you reload data files later, to ensure proper interpretation of the file contents.

7.4.4. Reloading Delimited-Text Format Backups

For backups produced with `mysqldump --tab`, each table is represented in the output directory by an `.sql` file containing the `CREATE TABLE` statement for the table, and a `.txt` file containing the table data. To reload a table, first change location into the output directory. Then process the `.sql` file with `mysql` to create an empty table and process the `.txt` file to load the data into the table:

```
shell> mysql db1 < t1.sql
shell> mysqlimport db1 t1.txt
```

An alternative to using `mysqlimport` to load the data file is to use the `LOAD DATA INFILE` statement from within the `mysql` client:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1;
```

If you used any data-formatting options with `mysqldump` when you initially dumped the table, you must use the same options with `mysqlimport` or `LOAD DATA INFILE` to ensure proper interpretation of the data file contents:

```
shell> mysqlimport --fields-terminated-by=,
--fields-enclosed-by=''' --lines-terminated-by=0x0d0a db1 t1.txt
```

Or:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1
-> FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY ''''
-> LINES TERMINATED BY '\r\n';
```

7.4.5. `mysqldump` Tips

This section surveys techniques that enable you to use `mysqldump` to solve specific problems:

- How to make a copy a database
- How to copy a database from one server to another
- How to dump stored programs (stored procedures and functions, triggers, and events)
- How to dump definitions and data separately

7.4.5.1. Making a Copy of a Database

```
shell> mysqldump db1 > dump.sql
shell> mysqladmin create db2
shell> mysql db2 < dump.sql
```

Do not use `--databases` [314] on the `mysqldump` command line because that causes `USE db1` to be included in the dump file, which overrides the effect of naming `db2` on the `mysql` command line.

7.4.5.2. Copy a Database from one Server to Another

On Server 1:

```
shell> mysqldump --databases db1 > dump.sql
```

Copy the dump file from Server 1 to Server 2.

On Server 2:

```
shell> mysql < dump.sql
```

Use of `--databases` [314] with the `mysqldump` command line causes the dump file to include `CREATE DATABASE` and `USE` statements that create the database if it does exist and make it the default database for the reloaded data.

Alternatively, you can omit `--databases` [314] from the `mysqldump` command. Then you will need to create the database on Server 2 (if necessary) and specify it as the default database when you reload the dump file.

On Server 1:

```
shell> mysqldump db1 > dump.sql
```

On Server 2:

```
shell> mysqladmin create db1
shell> mysql db1 < dump.sql
```

You can specify a different database name in this case, so omitting `--databases` [314] from the `mysqldump` command enables you to dump data from one database and load it into another.

7.4.5.3. Dumping Stored Programs

Several options control how `mysqldump` handles stored programs (stored procedures and functions, triggers, and events):

- `--events` [316]: Dump Event Scheduler events
- `--routines` [320]: Dump stored procedures and functions
- `--triggers` [322]: Dump triggers for tables

The `--triggers` [322] option is enabled by default so that when tables are dumped, they are accompanied by any triggers they have. The other options are disabled by default and must be specified explicitly to dump the corresponding objects. To disable any of these options explicitly, use its skip form: `--skip-events` [316], `--skip-routines` [320], or `--skip-triggers` [322].

7.4.5.4. Dumping Table Definitions and Content Separately

The `--no-data` [319] option tells `mysqldump` not to dump table data, resulting in the dump file containing only statements to create the tables. Conversely, the `--no-create-info` [318] option tells `mysqldump` to suppress `CREATE` statements from the output, so that the dump file contains only table data.

For example, to dump table definitions and data separately for the `test` database, use these commands:

```
shell> mysqldump --no-data test > dump-defs.sql
shell> mysqldump --no-create-info test > dump-data.sql
```

For a definition-only dump, add the `--routines` [320] and `--events` [316] options to also include stored routine and event definitions:

```
shell> mysqldump --no-data --routines --events test > dump-defs.sql
```

7.4.5.5. Using `mysqldump` to Test for Upgrade Incompatibilities

When contemplating a MySQL upgrade, it is prudent to install the newer version separately from your current production version. Then you can dump the database and database object definitions from the production server and load them into the new server to verify that they are handled properly. (This is also useful for testing downgrades.)

On the production server:

```
shell> mysqldump --all-databases --no-data --routines --events > dump-defs.sql
```

On the upgraded server:

```
shell> mysql < dump-defs.sql
```

Because the dump file does not contain table data, it can be processed quickly. This enables you to spot potential incompatibilities without waiting for lengthy data-loading operations. Look for warnings or errors while the dump file is being processed.

After you have verified that the definitions are handled properly, dump the data and try to load it into the upgraded server.

On the production server:

```
shell> mysqldump --all-databases --no-create-info > dump-data.sql
```

On the upgraded server:

```
shell> mysql < dump-data.sql
```

Now check the table contents and run some test queries.

7.5. Point-in-Time (Incremental) Recovery Using the Binary Log

Point-in-time recovery refers to recovery of data changes made since a given point in time. Typically, this type of recovery is performed after restoring a full backup that brings the server to its state as of the time the backup was made. (The full backup can be made in several ways, such as those listed in [Section 7.2, “Database Backup Methods”](#).) Point-in-time recovery then brings the server up to date incrementally from the time of the full backup to a more recent time.

Point-in-time recovery is based on these principles:

- The source of information for point-in-time recovery is the set of incremental backups represented by the binary log files generated subsequent to the full backup operation. Therefore, the server must be started with the `--log-bin` [1892] option to enable binary logging (see [Section 5.2.4, “The Binary Log”](#)).

To restore data from the binary log, you must know the name and location of the current binary log files. By default, the server creates binary log files in the data directory, but a path name can be specified with the `--log-bin` [1892] option to place the files in a different location. [Section 5.2.4, “The Binary Log”](#).

To see a listing of all binary log files, use this statement:

```
mysql> SHOW BINARY LOGS;
```

To determine the name of the current binary log file, issue the following statement:

```
mysql> SHOW MASTER STATUS;
```

- The `mysqlbinlog` utility converts the events in the binary log files from binary format to text so that they can be executed or viewed. `mysqlbinlog` has options for selecting sections of the binary log based on event times or position of events within the log. See [Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”](#).
- Executing events from the binary log causes the data modifications they represent to be redone. This enables recovery of data changes for a given span of time. To execute events from the binary log, process `mysqlbinlog` output using the `mysql` client:

```
shell> mysqlbinlog binlog_files | mysql -u root -p
```

- Viewing log contents can be useful when you need to determine event times or positions to select partial log contents prior to executing events. To view events from the log, send `mysqlbinlog` output into a paging program:

```
shell> mysqlbinlog binlog_files | more
```

Alternatively, save the output in a file and view the file in a text editor:

```
shell> mysqlbinlog binlog_files > tmpfile
shell> ... edit tmpfile ...
```

- Saving the output in a file is useful as a preliminary to executing the log contents with certain events removed, such as an accidental `DROP DATABASE`. You can delete from the file any statements not to be executed before executing its contents. After editing the file, execute the contents as follows:

```
shell> mysql -u root -p < tmpfile
```

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first `mysql` process terminates, the server drops the temporary table. When the second `mysql` process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 > /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

7.5.1. Point-in-Time Recovery Using Event Times

To indicate the start and end times for recovery, specify the `--start-datetime` [377] and `--stop-datetime` [377] options for `mysqlbinlog`, in `DATETIME` format. As an example, suppose that exactly

at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

```
shell> mysqlbinlog --stop-datetime="2005-04-20 9:59:59" \
/var/log/mysql/bin.123456 | mysql -u root -p
```

This command recovers all of the data up until the date and time given by the `--stop-datetime` [377] option. If you did not detect the erroneous SQL statement that was entered until hours later, you will probably also want to recover the activity that occurred afterward. Based on this, you could run `mysqlbinlog` again with a start date and time, like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 10:01:00" \
/var/log/mysql/bin.123456 | mysql -u root -p
```

In this command, the SQL statements logged from 10:01 a.m. on will be re-executed. The combination of restoring of the previous night's dump file and the two `mysqlbinlog` commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on.

To use this method of point-in-time recovery, you should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

Then open the `/tmp/mysql_restore.sql` file with a text editor to examine it.

Excluding specific changes by specifying times for `mysqlbinlog` does not work well if multiple statements executed at the same time as the one to be excluded.

7.5.2. Point-in-Time Recovery Using Event Positions

Instead of specifying dates and times, the `--start-position` [377] and `--stop-position` [377] options for `mysqlbinlog` can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions occurred around the same time as a damaging SQL statement. To determine the position numbers, run `mysqlbinlog` for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 9:55:00" \
--stop-datetime="2005-04-20 10:05:00" \
/var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

This command creates a small text file in the `/tmp` directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you do not want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as `log_pos` followed by a number. After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

```
shell> mysqlbinlog --stop-position=368312 /var/log/mysql/bin.123456 \
| mysql -u root -p

shell> mysqlbinlog --start-position=368315 /var/log/mysql/bin.123456 \
| mysql -u root -p
```

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of `mysqlbinlog` includes `SET TIMESTAMP` statements before each SQL statement recorded,

the recovered data and related MySQL logs will reflect the original times at which the transactions were executed.

7.6. MyISAM Table Maintenance and Crash Recovery

This section discusses how to use `myisamchk` to check or repair MyISAM tables (tables that have `.MYD` and `.MYI` files for storing data and indexes). For general `myisamchk` background, see [Section 4.6.3](#), “`myisamchk` — MyISAM Table-Maintenance Utility”. Other table-repair information can be found at [Section 2.12.4](#), “Rebuilding or Repairing Tables or Indexes”.

You can use `myisamchk` to check, repair, or optimize database tables. The following sections describe how to perform these operations and how to set up a table maintenance schedule. For information about using `myisamchk` to get information about your tables, see [Section 4.6.3.5](#), “Obtaining Table Information with `myisamchk`”.

Even though table repair with `myisamchk` is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table.

`myisamchk` operations that affect indexes can cause `FULLTEXT` indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in [Section 4.6.3.1](#), “`myisamchk` General Options”.

MyISAM table maintenance can also be done using the SQL statements that perform operations similar to what `myisamchk` can do:

- To check MyISAM tables, use `CHECK TABLE`.
- To repair MyISAM tables, use `REPAIR TABLE`.
- To optimize MyISAM tables, use `OPTIMIZE TABLE`.
- To analyze MyISAM tables, use `ANALYZE TABLE`.

For additional information about these statements, see [Section 13.7.2](#), “Table Maintenance Statements”.

These statements can be used directly or by means of the `mysqlcheck` client program. One advantage of these statements over `myisamchk` is that the server does all the work. With `myisamchk`, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between `myisamchk` and the server.

7.6.1. Using `myisamchk` for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why. See [Section C.5.4.2](#), “What to Do If MySQL Keeps Crashing”.

For an explanation of how MyISAM tables can become corrupted, see [Section 14.5.4](#), “MyISAM Table Problems”.

If you run `mysqld` with external locking disabled (which is the default), you cannot reliably use `myisamchk` to check a table when `mysqld` is using the same table. If you can be certain that no one will access the tables through `mysqld` while you run `myisamchk`, you only have to execute `mysqladmin flush-tables` before you start checking the tables. If you cannot guarantee this, you must stop `mysqld` while you check the tables. If you run `myisamchk` to check tables that `mysqld` is updating at the same time, you may get a warning that a table is corrupt even when it is not.

If the server is run with external locking enabled, you can use [myisamchk](#) to check tables at any time. In this case, if the server tries to update a table that [myisamchk](#) is using, the server will wait for [myisamchk](#) to finish before it continues.

If you use [myisamchk](#) to repair or optimize tables, you *must* always ensure that the [mysqld](#) server is not using the table (this also applies if external locking is disabled). If you do not stop [mysqld](#), you should at least do a [mysqladmin flush-tables](#) before you run [myisamchk](#). Your tables *may become corrupted* if the server and [myisamchk](#) access the tables simultaneously.

When performing crash recovery, it is important to understand that each [MyISAM](#) table *tbl_name* in a database corresponds to the three files in the database directory shown in the following table.

File	Purpose
<i>tbl_name.frm</i>	Definition (format) file
<i>tbl_name.MYD</i>	Data file
<i>tbl_name.MYI</i>	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

[myisamchk](#) works by creating a copy of the [.MYD](#) data file row by row. It ends the repair stage by removing the old [.MYD](#) file and renaming the new file to the original file name. If you use [--quick](#) [351], [myisamchk](#) does not create a temporary [.MYD](#) file, but instead assumes that the [.MYD](#) file is correct and generates only a new index file without touching the [.MYD](#) file. This is safe, because [myisamchk](#) automatically detects whether the [.MYD](#) file is corrupt and aborts the repair if it is. You can also specify the [--quick](#) [351] option twice to [myisamchk](#). In this case, [myisamchk](#) does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the [.MYD](#) file. Normally the use of two [--quick](#) [351] options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running [myisamchk](#).

7.6.2. How to Check [MyISAM](#) Tables for Errors

To check a [MyISAM](#) table, use the following commands:

- `myisamchk tbl_name`

This finds 99.99% of all errors. What it cannot find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run [myisamchk](#) without options or with the `-s` (silent) option.

- `myisamchk -m tbl_name`

This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- `myisamchk -e tbl_name`

This does a complete and thorough check of all data (`-e` means “extended check”). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, [myisamchk](#) stops after the first error it finds. If you want to obtain more information, you can add the `-v` (verbose) option. This causes [myisamchk](#) to keep going, up through a maximum of 20 errors.

- `myisamchk -e -i tbl_name`

This is like the previous command, but the `-i` option tells `myisamchk` to print additional statistical information.

In most cases, a simple `myisamchk` command with no arguments other than the table name is sufficient to check a table.

7.6.3. How to Repair `MyISAM` Tables

The discussion in this section describes how to use `myisamchk` on `MyISAM` tables (extensions `.MYI` and `.MYD`).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See [Section 13.7.2.2, “CHECK TABLE Syntax”](#), and [Section 13.7.2.5, “REPAIR TABLE Syntax”](#).

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

- `tbl_name.frm` is locked against change
- Can't find file `tbl_name.MYI` (Errcode: `nnn`)
- Unexpected end of file
- Record file is crashed
- Got error `nnn` from table handler

To get more information about the error, run `pererror nnn`, where `nnn` is the error number. The following example shows how to use `pererror` to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> pererror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use `ALTER TABLE` to increase the `MAX_ROWS` and `AVG_ROW_LENGTH` table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you do not know the current table option values, use `SHOW CREATE TABLE`.

For the other errors, you must repair your tables. `myisamchk` can usually detect and fix most problems that occur.

The repair process involves up to four stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that `mysqld` runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This section is for the cases where a table check fails (such as those described in [Section 7.6.2, “How to Check `MyISAM` Tables for Errors”](#)), or you want to use the extended features that `myisamchk` provides.

The `myisamchk` options used for table maintenance with are described in [Section 4.6.3, “`myisamchk` — `MyISAM` Table-Maintenance Utility”](#). `myisamchk` also has variables that you can set to control memory allocation that may improve performance. See [Section 4.6.3.6, “`myisamchk` Memory Usage”](#).

If you are going to repair a table from the command line, you must first stop the `mysqld` server. Note that when you do `mysqladmin shutdown` on a remote server, the `mysqld` server is still available for a while after `mysqladmin` returns, until all statement-processing has stopped and all index changes have been flushed to disk.

Stage 1: Checking your tables

Run `myisamchk *.MYI` or `myisamchk -e *.MYI` if you have more time. Use the `-s` (silent) option to suppress unnecessary information.

If the `mysqld` server is stopped, you should use the `--update-state [350]` option to tell `myisamchk` to mark the table as “checked.”

You have to repair only those tables for which `myisamchk` announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try `myisamchk -r -q tbl_name` (`-r -q` means “quick recovery mode”). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use `myisamchk -r tbl_name` (`-r` means “recovery mode”). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.
3. If the preceding step fails, use `myisamchk --safe-recover tbl_name`. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).



Note

If you want a repair operation to go much faster, you should set the values of the `sort_buffer_size [554]` and `key_buffer_size [508]` variables each to about 25% of your available memory when running `myisamchk`.

If you get unexpected errors when repairing (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.

- Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
mysql> SET autocommit=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

- Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)



Important

If you are using replication, you should stop it prior to performing the above procedure, since it involves file system operations, and these are not logged by MySQL.

Go back to Stage 2. `myisamchk -r -q` should work. (This should not be an endless loop.)

You can also use the `REPAIR TABLE tbl_name USE_FRM` SQL statement, which performs the whole procedure automatically. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `REPAIR TABLE`. See [Section 13.7.2.5, “REPAIR TABLE Syntax”](#).

Stage 4: Very difficult repair

You should reach this stage only if the `.frm` description file has also crashed. That should never happen, because the description file is not changed after the table is created:

- Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with `myisamchk -r`.
- If you do not have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, and then move the `.frm` description and `.MYI` index files from the other database to your crashed database. This gives you new description and index files, but leaves the `.MYD` data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

7.6.4. MyISAM Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run `myisamchk` in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the `OPTIMIZE TABLE` SQL statement. `OPTIMIZE TABLE` does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See [Section 13.7.2.4, “OPTIMIZE TABLE Syntax”](#).

`myisamchk` has a number of other options that you can use to improve the performance of a table:

- `--analyze [352]` or `-a`: Perform key distribution analysis. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use.
- `--sort-index [352]` or `-S`: Sort the index blocks. This optimizes seeks and makes table scans that use indexes faster.
- `--sort-records=index_num [352]` or `-R index_num`: Sort data rows according to a given index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index.

For a full description of all available options, see [Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

7.6.5. Setting Up a `MyISAM` Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair `MyISAM` tables is with the `CHECK TABLE` and `REPAIR TABLE` statements. See [Section 13.7.2, “Table Maintenance Statements”](#).

Another way to check tables is to use `myisamchk`. For maintenance purposes, you can use `myisamchk -s`. The `-s` option (short for `--silent` [347]) causes `myisamchk` to run in silent mode, printing messages only when errors occur.

It is also a good idea to enable automatic `MyISAM` table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are “expected crashed tables.”) To cause the server to check `MyISAM` tables automatically, start it with the `--myisam-recover-options` [454] option. See [Section 5.1.3, “Server Command Options”](#).

You should also check your tables regularly during normal system operation. For example, you can run a `cron` job to check important tables once a week, using a line like this in a `crontab` file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

This prints out information about crashed tables so that you can examine and repair them as necessary.

To start with, execute `myisamchk -s` each night on all tables that have been updated during the last 24 hours. As you see that problems occur infrequently, you can back off the checking frequency to once a week or so.

Normally, MySQL tables need little maintenance. If you are performing many updates to `MyISAM` tables with dynamic-sized rows (tables with `VARCHAR`, `BLOB`, or `TEXT` columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using `OPTIMIZE TABLE` on the tables in question. Alternatively, if you can stop the `mysqld` server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index --myisam_sort_buffer_size=16M /*/*.MYI
```

Chapter 8. Optimization

Table of Contents

8.1. Optimization Overview	796
8.2. Optimizing SQL Statements	798
8.2.1. Optimizing <code>SELECT</code> Statements	798
8.2.2. Optimizing DML Statements	802
8.2.3. Optimizing Database Privileges	804
8.2.4. Optimizing <code>INFORMATION_SCHEMA</code> Queries	804
8.2.5. Other Optimization Tips	809
8.3. Optimization and Indexes	810
8.3.1. How MySQL Uses Indexes	810
8.3.2. Using Primary Keys	811
8.3.3. Using Foreign Keys	811
8.3.4. Column Indexes	811
8.3.5. Multiple-Column Indexes	812
8.3.6. Verifying Index Usage	814
8.3.7. <code>InnoDB</code> and <code>MyISAM</code> Index Statistics Collection	814
8.3.8. Comparison of B-Tree and Hash Indexes	815
8.4. Optimizing Database Structure	817
8.4.1. Optimizing Data Size	817
8.4.2. Optimizing MySQL Data Types	819
8.4.3. Optimizing for Many Tables	820
8.5. Optimizing for <code>InnoDB</code> Tables	823
8.5.1. Optimizing Storage Layout for <code>InnoDB</code> Tables	823
8.5.2. Optimizing <code>InnoDB</code> Transaction Management	823
8.5.3. Optimizing <code>InnoDB</code> Logging	824
8.5.4. Bulk Data Loading for <code>InnoDB</code> Tables	824
8.5.5. Optimizing <code>InnoDB</code> Queries	825
8.5.6. Optimizing <code>InnoDB</code> DDL Operations	826
8.5.7. Optimizing <code>InnoDB</code> Disk I/O	826
8.5.8. Optimizing <code>InnoDB</code> Configuration Variables	827
8.5.9. Optimizing <code>InnoDB</code> for Systems with Many Tables	828
8.6. Optimizing for <code>MyISAM</code> Tables	828
8.6.1. Optimizing <code>MyISAM</code> Queries	828
8.6.2. Bulk Data Loading for <code>MyISAM</code> Tables	830
8.6.3. Speed of <code>REPAIR TABLE</code> Statements	831
8.7. Optimizing for <code>MEMORY</code> Tables	833
8.8. Understanding the Query Execution Plan	833
8.8.1. Optimizing Queries with <code>EXPLAIN</code>	833
8.8.2. <code>EXPLAIN</code> Output Format	834
8.8.3. <code>EXPLAIN EXTENDED</code> Output Format	844
8.8.4. Estimating Query Performance	846
8.8.5. Controlling the Query Optimizer	846
8.9. Buffering and Caching	848
8.9.1. The <code>InnoDB</code> Buffer Pool	848
8.9.2. The <code>MyISAM</code> Key Cache	851
8.9.3. The MySQL Query Cache	855
8.10. Optimizing Locking Operations	862
8.10.1. Internal Locking Methods	862
8.10.2. Table Locking Issues	864

8.10.3. Concurrent Inserts	866
8.10.4. Metadata Locking Within Transactions	866
8.10.5. External Locking	867
8.11. Optimizing the MySQL Server	868
8.11.1. System Factors and Startup Parameter Tuning	868
8.11.2. Tuning Server Parameters	868
8.11.3. Optimizing Disk I/O	875
8.11.4. Optimizing Memory Use	879
8.11.5. Optimizing Network Use	882
8.11.6. The Thread Pool Plugin	884
8.12. Measuring Performance (Benchmarking)	890
8.12.1. Measuring the Speed of Expressions and Functions	891
8.12.2. The MySQL Benchmark Suite	891
8.12.3. Using Your Own Benchmarks	892
8.12.4. Measuring Performance with <code>performance_schema</code>	892
8.12.5. Examining Thread Information	892
8.13. Internal Details of MySQL Optimizations	907
8.13.1. Range Optimization	907
8.13.2. Index Merge Optimization	910
8.13.3. Engine Condition Pushdown Optimization	913
8.13.4. <code>IS NULL</code> Optimization	915
8.13.5. <code>LEFT JOIN</code> and <code>RIGHT JOIN</code> Optimization	916
8.13.6. Nested-Loop Join Algorithms	917
8.13.7. Nested Join Optimization	918
8.13.8. Outer Join Simplification	924
8.13.9. <code>ORDER BY</code> Optimization	926
8.13.10. <code>GROUP BY</code> Optimization	929
8.13.11. <code>DISTINCT</code> Optimization	931
8.13.12. Optimizing Subqueries with <code>EXISTS</code> Strategy	932

This chapter explains how to optimize MySQL performance and provides examples. Optimization involves configuring, tuning, and measuring performance, at several levels. Depending on your job role (developer, DBA, or a combination of both), you might optimize at the level of individual SQL statements, entire applications, a single database server, or multiple networked database servers. Sometimes you can be proactive and plan in advance for performance, while other times you might troubleshoot a configuration or code issue after a problem occurs. Optimizing CPU and memory usage can also improve scalability, allowing the database to handle more load without slowing down.

8.1. Optimization Overview

Database performance depends on several factors at the database level, such as tables, queries, and configuration settings. These software constructs result in CPU and I/O operations at the hardware level, which you must minimize and make as efficient as possible. As you work on database performance, you start by learning the high-level rules and guidelines for the software side, and measuring performance using wall-clock time. As you become an expert, you learn more about what happens internally, and start measuring things such as CPU cycles and I/O operations.

Typical users aim to get the best database performance out of their existing software and hardware configurations. Advanced users look for opportunities to improve the MySQL software itself, or develop their own storage engines and hardware appliances to expand the MySQL ecosystem.

Optimizing at the Database Level

The most important factor in making a database application fast is its basic design:

- Are the tables structured properly? In particular, do the columns have the right data types, and does each table have the appropriate columns for the type of work? For example, applications that perform frequent updates often have many tables with few columns, while applications that analyze large amounts of data often have few tables with many columns.
- Are the right [indexes](#) in place to make queries efficient?
- Are you using the appropriate storage engine for each table, and taking advantage of the strengths and features of each storage engine you use? In particular, the choice of a transactional storage engine such as [InnoDB](#) or a non-transactional one such as [MyISAM](#) can be very important for performance and scalability.



Note

In MySQL 5.5 and higher, [InnoDB](#) is the default storage engine for new tables. In practice, the advanced [InnoDB](#) performance features mean that [InnoDB](#) tables often outperform the simpler [MyISAM](#) tables, especially for a busy database.

- Does each table use an appropriate row format? This choice also depends on the storage engine used for the table. In particular, compressed tables use less disk space and so require less disk I/O to read and write the data. Compression is available for all kinds of workloads with [InnoDB](#) tables, and for read-only [MyISAM](#) tables.
- Does the application use an appropriate [locking strategy](#)? For example, by allowing shared access when possible so that database operations can run concurrently, and requesting exclusive access when appropriate so that critical operations get top priority. Again, the choice of storage engine is significant. The [InnoDB](#) storage engine handles most locking issues without involvement from you, allowing for better concurrency in the database and reducing the amount of experimentation and tuning for your code.
- Are all [memory areas used for caching](#) sized correctly? That is, large enough to hold frequently accessed data, but not so large that they overload physical memory and cause paging. The main memory areas to configure are the [InnoDB](#) buffer pool, the [MyISAM](#) key cache, and the MySQL query cache.

Optimizing at the Hardware Level

Any database application eventually hits hardware limits as the database becomes more and more busy. A DBA must evaluate whether it is possible to tune the application or reconfigure the server to avoid these [bottlenecks](#), or whether more hardware resources are required. System bottlenecks typically arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read or write the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When the data is in main memory, we must process it to get our result. Having small tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.

- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

Balancing Portability and Performance

To use performance-oriented SQL extensions in a portable MySQL program, you can wrap MySQL-specific keywords in a statement within `/*! */` comment delimiters. Other SQL servers ignore the commented keywords. For information about writing comments, see [Section 9.6, “Comment Syntax”](#).

8.2. Optimizing SQL Statements

The core logic of a database application is performed through SQL statements, whether issued directly through an interpreter or submitted behind the scenes through an API. The tuning guidelines in this section help to speed up all kinds of MySQL applications. The guidelines cover SQL operations that read and write data, the behind-the-scenes overhead for SQL operations in general, and operations used in specific scenarios such as database monitoring.

8.2.1. Optimizing `SELECT` Statements

Queries, in the form of `SELECT` statements, perform all the lookup operations in the database. Tuning these statements is a top priority, whether to achieve sub-second response times for dynamic web pages, or to chop hours off the time to generate huge overnight reports.

Besides `SELECT` statements, the tuning techniques for queries also apply to constructs such as `CREATE TABLE...AS SELECT`, `INSERT INTO...SELECT`, and `WHERE` clauses in `DELETE` statements. Those statements have additional performance considerations because they combine write operations with the read-oriented query operations.

The main considerations for optimizing queries are:

- Set up indexes on columns used in the `WHERE` clause, to speed up evaluation, filtering, and the final retrieval of results. To avoid wasted disk space, construct a small set of indexes that speed up many related queries used in your application.
- Minimize the number of table scans in your queries, particularly for big tables.
- Keep table statistics up to date, so the optimizer has the information needed to construct an efficient execution plan.
- Learn the tuning techniques, indexing techniques, and configuration parameters that are specific to the storage engine for each table. Both `InnoDB` and `MyISAM` have sets of guidelines for enabling and sustaining high performance in queries.
- Isolate and tune any part of the query, such as a function call, that takes excessive time. Remember that such a function might get called millions of times for a big query or insert operation.
- Avoid transforming the query in ways that make it hard to understand, especially if the optimizer does some of the same transformations automatically.
- If a performance issue is not easily solved by one of the basic guidelines, investigate the internal details of the specific query by reading the `EXPLAIN` plan and adjusting your indexes, `WHERE` clauses, join clauses, and so on. (When you reach a certain level of expertise, reading the `EXPLAIN` plan might be your first step for every query.)

- Adjust the size and properties of the memory areas that MySQL uses for caching. With efficient use of the `InnoDB` buffer pool, `MyISAM` key cache, and the MySQL query cache, repeated queries run faster because the results are retrieved from memory the second and subsequent times.
- Even for a query that runs fast using the cache memory areas, you might still optimize further so that they require less cache memory, making your application more scalable. Scalability means that your application can handle more simultaneous users, larger requests, and so on without experiencing a big drop in performance.
- Deal with locking issues, where the speed of your query might be affected by other sessions accessing the tables at the same time.

8.2.1.1. Speed of `SELECT` Statements

In general, when you want to make a slow `SELECT ... WHERE` query faster, the first thing to check is whether you can add an index. Indexes are especially important for queries that reference different tables, using features such as joins and foreign keys. You can use the `EXPLAIN` statement to determine which indexes are used for a `SELECT`. See [Section 8.8.1, “Optimizing Queries with `EXPLAIN`”](#), and [Section 8.3.1, “How MySQL Uses Indexes”](#).

8.2.1.2. How MySQL Optimizes `WHERE` Clauses

This section discusses optimizations that can be made for processing `WHERE` clauses. The examples use `SELECT` statements, but the same optimizations apply for `WHERE` clauses in `DELETE` and `UPDATE` statements.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
  WHERE key_part1=constant;

SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... LIMIT 10;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL resolves the following queries using only the entries from a secondary index, if the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
  WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use the index data to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... ;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

You might be tempted to rewrite your queries to make arithmetic operations faster, while sacrificing readability. Because MySQL does similar optimizations automatically, you can often avoid this work, and leave the query in a more understandable and maintainable form. Some of the optimizations performed by MySQL follow:



Note

Because work on the MySQL optimizer is ongoing, not all of the optimizations that MySQL performs are documented here.

- Removal of unnecessary parentheses:

```
((a AND b) AND c OR ((a AND b) AND (c AND d)))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
(a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal (needed because of constant folding):

```
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- Constant expressions used by indexes are evaluated only once.
- `COUNT(*)` [1204] on a single table without a `WHERE` is retrieved directly from the table information for `MyISAM` and `MEMORY` tables. This is also done for any `NOT NULL` expression when used with only one table.
- Early detection of invalid constant expressions. MySQL quickly detects that some `SELECT` statements are impossible and returns no rows.
- `HAVING` is merged with `WHERE` if you do not use `GROUP BY` or aggregate functions (`COUNT()` [1204], `MIN()` [1206], and so on).
- For each table in a join, a simpler `WHERE` is constructed to get a fast `WHERE` evaluation for the table and also to skip rows as soon as possible.
- All constant tables are read first before any other tables in the query. A constant table is any of the following:
 - An empty table or a table with one row.
 - A table that is used with a `WHERE` clause on a `PRIMARY KEY` or a `UNIQUE` index, where all index parts are compared to constant expressions and are defined as `NOT NULL`.

All of the following tables are used as constant tables:

```
SELECT * FROM t WHERE primary_key=1;
SELECT * FROM t1,t2
WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in `ORDER BY` and `GROUP BY` clauses come from the same table, that table is preferred first when joining.
- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use the `SQL_SMALL_RESULT` option, MySQL uses an in-memory temporary table.

- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table, but a fixed percentage no longer determines the choice between using an index or a scan. The optimizer now is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size.
- MySQL can sometimes produce query results using data from the index, without consulting the table data. If all columns used from the index are numeric, only the index data is used to resolve the query.
- Before each row is output, those that do not match the `HAVING` clause are skipped.

8.2.1.3. Optimizing `LIMIT` Queries

If you need only a specified number of rows from a result set, use a `LIMIT` clause in the query, rather than fetching the whole result set and throwing away the extra data.

MySQL sometimes optimizes a query that has a `LIMIT row_count` clause and no `HAVING` clause:

- If you select only a few rows with `LIMIT`, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.
- If you use `LIMIT row_count` with `ORDER BY`, MySQL ends the sorting as soon as it has found the first `row_count` rows of the sorted result, rather than sorting the entire result. If ordering is done by using an index, this is very fast. If a filesort must be done, all rows that match the query without the `LIMIT` clause are selected, and most or all of them are sorted, before the first `row_count` are found. After the initial rows have been found, MySQL does not sort any remainder of the result set.
- When combining `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds `row_count` unique rows.
- In some cases, a `GROUP BY` can be resolved by reading the key in order (or doing a sort on the key) and then calculating summaries until the key value changes. In this case, `LIMIT row_count` does not calculate any unnecessary `GROUP BY` values.
- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using `SQL_CALC_FOUND_ROWS`.
- `LIMIT 0` quickly returns an empty set. This can be useful for checking the validity of a query. When using one of the MySQL APIs, it can also be employed for obtaining the types of the result columns. (This trick does not work in the MySQL Monitor (the `mysql` program), which merely displays `Empty set` in such cases; instead, use `SHOW COLUMNS` or `DESCRIBE` for this purpose.)
- When the server uses temporary tables to resolve the query, it uses the `LIMIT row_count` clause to calculate how much space is required.

8.2.1.4. How to Avoid Full Table Scans

Frequently, a `full table scan` is a danger sign that a query can be speeded up significantly. For tables with more than a few rows, consider redesigning the query by adding an index for one or more of the columns tested in the `WHERE` clause. Put extra effort into avoiding table scans for queries that perform joins or reference foreign keys. If the nature of the data means there is no way to avoid reading all the rows, then it might not be practical to make the query faster, or making it faster might involve extensive restructuring of your tables that is beyond the scope of this section.

The output from `EXPLAIN` shows `ALL` in the `type` column when MySQL uses a table scan to resolve a query. This usually happens under the following conditions:

- The `ON` or `WHERE` clauses do not reference any indexed columns that the query can use. Consider adding an index, or refining those clauses to refer to an indexed column.
- The table is so small that it is faster to perform a table scan than to bother with a key lookup. This is common for tables with fewer than 10 rows and a short row length. Don't worry in this case.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See [Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”](#). For example, to query census data only for males or only for females, MySQL must read most of the data blocks in the table, so locating the rows through the index would add unnecessary overhead. Don't worry if you encounter this condition for occasional big reports. If these reports are frequent or truly time-critical, and the table is huge, you might partition, shard, or create dimension tables using the relevant column.
- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it probably will do many key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate and the performance impact is negligible. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Minimize the `OR` keywords in your `WHERE` clauses. If there is no index that helps to locate the values on *both* sides of the `OR`, any row could potentially be part of the result set, so all rows must be tested, and that requires a full table scan. If you have one index that helps to optimize one side of an `OR` query, and a different index that helps to optimize the other side, use a `UNION` operator to run separate fast queries and merge the results afterward.
- With tables that use the `MEMORY` storage engine, if you run queries that examine ranges of values (using operators such as `>`, `<=`, or `BETWEEN` on the indexed columns), create the index with the `USING BTREE` clause. The default (`USING HASH`) is fast for retrieving individual rows with an equality operator (`=` or `<=>`), but is much slower (requiring a full table scan) to examine a range of column values. A `MEMORY` table created with the `USING BTREE` clause is still fast for equality comparisons, so use that clause for your `MEMORY` tables that handle a variety of queries.
- Use `ANALYZE TABLE tbl_name` to update the key distributions for the scanned table. See [Section 13.7.2.1, “ANALYZE TABLE Syntax”](#).
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

See [Section 13.2.9.3, “Index Hint Syntax”](#).

- Start `mysqld` with the `--max-seeks-for-key=1000` [\[524\]](#) option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See [Section 5.1.4, “Server System Variables”](#).

8.2.2. Optimizing DML Statements

This section explains how to speed up the data manipulation language (DML) statements, `INSERT`, `UPDATE`, and `DELETE`. Traditional OLTP applications and modern web applications typically do many small DML operations, where concurrency is vital. Data analysis and reporting applications typically run DML operations that affect many rows at once, where the main considerations is the I/O to write large amounts of data and keep indexes up-to-date. For inserting and updating large volumes of data (known in the industry as ETL, for “extract-transform-load”), sometimes you use other SQL statements or external commands, that mimic the effects of `INSERT`, `UPDATE`, and `DELETE` statements.

8.2.2.1. Speed of **INSERT** Statements

To optimize insert speed, combine many small operations into a single large operation. Ideally, you make a single connection, send the data for many new rows at once, and delay all index updates and consistency checking until the very end.

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting row: (1 × size of row)
- Inserting indexes: (1 × number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by $\log N$, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use **INSERT** statements with multiple **VALUES** lists to insert several rows at a time. This is considerably faster (many times faster in some cases) than using separate single-row **INSERT** statements. If you are adding data to a nonempty table, you can tune the `bulk_insert_buffer_size` [486] variable to make data insertion even faster. See [Section 5.1.4, “Server System Variables”](#).
- When loading a table from a text file, use **LOAD DATA INFILE**. This is usually 20 times faster than using **INSERT** statements. See [Section 13.2.6, “LOAD DATA INFILE Syntax”](#).
- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.
- See [Section 8.5.4, “Bulk Data Loading for InnoDB Tables”](#) for tips specific to **InnoDB** tables.
- See [Section 8.6.2, “Bulk Data Loading for MyISAM Tables”](#) for tips specific to **MyISAM** tables.

8.2.2.2. Speed of **UPDATE** Statements

An update statement is optimized like a **SELECT** query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed do not get updated.

Another way to get fast updates is to delay updates and then do many updates in a row later. Performing multiple updates together is much quicker than doing one at a time if you lock the table.

For a **MyISAM** table that uses dynamic row format, updating a row to a longer total length may split the row. If you do this often, it is very important to use **OPTIMIZE TABLE** occasionally. See [Section 13.7.2.4, “OPTIMIZE TABLE Syntax”](#).

8.2.2.3. Speed of **DELETE** Statements

The time required to delete individual rows in a [MyISAM](#) table is exactly proportional to the number of indexes. To delete rows more quickly, you can increase the size of the key cache by increasing the `key_buffer_size` [508] system variable. See [Section 8.11.2](#), “[Tuning Server Parameters](#)”.

To delete all rows from a [MyISAM](#) table, `TRUNCATE TABLE tbl_name` is faster than `DELETE FROM tbl_name`. Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock. See [Section 13.1.33](#), “[TRUNCATE TABLE Syntax](#)”.

8.2.3. Optimizing Database Privileges

The more complex your privilege setup, the more overhead applies to all SQL statements. Simplifying the privileges established by `GRANT` statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you do not grant any table-level or column-level privileges, the server need not ever check the contents of the `tables_priv` and `columns_priv` tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high statement-processing load, consider using a simplified grant structure to reduce permission-checking overhead.

8.2.4. Optimizing [INFORMATION_SCHEMA](#) Queries

Applications that monitor the database can make frequent use of the [INFORMATION_SCHEMA](#) tables. Certain types of queries for [INFORMATION_SCHEMA](#) tables can be optimized to execute more quickly. The goal is to minimize file operations (for example, scanning a directory or opening a table file) to collect the information that makes up these dynamic tables. These optimizations do have an effect on how collations are used for searches in [INFORMATION_SCHEMA](#) tables. For more information, see [Section 10.1.7.9](#), “[Collation and INFORMATION_SCHEMA Searches](#)”.

1) Try to use constant lookup values for database and table names in the `WHERE` clause

You can take advantage of this principle as follows:

- To look up databases or tables, use expressions that evaluate to a constant, such as literal values, functions that return a constant, or scalar subqueries.
- Avoid queries that use a nonconstant database name lookup value (or no lookup value) because they require a scan of the data directory to find matching database directory names.
- Within a database, avoid queries that use a nonconstant table name lookup value (or no lookup value) because they require a scan of the database directory to find matching table files.

This principle applies to the [INFORMATION_SCHEMA](#) tables shown in the following table, which shows the columns for which a constant lookup value enables the server to avoid a directory scan. For example, if you are selecting from `TABLES`, using a constant lookup value for `TABLE_SCHEMA` in the `WHERE` clause enables a data directory scan to be avoided.

Table	Column to specify to avoid data directory scan	Column to specify to avoid database directory scan
<code>COLUMNS</code>	<code>TABLE_SCHEMA</code>	<code>TABLE_NAME</code>
<code>KEY_COLUMN_USAGE</code>	<code>TABLE_SCHEMA</code>	<code>TABLE_NAME</code>
<code>PARTITIONS</code>	<code>TABLE_SCHEMA</code>	<code>TABLE_NAME</code>
<code>REFERENTIAL_CONSTRAINTS</code>	<code>CONSTRAINT_SCHEMA</code>	<code>TABLE_NAME</code>
<code>STATISTICS</code>	<code>TABLE_SCHEMA</code>	<code>TABLE_NAME</code>
<code>TABLES</code>	<code>TABLE_SCHEMA</code>	<code>TABLE_NAME</code>

Table	Column to specify to avoid data directory scan	Column to specify to avoid database directory scan
TABLE_CONSTRAINTS	TABLE_SCHEMA	TABLE_NAME
TRIGGERS	EVENT_OBJECT_SCHEMA	EVENT_OBJECT_TABLE
VIEWS	TABLE_SCHEMA	TABLE_NAME

The benefit of a query that is limited to a specific constant database name is that checks need be made only for the named database directory. Example:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test';
```

Use of the literal database name `test` enables the server to check only the `test` database directory, regardless of how many databases there might be. By contrast, the following query is less efficient because it requires a scan of the data directory to determine which database names match the pattern `'test%'`:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA LIKE 'test%';
```

For a query that is limited to a specific constant table name, checks need be made only for the named table within the corresponding database directory. Example:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 't1';
```

Use of the literal table name `t1` enables the server to check only the files for the `t1` table, regardless of how many tables there might be in the `test` database. By contrast, the following query requires a scan of the `test` database directory to determine which table names match the pattern `'t%'`:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME LIKE 't%';
```

The following query requires a scan of the database directory to determine matching database names for the pattern `'test%'`, and for each matching database, it requires a scan of the database directory to determine matching table names for the pattern `'t%'`:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test%' AND TABLE_NAME LIKE 't%';
```

2) Write queries that minimize the number of table files that must be opened

For queries that refer to certain `INFORMATION_SCHEMA` table columns, several optimizations are available that minimize the number of table files that must be opened. Example:

```
SELECT TABLE_NAME, ENGINE FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test';
```

In this case, after the server has scanned the database directory to determine the names of the tables in the database, those names become available with no further file system lookups. Thus, `TABLE_NAME` requires no files to be opened. The `ENGINE` (storage engine) value can be determined by opening the table's `.frm` file, without touching other table files such as the `.MYD` or `.MYI` file.

Some values, such as `INDEX_LENGTH` for `MyISAM` tables, require opening the `.MYD` or `.MYI` file as well.

The file-opening optimization types are denoted thus:

- `SKIP_OPEN_TABLE`: Table files do not need to be opened. The information has already become available within the query by scanning the database directory.

- **OPEN_FRM_ONLY**: Only the table's `.frm` file need be opened.
- **OPEN_TRIGGER_ONLY**: Only the table's `.TRG` file need be opened.
- **OPEN_FULL_TABLE**: The unoptimized information lookup. The `.frm`, `.MYD`, and `.MYI` files must be opened.

The following list indicates how the preceding optimization types apply to `INFORMATION_SCHEMA` table columns. For tables and columns not named, none of the optimizations apply.

- **COLUMNS**: `OPEN_FRM_ONLY` applies to all columns
- **KEY_COLUMN_USAGE**: `OPEN_FULL_TABLE` applies to all columns
- **PARTITIONS**: `OPEN_FULL_TABLE` applies to all columns
- **REFERENTIAL_CONSTRAINTS**: `OPEN_FULL_TABLE` applies to all columns
- **STATISTICS**:

Column	Optimization type
TABLE_CATALOG	OPEN_FRM_ONLY
TABLE_SCHEMA	OPEN_FRM_ONLY
TABLE_NAME	OPEN_FRM_ONLY
NON_UNIQUE	OPEN_FRM_ONLY
INDEX_SCHEMA	OPEN_FRM_ONLY
INDEX_NAME	OPEN_FRM_ONLY
SEQ_IN_INDEX	OPEN_FRM_ONLY
COLUMN_NAME	OPEN_FRM_ONLY
COLLATION	OPEN_FRM_ONLY
CARDINALITY	OPEN_FULL_TABLE
SUB_PART	OPEN_FRM_ONLY
PACKED	OPEN_FRM_ONLY
NULLABLE	OPEN_FRM_ONLY
INDEX_TYPE	OPEN_FULL_TABLE
COMMENT	OPEN_FRM_ONLY

- **TABLES**:

Column	Optimization type
TABLE_CATALOG	SKIP_OPEN_TABLE
TABLE_SCHEMA	SKIP_OPEN_TABLE
TABLE_NAME	SKIP_OPEN_TABLE
TABLE_TYPE	OPEN_FRM_ONLY
ENGINE	OPEN_FRM_ONLY
VERSION	OPEN_FRM_ONLY
ROW_FORMAT	OPEN_FULL_TABLE

Column	Optimization type
TABLE_ROWS	OPEN_FULL_TABLE
AVG_ROW_LENGTH	OPEN_FULL_TABLE
DATA_LENGTH	OPEN_FULL_TABLE
MAX_DATA_LENGTH	OPEN_FULL_TABLE
INDEX_LENGTH	OPEN_FULL_TABLE
DATA_FREE	OPEN_FULL_TABLE
AUTO_INCREMENT	OPEN_FULL_TABLE
CREATE_TIME	OPEN_FULL_TABLE
UPDATE_TIME	OPEN_FULL_TABLE
CHECK_TIME	OPEN_FULL_TABLE
TABLE_COLLATION	OPEN_FRM_ONLY
CHECKSUM	OPEN_FULL_TABLE
CREATE_OPTIONS	OPEN_FRM_ONLY
TABLE_COMMENT	OPEN_FRM_ONLY

- TABLE_CONSTRAINTS: OPEN_FULL_TABLE applies to all columns
- TRIGGERS: OPEN_TRIGGER_ONLY applies to all columns
- VIEWS:

Column	Optimization type
TABLE_CATALOG	OPEN_FRM_ONLY
TABLE_SCHEMA	OPEN_FRM_ONLY
TABLE_NAME	OPEN_FRM_ONLY
VIEW_DEFINITION	OPEN_FRM_ONLY
CHECK_OPTION	OPEN_FRM_ONLY
IS_UPDATABLE	OPEN_FULL_TABLE
DEFINER	OPEN_FRM_ONLY
SECURITY_TYPE	OPEN_FRM_ONLY
CHARACTER_SET_CLIENT	OPEN_FRM_ONLY
COLLATION_CONNECTION	OPEN_FRM_ONLY

3) Use EXPLAIN to determine whether the server can use INFORMATION_SCHEMA optimizations for a query

This applies particularly for INFORMATION_SCHEMA queries that search for information from more than one database, which might take a long time and impact performance. The Extra value in EXPLAIN output indicates which, if any, of the optimizations described earlier the server can use to evaluate INFORMATION_SCHEMA queries. The following examples demonstrate the kinds of information you can expect to see in the Extra value.

```
mysql> EXPLAIN SELECT TABLE_NAME FROM INFORMATION_SCHEMA.VIEWS WHERE
-> TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v1'\G
***** 1. row *****
```

```

        id: 1
    select_type: SIMPLE
        table: VIEWS
        type: ALL
possible_keys: NULL
        key: TABLE_SCHEMA, TABLE_NAME
    key_len: NULL
        ref: NULL
        rows: NULL
    Extra: Using where; Open_frm_only; Scanned 0 databases
    
```

Use of constant database and table lookup values enables the server to avoid directory scans. For references to `VIEWS.TABLE_NAME`, only the `.frm` file need be opened.

```

mysql> EXPLAIN SELECT TABLE_NAME, ROW_FORMAT FROM INFORMATION_SCHEMA.TABLES\G
***** 1. row *****
        id: 1
    select_type: SIMPLE
        table: TABLES
        type: ALL
possible_keys: NULL
        key: NULL
    key_len: NULL
        ref: NULL
        rows: NULL
    Extra: Open_full_table; Scanned all databases
    
```

No lookup values are provided (there is no `WHERE` clause), so the server must scan the data directory and each database directory. For each table thus identified, the table name and row format are selected. `TABLE_NAME` requires no further table files to be opened (the `SKIP_OPEN_TABLE` optimization applies). `ROW_FORMAT` requires all table files to be opened (`OPEN_FULL_TABLE` applies). `EXPLAIN` reports `OPEN_FULL_TABLE` because it is more expensive than `SKIP_OPEN_TABLE`.

```

mysql> EXPLAIN SELECT TABLE_NAME, TABLE_TYPE FROM INFORMATION_SCHEMA.TABLES
-> WHERE TABLE_SCHEMA = 'test'\G
***** 1. row *****
        id: 1
    select_type: SIMPLE
        table: TABLES
        type: ALL
possible_keys: NULL
        key: TABLE_SCHEMA
    key_len: NULL
        ref: NULL
        rows: NULL
    Extra: Using where; Open_frm_only; Scanned 1 database
    
```

No table name lookup value is provided, so the server must scan the `test` database directory. For the `TABLE_NAME` and `TABLE_TYPE` columns, the `SKIP_OPEN_TABLE` and `OPEN_FRM_ONLY` optimizations apply, respectively. `EXPLAIN` reports `OPEN_FRM_ONLY` because it is more expensive.

```

mysql> EXPLAIN SELECT B.TABLE_NAME
-> FROM INFORMATION_SCHEMA.TABLES AS A, INFORMATION_SCHEMA.COLUMNS AS B
-> WHERE A.TABLE_SCHEMA = 'test'
-> AND A.TABLE_NAME = 't1'
-> AND B.TABLE_NAME = A.TABLE_NAME\G
***** 1. row *****
        id: 1
    select_type: SIMPLE
        table: A
        type: ALL
possible_keys: NULL
        key: TABLE_SCHEMA, TABLE_NAME
    key_len: NULL
        ref: NULL
    
```

```

        rows: NULL
        Extra: Using where; Skip_open_table; Scanned 0 databases
***** 2. row *****
        id: 1
        select_type: SIMPLE
        table: B
        type: ALL
possible_keys: NULL
        key: NULL
        key_len: NULL
        ref: NULL
        rows: NULL
        Extra: Using where; Open_frm_only; Scanned all databases;
              Using join buffer

```

For the first `EXPLAIN` output row: Constant database and table lookup values enable the server to avoid directory scans for `TABLES` values. References to `TABLES . TABLE_NAME` require no further table files.

For the second `EXPLAIN` output row: All `COLUMNS` table values are `OPEN_FRM_ONLY` lookups, so `COLUMNS . TABLE_NAME` requires the `.frm` file to be opened.

```

mysql> EXPLAIN SELECT * FROM INFORMATION_SCHEMA.COLLATIONS\G
***** 1. row *****
        id: 1
        select_type: SIMPLE
        table: COLLATIONS
        type: ALL
possible_keys: NULL
        key: NULL
        key_len: NULL
        ref: NULL
        rows: NULL
        Extra:

```

In this case, no optimizations apply because `COLLATIONS` is not one of the `INFORMATION_SCHEMA` tables for which optimizations are available.

8.2.5. Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- If your application makes several database requests to perform related updates, combining the statements into a stored routine can help performance. Similarly, if your application computes a single result based on several column values or large volumes of data, combining the computation into a UDF (user-defined function) can help performance. The resulting fast database operations are then available to be reused by other queries, applications, and even code written in different programming languages. See [Section 19.2, “Using Stored Routines \(Procedures and Functions\)”](#) and [Section 23.3, “Adding New Functions to MySQL”](#) for more information.
- To fix any compression issues that occur with `ARCHIVE` tables, use `OPTIMIZE TABLE`. See [Section 14.8, “The ARCHIVE Storage Engine”](#).
- If possible, classify reports as “live” or as “statistical”, where data needed for statistical reports is created only from summary tables that are generated periodically from the live data.
- If you have data that does not conform well to a rows-and-columns table structure, you can pack and store data into a `BLOB` column. In this case, you must provide code in your application to pack and unpack information, but this might save I/O operations to read and write the sets of related values.
- With Web servers, store images and other binary assets as files, with the path name stored in the database rather than the file itself. Most Web servers are better at caching files than database contents,

so using files is generally faster. (Although you must handle backups and storage issues yourself in this case.)

- If you need really high speed, look at the low-level MySQL interfaces. For example, by accessing the MySQL [InnoDB](#) or [MyISAM](#) storage engine directly, you could get a substantial speed increase compared to using the SQL interface.
- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replication servers to split up the load. To avoid slowing down the master while making backups, you can make backups using a slave server. See [Chapter 16, Replication](#).

8.3. Optimization and Indexes

The best way to improve the performance of [SELECT](#) operations is to create indexes on one or more of the columns that are tested in the query. The index entries act like pointers to the table rows, allowing the query to quickly determine which rows match a condition in the [WHERE](#) clause, and retrieve the other column values for those rows. All MySQL data types can be indexed.

Although it can be tempting to create an indexes for every possible column used in a query, unnecessary indexes waste space and waste time for MySQL to determine which indexes to use. You must find the right balance to achieve fast queries using the optimal set of indexes.

8.3.1. How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. If a table has 1,000 rows, this is at least 100 times faster than reading sequentially.

Most MySQL indexes ([PRIMARY KEY](#), [UNIQUE](#), [INDEX](#), and [FULLTEXT](#)) are stored in B-trees. Exceptions are that indexes on spatial data types use R-trees, and that [MEMORY](#) tables also support hash indexes.

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in [MEMORY](#) tables) are described at the end of this section.

MySQL uses indexes for these operations:

- To find the rows matching a [WHERE](#) clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most [selective](#) index).
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, [VARCHAR](#) and [CHAR](#) are considered the same if they are declared as the same size. For example, [VARCHAR\(10\)](#) and [CHAR\(10\)](#) are the same size, but [VARCHAR\(10\)](#) and [CHAR\(15\)](#) are not.

Comparison of dissimilar columns may prevent use of indexes if values cannot be compared directly without conversion. Suppose that a numeric column is compared to a string column. For a given value such as `1` in the numeric column, it might compare equal to any number of values in the string column such as `'1'`, `' 1'`, `'00001'`, or `'01.e1'`. This rules out use of any indexes for the string column.

- To find the [MIN\(\)](#) [1206] or [MAX\(\)](#) [1205] value for a specific indexed column `key_col`. This is optimized by a preprocessor that checks whether you are using [WHERE key_part_N = constant](#) on all key parts that occur before `key_col` in the index. In this case, MySQL does a single key lookup for

each `MIN()` [1206] or `MAX()` [1205] expression and replaces it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

```
SELECT MIN(key_part2),MAX(key_part2)
FROM tbl_name WHERE key_part1=10;
```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable key (for example, `ORDER BY key_part1, key_part2`). If all key parts are followed by `DESC`, the key is read in reverse order. See [Section 8.13.9, “ORDER BY Optimization”](#), and [Section 8.13.10, “GROUP BY Optimization”](#).
- In some cases, a query can be optimized to retrieve values without consulting the data rows. (An index that provides all the necessary results for a query is called a [covering index](#).) If a query uses only columns from a table that are numeric and that form a leftmost prefix for some key, the selected values can be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name
WHERE key_part1=1
```

Indexes are less important for queries on small tables, or big tables where report queries process most or all of the rows. When a query needs to access most of the rows, reading sequentially is faster than working through an index. Sequential reads minimize disk seeks, even if not all the rows are needed for the query. See [Section 8.2.1.4, “How to Avoid Full Table Scans”](#) for details.

8.3.2. Using Primary Keys

The primary key for a table represents the column or set of columns that you use in your most vital queries. It has an associated index, for fast query performance. Query performance benefits from the [NOT NULL](#) optimization, because it cannot include any `NULL` values. With the [InnoDB](#) storage engine, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key column or columns.

If your table is big and important, but does not have an obvious column or set of columns to use as a primary key, you might create a separate column with auto-increment values to use as the primary key. These unique IDs can serve as pointers to corresponding rows in other tables when you join tables using foreign keys.

8.3.3. Using Foreign Keys

If a table has many columns, and you query many different combinations of columns, it might be efficient to split the less-frequently used data into separate tables with a few columns each, and relate them back to the main table by duplicating the numeric ID column from the main table. That way, each small table can have a primary key for fast lookups of its data, and you can query just the set of columns that you need using a join operation. Depending on how the data is distributed, the queries might perform less I/O and take up less cache memory because the relevant columns are packed together on disk. (To maximize performance, queries try to read as few data blocks as possible from disk; tables with only a few columns can fit more rows in each data block.)

8.3.4. Column Indexes

The most common type of index involves a single column, storing copies of the values from that column in a data structure, allowing fast lookups for the rows with the corresponding column values. The B-tree data structure lets the index quickly find a specific value, a set of values, or a range of values, corresponding to operators such as `=`, `>`, `≤`, `BETWEEN`, `IN`, and so on, in a `WHERE` clause.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See [Chapter 14, Storage Engines](#). All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

Prefix Indexes

With `col_name(N)` syntax in an index specification, you can create an index that uses only the first *N* characters of a string column. Indexing only a prefix of column values in this way can make the index file much smaller. When you index a `BLOB` or `TEXT` column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for `InnoDB` tables).



Note

Prefix limits are measured in bytes, while the prefix length in `CREATE TABLE` statements is interpreted as number of characters. *Take this into account when specifying a prefix length for a column that uses a multi-byte character set.*

FULLTEXT Indexes

You can also create `FULLTEXT` indexes. These are used for full-text searches. Only the `MyISAM` storage engine supports `FULLTEXT` indexes and only for `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always takes place over the entire column and column prefix indexing is not supported. For details, see [Section 12.9, “Full-Text Search Functions”](#).

Spatial Indexes

You can also create indexes on spatial data types. Currently, only `MyISAM` supports R-tree indexes on spatial types. Other storage engines use B-trees for indexing spatial types (except for `ARCHIVE`, which does not support spatial type indexing).

Indexes in the MEMORY Storage Engine

The `MEMORY` storage engine uses `HASH` indexes by default, but also supports `BTREE` indexes.

8.3.5. Multiple-Column Indexes

MySQL can create composite indexes (that is, indexes on multiple columns). An index may consist of up to 16 columns. For certain data types, you can index a prefix of the column (see [Section 8.3.4, “Column Indexes”](#)).

MySQL can use multiple-column indexes for queries that test all the columns in the index, or queries that test just the first column, the first two columns, the first three columns, and so on. If you specify the columns in the right order in the index definition, a single composite index can speed up several kinds of queries on the same table.

A multiple-column index can be considered a sorted array, the rows of which contain values that are created by concatenating the values of the indexed columns.



Note

As an alternative to a composite index, you can introduce a column that is “hashed” based on information from other columns. If this column is short, reasonably unique, and indexed, it might be faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:

```
SELECT * FROM tbl_name
WHERE hash_col=MD5(CONCAT(val1,val2))
```

```
AND col1=val1 AND col2=val2;
```

Suppose that a table has the following specification:

```
CREATE TABLE test (
  id          INT NOT NULL,
  last_name   CHAR(30) NOT NULL,
  first_name  CHAR(30) NOT NULL,
  PRIMARY KEY (id),
  INDEX name  (last_name,first_name)
);
```

The `name` index is an index over the `last_name` and `first_name` columns. The index can be used for lookups in queries that specify values in a known range for combinations of `last_name` and `first_name` values. It can also be used for queries that specify just a `last_name` value because that column is a leftmost prefix of the index (see [Section 8.3.5, “Multiple-Column Indexes”](#)). Therefore, the `name` index is used for lookups in the following queries:

```
SELECT * FROM test WHERE last_name='Widenius';

SELECT * FROM test
  WHERE last_name='Widenius' AND first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius'
    AND (first_name='Michael' OR first_name='Monty');

SELECT * FROM test
  WHERE last_name='Widenius'
    AND first_name >='M' AND first_name < 'N';
```

However, the `name` index is *not* used for lookups in the following queries:

```
SELECT * FROM test WHERE first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius' OR first_name='Michael';
```

Suppose that you issue the following `SELECT` statement:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on `col1` and `col2`, the appropriate rows can be fetched directly. If separate single-column indexes exist on `col1` and `col2`, the optimizer attempts to use the Index Merge optimization (see [Section 8.13.2, “Index Merge Optimization”](#)), or attempts to find the most restrictive index by deciding which index excludes more rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to find rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`.

MySQL cannot use the index to perform lookups if the columns do not form a leftmost prefix of the index. Suppose that you have the `SELECT` statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on `(col1, col2, col3)`, only the first two queries use the index. The third and fourth queries do involve indexed columns, but `(col2)` and `(col2, col3)` are not leftmost prefixes of `(col1, col2, col3)`.

8.3.6. Verifying Index Usage

Always check whether all your queries really use the indexes that you have created in the tables. Use the `EXPLAIN` statement, as described in [Section 8.8.1, “Optimizing Queries with `EXPLAIN`”](#).

8.3.7. InnoDB and MyISAM Index Statistics Collection

Storage engines collect statistics about tables for use by the optimizer. Table statistics are based on value groups, where a value group is a set of rows with the same key prefix value. For optimizer purposes, an important statistic is the average value group size.

MySQL uses the average value group size in the following ways:

- To estimate how many rows must be read for each [ref \[837\]](#) access
- To estimate how many row a partial join will produce; that is, the number of rows that an operation of this form will produce:

```
(...) JOIN tbl_name ON tbl_name.key = expr
```

As the average value group size for an index increases, the index is less useful for those two purposes because the average number of rows per lookup increases: For the index to be good for optimization purposes, it is best that each index value target a small number of rows in the table. When a given index value yields a large number of rows, the index is less useful and MySQL is less likely to use it.

The average value group size is related to table cardinality, which is the number of value groups. The `SHOW INDEX` statement displays a cardinality value based on N/S , where N is the number of rows in the table and S is the average value group size. That ratio yields an approximate number of value groups in the table.

For a join based on the `<=>` comparison operator, `NULL` is not treated differently from any other value: `NULL <=> NULL`, just as `N <=> N` for any other N .

However, for a join based on the `=` operator, `NULL` is different from non-`NULL` values: `expr1 = expr2` is not true when `expr1` or `expr2` (or both) are `NULL`. This affects [ref \[837\]](#) accesses for comparisons of the form `tbl_name.key = expr`: MySQL will not access the table if the current value of `expr` is `NULL`, because the comparison cannot be true.

For `=` comparisons, it does not matter how many `NULL` values are in the table. For optimization purposes, the relevant value is the average size of the non-`NULL` value groups. However, MySQL does not currently enable that average size to be collected or used.

For `InnoDB` and `MyISAM` tables, you have some control over collection of table statistics by means of the `innodb_stats_method [1570]` and `myisam_stats_method [530]` system variables, respectively. These variables have three possible values, which differ as follows:

- When the variable is set to `nulls_equal`, all `NULL` values are treated as identical (that is, they all form a single value group).

If the `NULL` value group size is much higher than the average non-`NULL` value group size, this method skews the average value group size upward. This makes index appear to the optimizer to be less useful than it really is for joins that look for non-`NULL` values. Consequently, the `nulls_equal` method may cause the optimizer not to use the index for [ref \[837\]](#) accesses when it should.

- When the variable is set to `nulls_unequal`, `NULL` values are not considered the same. Instead, each `NULL` value forms a separate value group of size 1.

If you have many `NULL` values, this method skews the average value group size downward. If the average non-`NULL` value group size is large, counting `NULL` values each as a group of size 1 causes the optimizer to overestimate the value of the index for joins that look for non-`NULL` values. Consequently, the `nulls_unequal` method may cause the optimizer to use this index for `ref` [837] lookups when other methods may be better.

- When the variable is set to `nulls_ignored`, `NULL` values are ignored.

If you tend to use many joins that use `<=>` rather than `=`, `NULL` values are not special in comparisons and one `NULL` is equal to another. In this case, `nulls_equal` is the appropriate statistics method.

The `innodb_stats_method` [1570] system variable has a global value; the `myisam_stats_method` [530] system variable has both global and session values. Setting the global value affects statistics collection for tables from the corresponding storage engine. Setting the session value affects statistics collection only for the current client connection. This means that you can force a table's statistics to be regenerated with a given method without affecting other clients by setting the session value of `myisam_stats_method` [530].

To regenerate table statistics, you can use any of the following methods:

- Execute `myisamchk --stats_method=method_name --analyze`
- Change the table to cause its statistics to go out of date (for example, insert a row and then delete it), and then set `myisam_stats_method` [530] and issue an `ANALYZE TABLE` statement

Some caveats regarding the use of `innodb_stats_method` [1570] and `myisam_stats_method` [530]:

- You can force table statistics to be collected explicitly, as just described. However, MySQL may also collect statistics automatically. For example, if during the course of executing statements for a table, some of those statements modify the table, MySQL may collect statistics. (This may occur for bulk inserts or deletes, or some `ALTER TABLE` statements, for example.) If this happens, the statistics are collected using whatever value `innodb_stats_method` [1570] or `myisam_stats_method` [530] has at the time. Thus, if you collect statistics using one method, but the system variable is set to the other method when a table's statistics are collected automatically later, the other method will be used.
- There is no way to tell which method was used to generate statistics for a given table.
- These variables apply only to `InnoDB` and `MyISAM` tables. Other storage engines have only one method for collecting table statistics. Usually it is closer to the `nulls_equal` method.

8.3.8. Comparison of B-Tree and Hash Indexes

Understanding the B-tree and hash data structures can help predict how different queries perform on different storage engines that use these data structures in their indexes, particularly for the `MEMORY` storage engine that lets you choose B-tree or hash indexes.

B-Tree Index Characteristics

A B-tree index can be used for column comparisons in expressions that use the `=` [1089], `>` [1090], `>=` [1090], `<` [1090], `<=` [1090], or `BETWEEN` [1091] operators. The index also can be used for `LIKE` [1111] comparisons if the argument to `LIKE` [1111] is a constant string that does not start with a wildcard character. For example, the following `SELECT` statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

In the first statement, only rows with 'Patrick' <= *key_col* < 'Patricl' are considered. In the second statement, only rows with 'Pat' <= *key_col* < 'Pau' are considered.

The following `SELECT` statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

In the first statement, the `LIKE [1111]` value begins with a wildcard character. In the second statement, the `LIKE [1111]` value is not a constant.

If you use `... LIKE '%string%'` and *string* is longer than three characters, MySQL uses the *Turbo Boyer-Moore algorithm* to initialize the pattern for the string and then uses this pattern to perform the search more quickly.

A search using `col_name IS NULL` employs indexes if *col_name* is indexed.

Any index that does not span all `AND [1094]` levels in the `WHERE` clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every `AND [1094]` group.

The following `WHERE` clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3
/* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2
/* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5
/* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

These `WHERE` clauses do *not* use indexes:

```
/* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

/* Index is not used in both parts of the WHERE clause */
... WHERE index=1 OR A=10

/* No index spans all rows */
... WHERE index_part1=1 OR index_part2=10
```

Sometimes MySQL does not use an index, even if one is available. One circumstance under which this occurs is when the optimizer estimates that using the index would require MySQL to access a very large percentage of the rows in the table. (In this case, a table scan is likely to be much faster because it requires fewer seeks.) However, if such a query uses `LIMIT` to retrieve only some of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash Index Characteristics

Hash indexes have somewhat different characteristics from those just discussed:

- They are used only for equality comparisons that use the `=` or `<=>` operators (but are *very fast*). They are not used for comparison operators such as `<` that find a range of values. Systems that rely on this type of single-value lookup are known as “key-value stores”; to use MySQL for such applications, use hash indexes wherever possible.
- The optimizer cannot use a hash index to speed up `ORDER BY` operations. (This type of index cannot be used to search for the next entry in order.)
- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a `MyISAM` table to a hash-indexed `MEMORY` table.

- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

8.4. Optimizing Database Structure

In your role as a database designer, look for the most efficient way to organize your schemas, tables, and columns. As when tuning application code, you minimize I/O, keep related items together, and plan ahead so that performance stays high as the data volume increases. Starting with an efficient database design makes it easier for team members to write high-performing application code, and makes the database likely to endure as applications evolve and are rewritten.

8.4.1. Optimizing Data Size

Design your tables to minimize their space on the disk. This can result in huge improvements by reducing the amount of data written to and read from disk. Smaller tables normally require less main memory while their contents are being actively processed during query execution. Any space reduction for table data also results in smaller indexes that can be processed faster.

MySQL supports many different storage engines (table types) and row formats. For each table, you can decide which storage and indexing method to use. Choosing the proper table format for your application can give you a big performance gain. See [Chapter 14, Storage Engines](#).

You can get better performance for a table and minimize storage space by using the techniques listed here:

Table Columns

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory. For example, use the smaller integer types if possible to get smaller tables. `MEDIUMINT` is often a better choice than `INT` because a `MEDIUMINT` column uses 25% less space.
- Declare columns to be `NOT NULL` if possible. It makes SQL operations faster, by enabling better use of indexes and eliminating overhead for testing whether each value is `NULL`. You also save some storage space, one bit per column. If you really need `NULL` values in your tables, use them. Just avoid the default setting that allows `NULL` values in every column.

Row Format

- `InnoDB` tables use a compact storage format. In versions of MySQL earlier than 5.0.3, `InnoDB` rows contain some redundant information, such as the number of columns and the length of each column, even for fixed-size columns. By default, tables are created in the compact format (`ROW_FORMAT=COMPACT`). If you wish to downgrade to older versions of MySQL, you can request the old format with `ROW_FORMAT=REDUNDANT`.

The presence of the compact row format decreases row storage space by about 20% at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed it is likely to be faster. If it is a rare case that is limited by CPU speed, it might be slower.

The compact `InnoDB` format also changes how `CHAR` columns containing UTF-8 data are stored. With `ROW_FORMAT=REDUNDANT`, a UTF-8 `CHAR(N)` occupies $3 \times N$ bytes, given that the maximum length of a UTF-8 encoded character is three bytes. Many languages can be written primarily using single-byte UTF-8 characters, so a fixed storage length often wastes space. With `ROW_FORMAT=COMPACT` format, `InnoDB` allocates a variable amount of storage in the range from N to $3 \times N$ bytes for these columns by

stripping trailing spaces if necessary. The minimum storage length is kept as *N* bytes to facilitate in-place updates in typical cases.

- To minimize space even further by storing table data in compressed form, specify `ROW_FORMAT=COMPRESSED` when creating InnoDB tables, or run the `myisampack` command on an existing MyISAM table. (InnoDB tables compressed tables are readable and writable, while MyISAM compressed tables are read-only.)
- For MyISAM tables, if you do not have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size row format is used. This is faster but may waste some space. See [Section 14.5.3, “MyISAM Table Storage Formats”](#). You can hint that you want to have fixed length rows even if you have `VARCHAR` columns with the `CREATE TABLE` option `ROW_FORMAT=FIXED`.

Indexes

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient. For InnoDB tables, the primary key columns are duplicated in each secondary index entry, so a short primary key saves considerable space if you have many secondary indexes.
- Create only the indexes that you need to improve query performance. Indexes are good for retrieval, but slow down insert and update operations. If you access a table mostly by searching on a combination of columns, create a single composite index on them rather than a separate index for each column. The first part of the index should be the column most used. If you *always* use many columns when selecting from the table, the first column in the index should be the one with the most duplicates, to obtain better compression of the index.
- If it is very likely that a long string column has a unique prefix on the first number of characters, it is better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see [Section 13.1.13, “CREATE INDEX Syntax”](#)). Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks. See [Section 8.11.2, “Tuning Server Parameters”](#).

Joins

- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic-format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.
- Declare columns with identical information in different tables with identical data types, to speed up joins based on the corresponding columns.
- Keep column names simple, so that you can use the same name across different tables and simplify join queries. For example, in a table named `customer`, use a column name of `name` instead of `customer_name`. To make your names portable to other SQL servers, consider keeping them shorter than 18 characters.

Normalization

- Normally, try to keep all data nonredundant (observing what is referred to in database theory as *third normal form*). Instead of repeating lengthy values such as names and addresses, assign them unique IDs, repeat these IDs as needed across multiple smaller tables, and join the tables in queries by referencing the IDs in the join clause.
- If speed is more important than disk space and the maintenance costs of keeping multiple copies of data, for example in a business intelligence scenario where you analyze all the data from large tables, you can relax the normalization rules, duplicating information or creating summary tables to gain more speed.

8.4.2. Optimizing MySQL Data Types

8.4.2.1. Optimizing for Numeric Data

- For unique IDs or other values that can be represented as either strings or numbers, prefer numeric columns to string columns. Since large numeric values can be stored in fewer bytes than the corresponding strings, it is faster and takes less memory to transfer and compare them.
- If you are using numeric data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you can avoid parsing the text file to find line and column boundaries.

8.4.2.2. Optimizing for Character and String Types

For character and string columns, follow these guidelines:

- Use binary collation order for fast comparison and sort operations, when you do not need language-specific collation features. You can use the [BINARY](#) operator to use binary collation within a particular query.
- When comparing values from different columns, declare those columns with the same character set and collation wherever possible, to avoid string conversions while running the query.
- For column values less than 8KB in size, use binary [VARCHAR](#) instead of [BLOB](#). The [GROUP BY](#) and [ORDER BY](#) clauses can generate temporary tables, and these temporary tables can use the [MEMORY](#) storage engine if the original table does not contain any [BLOB](#) columns.
- If a table contains string columns such as name and address, but many queries do not retrieve those columns, consider splitting the string columns into a separate table and using join queries with a foreign key when necessary. When MySQL retrieves any value from a row, it reads a data block containing all the columns of that row (and possibly other adjacent rows). Keeping each row small, with only the most frequently used columns, allows more rows to fit in each data block. Such compact tables reduce disk I/O and memory usage for common queries.
- When you use a randomly generated value as a primary key in an [InnoDB](#) table, prefix it with an ascending value such as the current date and time if possible. When consecutive primary values are physically stored near each other, [InnoDB](#) can insert and retrieve them faster.
- See [Section 8.4.2.1, “Optimizing for Numeric Data”](#) for reasons why a numeric column is usually preferable to an equivalent string column.

8.4.2.3. Optimizing for BLOB Types

- When storing a large blob containing textual data, consider compressing it first. Do not use this technique when the entire table is compressed by [InnoDB](#) or [MyISAM](#).
- For a table with several columns, to reduce memory requirements for queries that do not use the BLOB column, consider splitting the BLOB column into a separate table and referencing it with a join query when needed.
- Since the performance requirements to retrieve and display a BLOB value might be very different from other data types, you could put the BLOB-specific table on a different storage device or even a separate

database instance. For example, to retrieve a BLOB might require a large sequential disk read that is better suited to a traditional hard drive than to an [SSD device](#).

- See [Section 8.4.2.2, “Optimizing for Character and String Types”](#) for reasons why a binary `VARCHAR` column is sometimes preferable to an equivalent BLOB column.
- Rather than testing for equality against a very long text string, you can store a hash of the column value in a separate column, index that column, and test the hashed value in queries. (Use the `MD5()` or `CRC32()` function to produce the hash value.) Since hash functions can produce duplicate results for different inputs, you still include a clause `AND blob_column = long_string_value` in the query to guard against false matches; the performance benefit comes from the smaller, easily scanned index for the hashed values.

8.4.2.4. Using `PROCEDURE ANALYSE`

```
ANALYSE([max_elements[,max_memory]])
```

`ANALYSE()` examines the result from a query and returns an analysis of the results that suggests optimal data types for each column that may help reduce table sizes. To obtain this analysis, append `PROCEDURE ANALYSE` to the end of a `SELECT` statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that `PROCEDURE ANALYSE()` does not suggest the `ENUM` data type when it is not appropriate.

The arguments are optional and are used as follows:

- `max_elements` (default 256) is the maximum number of distinct values that `ANALYSE()` notices per column. This is used by `ANALYSE()` to check whether the optimal data type should be of type `ENUM`; if there are more than `max_elements` distinct values, then `ENUM` is not a suggested type.
- `max_memory` (default 8192) is the maximum amount of memory that `ANALYSE()` should allocate per column while trying to find all distinct values.

8.4.3. Optimizing for Many Tables

Some techniques for keeping individual queries fast involve splitting data across many tables. When the number of tables runs into the thousands or even millions, the overhead of dealing with all these tables becomes a new performance consideration.

8.4.3.1. How MySQL Opens and Closes Tables

When you execute a `mysqladmin status` command, you should see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The `Open tables` value of 12 can be somewhat puzzling if you have only six tables.

MySQL is multi-threaded, so there may be many clients issuing queries for a given table simultaneously. To minimize the problem with multiple client sessions having different states on the same table, the table

is opened independently by each concurrent session. This uses additional memory but normally increases performance. With `MyISAM` tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all sessions.)

The `table_open_cache` [561] and `max_connections` [519] system variables affect the maximum number of files the server keeps open. If you increase one or both of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems permit you to increase the open-files limit, although the method varies widely from system to system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

`table_open_cache` [561] is related to `max_connections` [519]. For example, for 200 concurrent running connections, specify a table cache size of at least $200 * N$, where N is the maximum number of tables per join in any of the queries which you execute. You must also reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the `table_open_cache` [561] setting. If `table_open_cache` [561] is set too high, MySQL may run out of file descriptors and refuse connections, fail to perform queries, and be very unreliable. You also have to take into account that the `MyISAM` storage engine needs two file descriptors for each unique open table. You can increase the number of file descriptors available to MySQL using the `--open-files-limit` [456] startup option to `mysqld`. See [Section C.5.2.18, “'File' Not Found and Similar Errors”](#).

The cache of open tables is kept at a level of `table_open_cache` [561] entries. The default value is 400; this can be changed with the `--table_open_cache` [561] option to `mysqld`. Note that MySQL may temporarily open more tables than this to execute queries.

MySQL closes an unused table and removes it from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.
- When the cache contains more than `table_open_cache` [561] entries and a table in the cache is no longer being used by any threads.
- When a table flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a `mysqladmin flush-tables` or `mysqladmin refresh` command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables that are not currently in use are released, beginning with the table least recently used.
- If a new table needs to be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary. When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A `MyISAM` table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any `MyISAM` table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the `HANDLER tbl_name OPEN` statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls `HANDLER tbl_name CLOSE` or the thread terminates. When this happens, the table is put back in the table cache (if the cache is not full). See [Section 13.2.4, “HANDLER Syntax”](#).

You can determine whether your table cache is too small by checking the `mysqld` status variable `Opened_tables` [604], which indicates the number of table-opening operations since the server started:

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 2741  |
+-----+-----+
```

If the value is very large or increases rapidly, even when you have not issued many `FLUSH TABLES` statements, increase the table cache size. See [Section 5.1.4, “Server System Variables”](#), and [Section 5.1.6, “Server Status Variables”](#).

8.4.3.2. Disadvantages of Creating Many Tables in the Same Database

If you have many `MyISAM` tables in the same database directory, open, close, and create operations are slow. If you execute `SELECT` statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by increasing the number of entries permitted in the table cache.

8.4.3.3. How MySQL Uses Internal Temporary Tables

In some cases, the server creates internal temporary tables while processing queries. Such a table can be held in memory and processed by the `MEMORY` storage engine, or stored on disk and processed by the `MyISAM` storage engine. The server may create a temporary table initially as an in-memory table, then convert it to an on-disk table if it becomes too large. Users have no direct control over when the server creates an internal temporary table or which storage engine the server uses to manage it.

Temporary tables can be created under conditions such as these:

- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- `DISTINCT` combined with `ORDER BY` may require a temporary table.
- If you use the `SQL_SMALL_RESULT` option, MySQL uses an in-memory temporary table, unless the query also contains elements (described later) that require on-disk storage.

To determine whether a query requires a temporary table, use `EXPLAIN` and check the `Extra` column to see whether it says `Using temporary`. See [Section 8.8.1, “Optimizing Queries with EXPLAIN”](#).

Some conditions prevent the use of an in-memory temporary table, in which case the server uses an on-disk table instead:

- Presence of a `BLOB` or `TEXT` column in the table
- Presence of any column in a `GROUP BY` or `DISTINCT` clause larger than 512 bytes
- Presence of any column larger than 512 bytes in the `SELECT` list, if `UNION` or `UNION ALL` is used

If an internal temporary table is created initially as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the `tmp_table_size` [568] and `max_heap_table_size` [521] values. This differs from `MEMORY` tables explicitly created with `CREATE TABLE`: For such tables, the `max_heap_table_size` [521] system variable determines how large the table is permitted to grow and there is no conversion to on-disk format.

When the server creates an internal temporary table (either in memory or on disk), it increments the `Created_tmp_tables` [598] status variable. If the server creates the table on disk (either initially or by converting an in-memory table) it increments the `Created_tmp_disk_tables` [598] status variable.

8.5. Optimizing for InnoDB Tables

InnoDB is the storage engine that MySQL customers typically use in production databases where reliability and concurrency are important. Because InnoDB is the default storage engine in MySQL 5.5 and higher, you can expect to see InnoDB tables more often than before. This section explains how to optimize database operations for InnoDB tables.

8.5.1. Optimizing Storage Layout for InnoDB Tables

- Once your data reaches a stable size, or a growing table has increased by tens or some hundreds of megabytes, consider using the `OPTIMIZE TABLE` statement to reorganize the table and compact any wasted space. The reorganized tables require less disk I/O to perform full table scans. This is a straightforward technique that can improve performance when other techniques such as improving index usage or tuning application code are not practical.

`OPTIMIZE TABLE` copies the data part of the table and rebuilds the indexes. The benefits come from improved packing of data within indexes, and reduced fragmentation within the tablespaces and on disk. The benefits vary depending on the data in each table. You may find that there are significant gains for some and not for others, or that the gains decrease over time until you next optimize the table. This operation can be slow if the table is large or if the indexes being rebuilt don't fit into the buffer pool. The first run after adding a lot of data to a table is often much slower than later runs.

- In InnoDB, having a long `PRIMARY KEY` (either a single column with a lengthy value, or several columns that form a long composite value) wastes a lot of disk space. The primary key value for a row is duplicated in all the secondary index records that point to the same row. (See [Section 14.3.11, “InnoDB Table and Index Structures”](#).) Create an `AUTO_INCREMENT` column as the primary key if your primary key is long, or index a prefix of a long `VARCHAR` column instead of the entire column.
- Use the `VARCHAR` data type instead of `CHAR` to store variable-length strings or for columns with many `NULL` values. A `CHAR(N)` column always takes `N` characters to store data, even if the string is shorter or its value is `NULL`. Smaller tables fit better in the buffer pool and reduce disk I/O.

When using `COMPACT` row format (the default InnoDB format in MySQL 5.5) and variable-length character sets, such as `utf8` or `sjis`, `CHAR(N)` columns occupy a variable amount of space, but still at least `N` bytes.

- For tables that are big, or contain lots of repetitive text or numeric data, consider using `COMPRESSED` row format. Less disk I/O is required to bring data into the buffer pool, or to perform full table scans. Before making a permanent decision, measure the amount of compression you can achieve by using `COMPRESSED` versus `COMPACT` row format.

8.5.2. Optimizing InnoDB Transaction Management

To optimize InnoDB transaction processing, find the ideal balance between the performance overhead of transactional features and the workload of your server. For example, an application might encounter performance issues if it commits thousands of times per second, and different performance issues if it commits only every 2-3 hours.

- The default MySQL setting `AUTOCOMMIT=1` can impose performance limitations on a busy database server. Where practical, wrap several related DML operations into a single transaction, by issuing `SET AUTOCOMMIT=0` or a `START TRANSACTION` statement, followed by a `COMMIT` statement after making all the changes.

InnoDB must flush the log to disk at each transaction commit if that transaction made modifications to the database. When each change is followed by a commit (as with the default autocommit setting), the I/O throughput of the storage device puts a cap on the number of potential operations per second.

- Avoid performing rollbacks after inserting, updating, or deleting huge numbers of rows. If a big transaction is slowing down server performance, rolling it back can make the problem worse, potentially taking several times as long to perform as the original DML operations. Killing the database process does not help, because the rollback starts again on server startup.

To minimize the chance of this issue occurring: increase the size of the buffer pool so that all the DML changes can be cached rather than immediately written to disk; set `innodb_change_buffering=all` so that update and delete operations are buffered in addition to inserts; and consider issuing `COMMIT` statements periodically during the big DML operation, possibly breaking a single delete or update into multiple statements that operate on smaller numbers of rows.

To get rid of a runaway rollback once it occurs, increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or kill the server and restart with `innodb_force_recovery=3`, as explained in [Section 14.3.7.1, “The InnoDB Recovery Process”](#).

This issue is expected to be less prominent in MySQL 5.5 and up, or in MySQL 5.1 with the InnoDB Plugin, because the default setting `innodb_change_buffering=all` allows update and delete operations to be cached in memory, making them faster to perform in the first place, and also faster to roll back if needed. Make sure to use this parameter setting on servers that process long-running transactions with many inserts, updates, or deletes.

- If you can afford the loss of some of the latest committed transactions if a crash occurs, you can set the `innodb_flush_log_at_trx_commit` [1556] parameter to 0. InnoDB tries to flush the log once per second anyway, although the flush is not guaranteed. Also, set the value of `innodb_support_xa` [1572] to 0, which will reduce the number of disk flushes due to synchronizing on disk data and the binary log.
- When rows are modified or deleted, the rows and associated undo logs are not physically removed immediately, or even immediately after the transaction commits. The old data is preserved until transactions that started earlier or concurrently are finished, so that those transactions can access the previous state of modified or deleted rows. Thus, a long-running transaction can prevent InnoDB from purging data that was changed by a different transaction.
- When rows are modified or deleted within a long-running transaction, other transactions using the `READ COMMITTED` [1402] and `REPEATABLE READ` [1401] isolation levels have to do more work to reconstruct the older data if they read those same rows.
- When a long-running transaction modifies a table, queries against that table from other transactions do not make use of the `covering index` technique. Queries that normally could retrieve all the result columns from a secondary index, instead look up the appropriate values from the table data.

8.5.3. Optimizing InnoDB Logging

- Make your log files big, even as big as the buffer pool. When InnoDB has written the log files full, it must write the modified contents of the buffer pool to disk in a checkpoint. Small log files cause many unnecessary disk writes. Although historically big log files caused lengthy recovery times, recovery is now much faster and you can confidently use large log files.
- Make the log buffer quite large as well (on the order of 8MB).

8.5.4. Bulk Data Loading for InnoDB Tables

These performance tips supplement the general guidelines for fast inserts in [Section 8.2.2.1, “Speed of INSERT Statements”](#).

- When importing data into InnoDB, turn off autocommit mode, because it performs a log flush to disk for every insert. To disable autocommit during your import operation, surround it with `SET autocommit` and `COMMIT` statements:

```
SET autocommit=0;... SQL import statements ...
COMMIT;
```

The `mysqldump` option `--opt` [319] creates dump files that are fast to import into an InnoDB table, even without wrapping them with the `SET autocommit` and `COMMIT` statements.

- If you have `UNIQUE` constraints on secondary keys, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

```
SET unique_checks=0;... SQL import statements ...
SET unique_checks=1;
```

For big tables, this saves a lot of disk I/O because InnoDB can use its insert buffer to write secondary index records in a batch. Be certain that the data contains no duplicate keys.

- If you have `FOREIGN KEY` constraints in your tables, you can speed up table imports by turning off the foreign key checks for the duration of the import session:

```
SET foreign_key_checks=0;... SQL import statements ...
SET foreign_key_checks=1;
```

For big tables, this can save a lot of disk I/O.

- Use the multiple-row `INSERT` syntax to reduce communication overhead between the client and the server if you need to insert many rows:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

This tip is valid for inserts into any table, not just InnoDB tables.

- When doing bulk inserts into tables with auto-increment columns, set `innodb_autoinc_lock_mode` [1548] to 2 instead of the default value 1. See [Section 14.3.5.3.2, “Configurable InnoDB Auto-Increment Locking”](#) for details.

8.5.5. Optimizing InnoDB Queries

To tune queries for InnoDB tables, create an appropriate set of indexes on each table. See [Section 8.3.1, “How MySQL Uses Indexes”](#) for details. Follow these guidelines for InnoDB indexes:

- Because each InnoDB table has a primary key (whether you request one or not), specify a set of primary key columns for each table, columns that are used in the most important and time-critical queries.
- Do not specify too many or too long columns in the primary key, because these column values are duplicated in each secondary index.
- Do not create a separate secondary index for each column, because each query can only make use of one index. Indexes on rarely tested columns or columns with only a few different values might not be helpful for any queries. If you have many queries for the same table, testing different combinations of columns, try to create a small number of [concatenated indexes](#) rather than a large number of single-column indexes. If an index contains all the columns needed for the result set (known as a [covering index](#)), the query might be able to avoid reading the table data at all.

- If an indexed column cannot contain any `NULL` values, declare it as `NOT NULL` when you create the table. The optimizer can better determine which index is most effective to use for a query, when it knows whether each column contains `NULL` values or not.
- If you often have recurring queries for tables that are not updated frequently, enable the query cache:

```
[mysqld]
query_cache_type = 1
query_cache_size = 10M
```

8.5.6. Optimizing InnoDB DDL Operations

- For DDL operations on tables and indexes (`CREATE`, `ALTER`, and `DROP` statements), the most significant aspect for InnoDB tables is that creating and dropping secondary indexes is much faster in MySQL 5.5 and higher, than in MySQL 5.1 and before. See [Section 14.4.2, “Fast Index Creation in the InnoDB Storage Engine”](#) for details.
- “Fast index creation” makes it faster in some cases to drop an index before loading data into a table, then re-create the index after loading the data.
- Use `TRUNCATE TABLE` to empty a table, not `DELETE FROM tbl_name`. Foreign key constraints can make a `TRUNCATE` statement work like a regular `DELETE` statement, in which case a sequence of commands like `DROP TABLE` and `CREATE TABLE` might be fastest.
- Because the primary key is integral to the storage layout of each InnoDB table, and changing the definition of the primary key involves reorganizing the whole table, always set up the primary key as part of the `CREATE TABLE` statement, and plan ahead so that you do not need to `ALTER` or `DROP` the primary key afterward.

8.5.7. Optimizing InnoDB Disk I/O

If you follow the best practices for database design and the tuning techniques for SQL operations, but your database is still slowed by heavy disk I/O activity, explore these low-level techniques related to disk I/O. If the Unix `top` tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound.

- When table data is cached in the InnoDB buffer pool, it can be processed over and over by queries without requiring any disk I/O. Specify the size of the buffer pool with the `innodb_buffer_pool_size` [1549] option. This memory area is important enough that busy databases often specify a size approximately 80% of the amount of physical memory. For more information, see [Section 8.9.1, “The InnoDB Buffer Pool”](#).
- In some versions of GNU/Linux and Unix, flushing files to disk with the Unix `fsync()` call (which InnoDB uses by default) and similar methods is surprisingly slow. If database write performance is an issue, conduct benchmarks with the `innodb_flush_method` [1557] parameter set to `O_DSYNC`.
- When using the InnoDB storage engine on Solaris 10 for x86_64 architecture (AMD Opteron), use direct I/O for InnoDB-related files, to avoid degradation of InnoDB performance. To use direct I/O for an entire UFS file system used for storing InnoDB-related files, mount it with the `forcedirectio` option; see `mount_ufs(1M)`. (The default on Solaris 10/x86_64 is *not* to use this option.) To apply direct I/O only to InnoDB file operations rather than the whole file system, set `innodb_flush_method = O_DIRECT` [1557]. With this setting, InnoDB calls `directio()` instead of `fcntl()` for I/O to data files (not for I/O to log files).
- When using the InnoDB storage engine with a large `innodb_buffer_pool_size` [1549] value on any release of Solaris 2.6 and up and any platform (sparc/x86/x64/amd64), conduct benchmarks with InnoDB data files and log files on raw devices or on a separate direct I/O UFS file system, using the

`forcedirectio` mount option as described earlier. (It is necessary to use the mount option rather than setting `innodb_flush_method` [1557] if you want direct I/O for the log files.) Users of the Veritas file system VxFS should use the `convosync=direct` mount option.

Do not place other MySQL data files, such as those for `MyISAM` tables, on a direct I/O file system. Executables or libraries *must not* be placed on a direct I/O file system.

- If you have additional storage devices available to set up a RAID configuration or symbolic links to different disks, [Section 8.11.3, “Optimizing Disk I/O”](#) for additional low-level I/O tips.

8.5.8. Optimizing InnoDB Configuration Variables

Different settings work best for servers with light, predictable loads, versus servers that are running near full capacity all the time, or that experience spikes of high activity.

Because the `InnoDB` storage engine performs many of its optimizations automatically, many performance-tuning tasks involve monitoring to ensure that the database is performing well, and changing configuration options when performance drops. See [Section 14.4.7.17, “Integration with MySQL PERFORMANCE_SCHEMA”](#) for information about detailed `InnoDB` performance monitoring.

For information about the most important and most recent `InnoDB` performance features, see [Section 14.4.7, “InnoDB Performance and Scalability Enhancements”](#). Even if you have used `InnoDB` tables in prior versions, these features might be new to you, because they are from the “`InnoDB` Plugin”. The Plugin co-existed alongside the built-in `InnoDB` in MySQL 5.1, and becomes the default storage engine in MySQL 5.5 and higher.

The main configuration steps you can perform include:

- Enabling `InnoDB` to use high-performance memory allocators on systems that include them. See [Section 14.4.7.3, “Using Operating System Memory Allocators”](#).
- Controlling the types of DML operations for which `InnoDB` buffers the changed data, to avoid frequent small disk writes. See [Section 14.4.7.4, “Controlling InnoDB Change Buffering”](#). Because the default is to buffer all types of DML operations, only change this setting if you need to reduce the amount of buffering.
- Turning the adaptive hash indexing feature on and off. See [Section 14.4.7.5, “Controlling Adaptive Hash Indexing”](#). You might change this setting during periods of unusual activity, then restore it to its original setting.
- Setting a limit on the number of concurrent threads that `InnoDB` processes, if context switching is a bottleneck. See [Section 14.4.7.6, “Changes Regarding Thread Concurrency”](#).
- Controlling the amount of prefetching that `InnoDB` does with its read-ahead operations. When the system has unused I/O capacity, more read-ahead can improve the performance of queries. Too much read-ahead can cause periodic drops in performance on a heavily loaded system. See [Section 14.4.7.7, “Changes in the Read-Ahead Algorithm”](#).
- Increasing the number of background threads for read or write operations, if you have a high-end I/O subsystem that is not fully utilized by the default values. See [Section 14.4.7.8, “Multiple Background I/O Threads”](#).
- Controlling how much I/O `InnoDB` performs in the background. See [Section 14.4.7.11, “Controlling the Master Thread I/O Rate”](#). The amount of background I/O is higher than in MySQL 5.1, so you might scale back this setting if you observe periodic drops in performance.
- Controlling the algorithm that determines when `InnoDB` performs certain types of background writes. See [Section 14.4.7.12, “Controlling the Flushing Rate of Dirty Pages”](#). The algorithm works for some

types of workloads but not others, so might turn off this setting if you observe periodic drops in performance.

- Taking advantage of multicore processors and their cache memory configuration, to minimize delays in context switching. See [Section 14.4.7.14, “Control of Spin Lock Polling”](#).
- Preventing one-time operations such as table scans from interfering with the frequently accessed data stored in the InnoDB buffer cache. See [Section 14.4.7.15, “Making Buffer Pool Scan Resistant”](#).
- Adjusting your log files to a size that makes sense for reliability and crash recovery. See [Section 14.4.7.16, “Improvements to Crash Recovery Performance”](#). Historically, people have kept their InnoDB log files small to avoid long startup times after a crash. Internal improvements in InnoDB make startup much faster, so the log file size is not such a performance factor anymore. If your log files are artificially small, increasing the size can help performance by reducing the I/O that occurs as redo log records are recycled.
- Configuring the size and number of instances for the InnoDB buffer pool, especially important for systems with multi-gigabyte buffer pools. See [Section 14.4.7.18, “Improvements to Performance from Multiple Buffer Pools”](#).
- Increasing the maximum number of concurrent transactions, which dramatically improves scalability for the busiest databases. See [Section 14.4.7.19, “Better Scalability with Multiple Rollback Segments”](#). Although this feature does not require any action during day-to-day operation, you must perform a [slow shutdown](#) during or after upgrading the database to MySQL 5.5 to enable the higher limit.
- Moving purge operations (a type of garbage collection) into a background thread. See [Section 14.4.7.20, “Better Scalability with Improved Purge Scheduling”](#). To effectively measure the results of this setting, tune the other I/O-related and thread-related configuration settings first.
- Reducing the amount of switching that InnoDB does between concurrent threads, so that SQL operations on a busy server do not queue up and form a “traffic jam”. Set a value for the `innodb_thread_concurrency [1573]` option, up to approximately 32 for a high-powered modern system. Increase the value for the `innodb_concurrency_tickets [1552]` option, typically to 5000 or so. This combination of options sets a cap on the number of threads that InnoDB processes at any one time, and allows each thread to do substantial work before being swapped out, so that the number of waiting threads stays low and operations can complete without excessive context switching.

8.5.9. Optimizing InnoDB for Systems with Many Tables

- InnoDB computes index [cardinality](#) values for a table the first time that table is accessed after startup, instead of storing such values in the table. This step can take significant time on systems that partition the data into many tables. Since this overhead only applies to the initial table open operation, to “warm up” a table for later use, access it immediately after startup by issuing a statement such as `SELECT 1 FROM tbl_name LIMIT 1`.

8.6. Optimizing for MyISAM Tables

The MyISAM storage engine performs best with read-mostly data or with low-concurrency operations, because table locks limit the ability to perform simultaneous updates. In MySQL 5.5, InnoDB is the default storage engine rather than MyISAM.

8.6.1. Optimizing MyISAM Queries

Some general tips for speeding up queries on MyISAM tables:

- To help MySQL better optimize queries, use `ANALYZE TABLE` or run `myisamchk --analyze` on a table after it has been loaded with data. This updates a value for each index part that indicates the

average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a nonconstant expression. You can check the result from the table analysis by using `SHOW INDEX FROM tbl_name` and examining the `Cardinality` value. `myisamchk --description --verbose` shows index distribution information.

- To sort an index and data according to an index, use `myisamchk --sort-index --sort-records=1` (assuming that you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all rows in order according to the index. The first time you sort a large table this way, it may take a long time.
- Try to avoid complex `SELECT` queries on MyISAM tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.
- MyISAM supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. If it is important to be able to do this, consider using the table in ways that avoid deleting rows. Another possibility is to run `OPTIMIZE TABLE` to defragment the table after you have deleted a lot of rows from it. This behavior is altered by setting the `concurrent_insert` [490] variable. You can force new rows to be appended (and therefore permit concurrent inserts), even in tables that have deleted rows. See [Section 8.10.3, “Concurrent Inserts”](#).
- For MyISAM tables that change frequently, try to avoid all variable-length columns (`VARCHAR`, `BLOB`, and `TEXT`). The table uses dynamic row format if it includes even a single variable-length column. See [Chapter 14, Storage Engines](#).
- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a MyISAM table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See [Chapter 14, Storage Engines](#).
- Use `ALTER TABLE ... ORDER BY expr1, expr2, ...` if you usually retrieve rows in `expr1, expr2, ...` order. By using this option after extensive changes to the table, you may be able to get higher performance.
- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

```
UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
```

This is very important when you use MySQL storage engines such as MyISAM that has only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.

- Use `INSERT DELAYED` when you do not need to know when your data is written. This reduces the overall insertion impact because many rows can be written with a single disk write.
- Use `INSERT LOW_PRIORITY` when you want to give `SELECT` statements higher priority than your inserts.

Use `SELECT HIGH_PRIORITY` to get retrievals that jump the queue. That is, the `SELECT` is executed even if there is another client waiting to do a write.

`LOW_PRIORITY` and `HIGH_PRIORITY` have an effect only for storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

- Use `OPTIMIZE TABLE` periodically to avoid fragmentation with dynamic-format MyISAM tables. See [Section 14.5.3, “MyISAM Table Storage Formats”](#).
- Declaring a MyISAM table with the `DELAY_KEY_WRITE=1` table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you must ensure that the table is okay by running the server with the `--myisam-recover-options [454]` option, or by running `myisamchk` before restarting the server. (However, even in this case, you should not lose anything by using `DELAY_KEY_WRITE`, because the key information can always be generated from the data rows.)
- Strings are automatically prefix- and end-space compressed in MyISAM indexes. See [Section 13.1.13, “CREATE INDEX Syntax”](#).
- You can increase performance by caching queries or answers in your application and then executing many inserts or updates together. Locking the table during this operation ensures that the index cache is only flushed once after all updates. You can also take advantage of MySQL's query cache to achieve similar results; see [Section 8.9.3, “The MySQL Query Cache”](#).

8.6.2. Bulk Data Loading for MyISAM Tables

These performance tips supplement the general guidelines for fast inserts in [Section 8.2.2.1, “Speed of INSERT Statements”](#).

- To improve performance when multiple clients insert a lot of rows, use the `INSERT DELAYED` statement. See [Section 13.2.5.2, “INSERT DELAYED Syntax”](#). This technique works for MyISAM and some other storage engines, but not InnoDB.
- For a MyISAM table, you can use concurrent inserts to add rows at the same time that `SELECT` statements are running, if there are no deleted rows in middle of the data file. See [Section 8.10.3, “Concurrent Inserts”](#).
- With some extra work, it is possible to make `LOAD DATA INFILE` run even faster for a MyISAM table when the table has many indexes. Use the following procedure:
 1. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.
 2. Use `myisamchk --keys-used=0 -rq /path/to/db/tbl_name` to remove all use of indexes for the table.
 3. Insert data into the table with `LOAD DATA INFILE`. This does not update any indexes and therefore is very fast.
 4. If you intend only to read from the table in the future, use `myisampack` to compress it. See [Section 14.5.3.3, “Compressed Table Characteristics”](#).
 5. Re-create the indexes with `myisamchk -rq /path/to/db/tbl_name`. This creates the index tree in memory before writing it to disk, which is much faster than updating the index during `LOAD DATA INFILE` because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.
 6. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.

`LOAD DATA INFILE` performs the preceding optimization automatically if the MyISAM table into which you insert data is empty. The main difference between automatic optimization and using the procedure explicitly is that you can let `myisamchk` allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the `LOAD DATA INFILE` statement.

You can also disable or enable the nonunique indexes for a `MyISAM` table by using the following statements rather than `myisamchk`. If you use these statements, you can skip the `FLUSH TABLE` operations:

```
ALTER TABLE tbl_name DISABLE KEYS;
ALTER TABLE tbl_name ENABLE KEYS;
```

- To speed up `INSERT` operations that are performed with multiple statements for nontransactional tables, lock your tables:

```
LOCK TABLES a WRITE;
INSERT INTO a VALUES (1,23),(2,34),(4,33);
INSERT INTO a VALUES (8,26),(6,29);
...
UNLOCK TABLES;
```

This benefits performance because the index buffer is flushed to disk only once, after all `INSERT` statements have completed. Normally, there would be as many index buffer flushes as there are `INSERT` statements. Explicit locking statements are not needed if you can insert all rows with a single `INSERT`.

Locking also lowers the total time for multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. Suppose that five clients attempt to perform inserts simultaneously as follows:

- Connection 1 does 1000 inserts
- Connections 2, 3, and 4 do 1 insert
- Connection 5 does 1000 inserts

If you do not use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

`INSERT`, `UPDATE`, and `DELETE` operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five successive inserts or updates. If you do very many successive inserts, you could do a `LOCK TABLES` followed by an `UNLOCK TABLES` once in a while (each 1,000 rows or so) to permit other threads to access table. This would still result in a nice performance gain.

`INSERT` is still much slower for loading data than `LOAD DATA INFILE`, even when using the strategies just outlined.

- To increase performance for `MyISAM` tables, for both `LOAD DATA INFILE` and `INSERT`, enlarge the key cache by increasing the `key_buffer_size` [508] system variable. See [Section 8.11.2, “Tuning Server Parameters”](#).

8.6.3. Speed of `REPAIR TABLE` Statements

`REPAIR TABLE` for `MyISAM` tables is similar to using `myisamchk` for repair operations, and some of the same performance optimizations apply:

- `myisamchk` has variables that control memory allocation. You may be able to its improve performance by setting these variables, as described in [Section 4.6.3.6, “myisamchk Memory Usage”](#).
- For `REPAIR TABLE`, the same principle applies, but because the repair is done by the server, you set server system variables instead of `myisamchk` variables. Also, in addition to setting memory-allocation variables, increasing the `myisam_max_sort_file_size` [528] system variable increases

the likelihood that the repair will use the faster filesort method and avoid the slower repair by key cache method. Set the variable to the maximum file size for your system, after checking to be sure that there is enough free space to hold a copy of the table files. The free space must be available in the file system containing the original table files.

Suppose that a `myisamchk` table-repair operation is done using the following options to set its memory-allocation variables:

```
--key_buffer_size=128M --myisam_sort_buffer_size=256M
--read_buffer_size=64M --write_buffer_size=64M
```

Some of those `myisamchk` variables correspond to server system variables:

<code>myisamchk</code> Variable	System Variable
<code>key_buffer_size</code>	<code>key_buffer_size</code> [508]
<code>myisam_sort_buffer_size</code>	<code>myisam_sort_buffer_size</code> [530]
<code>read_buffer_size</code>	<code>read_buffer_size</code> [544]
<code>write_buffer_size</code>	none

Each of the server system variables can be set at runtime, and some of them (`myisam_sort_buffer_size` [530], `read_buffer_size` [544]) have a session value in addition to a global value. Setting a session value limits the effect of the change to your current session and does not affect other users. Changing a global-only variable (`key_buffer_size` [508], `myisam_max_sort_file_size` [528]) affects other users as well. For `key_buffer_size` [508], you must take into account that the buffer is shared with those users. For example, if you set the `myisamchk key_buffer_size` variable to 128MB, you could set the corresponding `key_buffer_size` [508] system variable larger than that (if it is not already set larger), to permit key buffer use by activity in other sessions. However, changing the global key buffer size invalidates the buffer, causing increased disk I/O and slowdown for other sessions. An alternative that avoids this problem is to use a separate key cache, assign to it the indexes from the table to be repaired, and deallocate it when the repair is complete. See [Section 8.9.2.2, “Multiple Key Caches”](#).

Based on the preceding remarks, a `REPAIR TABLE` operation can be done as follows to use settings similar to the `myisamchk` command. Here a separate 128MB key buffer is allocated and the file system is assumed to permit a file size of at least 100GB.

```
SET SESSION myisam_sort_buffer_size = 256*1024*1024;
SET SESSION read_buffer_size = 64*1024*1024;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
SET GLOBAL repair_cache.key_buffer_size = 128*1024*1024;
CACHE INDEX tbl_name IN repair_cache;
LOAD INDEX INTO CACHE tbl_name;
REPAIR TABLE tbl_name ;
SET GLOBAL repair_cache.key_buffer_size = 0;
```

If you intend to change a global variable but want to do so only for the duration of a `REPAIR TABLE` operation to minimally affect other users, save its value in a user variable and restore it afterward. For example:

```
SET @old_myisam_sort_buffer_size = @@global.myisam_max_sort_file_size;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
REPAIR TABLE tbl_name ;
SET GLOBAL myisam_max_sort_file_size = @old_myisam_max_sort_file_size;
```

The system variables that affect `REPAIR TABLE` can be set globally at server startup if you want the values to be in effect by default. For example, add these lines to the server `my.cnf` file:

```
[mysqld]
```

```
mysam_sort_buffer_size=256M
key_buffer_size=1G
mysam_max_sort_file_size=100G
```

These settings do not include `read_buffer_size` [544]. Setting `read_buffer_size` [544] globally to a large value does so for all sessions and can cause performance to suffer due to excessive memory allocation for a server with many simultaneous sessions.

8.7. Optimizing for MEMORY Tables

Consider using MEMORY tables for noncritical data that is accessed often, and is read-only or rarely updated. Benchmark your application against equivalent InnoDB or MyISAM tables under a realistic workload, to confirm that any additional performance is worth the risk of losing data, or the overhead of copying data from a disk-based table at application start.

For best performance with MEMORY tables, examine the kinds of queries against each table, and specify the type to use for each associated index, either a B-tree index or a hash index. On the CREATE INDEX statement, use the clause USING BTREE or USING HASH. B-tree indexes are fast for queries that do greater-than or less-than comparisons through operators such as > or BETWEEN. Hash indexes are only fast for queries that look up single values through the = operator, or a restricted set of values through the IN operator. For why USING BTREE is often a better choice than the default USING HASH, see Section 8.2.1.4, “How to Avoid Full Table Scans”. For implementation details of the different types of MEMORY indexes, see Section 8.3.8, “Comparison of B-Tree and Hash Indexes”.

8.8. Understanding the Query Execution Plan

Depending on the details of your tables, columns, indexes, and the conditions in your WHERE clause, the MySQL optimizer considers many techniques to efficiently perform the lookups involved in an SQL query. A query on a huge table can be performed without reading all the rows; a join involving several tables can be performed without comparing every combination of rows. The set of operations that the optimizer chooses to perform the most efficient query is called the “query execution plan”, also known as the EXPLAIN plan. Your goals are to recognize the aspects of the EXPLAIN plan that indicate a query is optimized well, and to learn the SQL syntax and indexing techniques to improve the plan if you see some inefficient operations.

8.8.1. Optimizing Queries with EXPLAIN

The EXPLAIN statement can be used either as a way to obtain information about how MySQL executes a statement, or as a synonym for DESCRIBE:

- When you precede a SELECT statement with the keyword EXPLAIN, MySQL displays information from the optimizer about the query execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. See Section 8.8.2, “EXPLAIN Output Format”.
- EXPLAIN EXTENDED can be used to obtain additional information; see Section 8.8.3, “EXPLAIN EXTENDED Output Format”.
- EXPLAIN PARTITIONS is useful only when examining queries involving partitioned tables. For details, see Section 18.3.4, “Obtaining Information About Partitions”.
- EXPLAIN tbl_name is synonymous with DESCRIBE tbl_name or SHOW COLUMNS FROM tbl_name. For information about DESCRIBE and SHOW COLUMNS, see Section 13.8.1, “DESCRIBE Syntax”, and Section 13.7.5.6, “SHOW COLUMNS Syntax”.

With the help of EXPLAIN, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use EXPLAIN to check whether the optimizer

joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`. (See [Section 13.2.9](#), “`SELECT Syntax`”.)

If you have a problem with indexes not being used when you believe that they should be, run `ANALYZE TABLE` to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See [Section 13.7.2.1](#), “`ANALYZE TABLE Syntax`”.

8.8.2. EXPLAIN Output Format

The `EXPLAIN` statement provides information about the execution plan for a `SELECT` statement.

`EXPLAIN` returns a row of information for each table used in the `SELECT` statement. It lists the tables in the output in the order that MySQL would read them while processing the statement. MySQL resolves all joins using a nested-loop join method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

When the `EXTENDED` keyword is used, `EXPLAIN` produces extra information that can be viewed by issuing a `SHOW WARNINGS` statement following the `EXPLAIN` statement. `EXPLAIN EXTENDED` also displays the `filtered` column. See [Section 8.8.3](#), “`EXPLAIN EXTENDED Output Format`”.



Note

You cannot use the `EXTENDED` and `PARTITIONS` keywords together in the same `EXPLAIN` statement.

- [EXPLAIN Output Columns](#)
- [EXPLAIN Join Types](#)
- [EXPLAIN Extra Information](#)
- [EXPLAIN Output Interpretation](#)

EXPLAIN Output Columns

This section describes the output columns produced by `EXPLAIN`. Later sections provide additional information about the `type` and `Extra` columns.

Each output row from `EXPLAIN` provides information about one table. Each row contains the values summarized in [Table 8.1](#), “`EXPLAIN Output Columns`”, and described in more detail following the table.

Table 8.1. EXPLAIN Output Columns

Column	Meaning
<code>id</code> [835]	The <code>SELECT</code> identifier
<code>select_type</code> [835]	The <code>SELECT</code> type
<code>table</code> [835]	The table for the output row
<code>partitions</code> [836]	The matching partitions
<code>type</code> [836]	The join type
<code>possible_keys</code> [836]	The possible indexes to choose
<code>key</code> [836]	The index actually chosen
<code>key_len</code> [836]	The length of the chosen key

Column	Meaning
<code>ref</code> [836]	The columns compared to the index
<code>rows</code> [837]	Estimate of rows to be examined
<code>filtered</code> [837]	Percentage of rows filtered by table condition
<code>Extra</code> [837]	Additional information

- `id`

The `SELECT` identifier. This is the sequential number of the `SELECT` within the query. The value can be `NULL` if the row refers to the union result of other rows. In this case, the `table` column shows a value like `<unionM,N>` to indicate that the row refers to the union of the rows with `id` values of `M` and `N`.

- `select_type`

The type of `SELECT`, which can be any of those shown in the following table.

<code>select_type</code> Value	Meaning
<code>SIMPLE</code>	Simple <code>SELECT</code> (not using <code>UNION</code> or subqueries)
<code>PRIMARY</code>	Outermost <code>SELECT</code>
<code>UNION</code>	Second or later <code>SELECT</code> statement in a <code>UNION</code>
<code>DEPENDENT UNION</code>	Second or later <code>SELECT</code> statement in a <code>UNION</code> , dependent on outer query
<code>UNION RESULT</code>	Result of a <code>UNION</code> .
<code>SUBQUERY</code>	First <code>SELECT</code> in subquery
<code>DEPENDENT SUBQUERY</code>	First <code>SELECT</code> in subquery, dependent on outer query
<code>DERIVED</code>	Derived table <code>SELECT</code> (subquery in <code>FROM</code> clause)
<code>UNCACHEABLE SUBQUERY</code>	A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query
<code>UNCACHEABLE UNION</code>	The second or later select in a <code>UNION</code> that belongs to an uncacheable subquery (see <code>UNCACHEABLE SUBQUERY</code>)

`DEPENDENT` typically signifies the use of a correlated subquery. See [Section 13.2.10.7, “Correlated Subqueries”](#).

`DEPENDENT SUBQUERY` evaluation differs from `UNCACHEABLE SUBQUERY` evaluation. For `DEPENDENT SUBQUERY`, the subquery is re-evaluated only once for each set of different values of the variables from its outer context. For `UNCACHEABLE SUBQUERY`, the subquery is re-evaluated for each row of the outer context.

Cacheability of subqueries differs from caching of query results in the query cache (which is described in [Section 8.9.3.1, “How the Query Cache Operates”](#)). Subquery caching occurs during query execution, whereas the query cache is used to store results only after query execution finishes.

- `table`

The name of the table to which the row of output refers. This can also be one of the following values:

- `<unionM,N>`: The row refers to the union of the rows with `id` values of `M` and `N`.
- `<derivedN>`: The row refers to the derived table result for the row with an `id` value of `N`. A derived table may result, for example, from a subquery in the `FROM` clause.

- `partitions`

The `partitions` column indicates the partitions from which records would be matched by the query. This column is displayed only if the `PARTITIONS` keyword is used. The value is `NULL` for nonpartitioned tables. See [Section 18.3.4, “Obtaining Information About Partitions”](#).

- `type`

The join type. For descriptions of the different types, see [EXPLAIN Join Types](#).

- `possible_keys`

The `possible_keys` column indicates which indexes MySQL can choose from use to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from `EXPLAIN`. That means that some of the keys in `possible_keys` might not be usable in practice with the generated table order.

If this column is `NULL`, there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the `WHERE` clause to check whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with `EXPLAIN` again. See [Section 13.1.7, “ALTER TABLE Syntax”](#).

To see what indexes a table has, use `SHOW INDEX FROM tbl_name`.

- `key`

The `key` column indicates the key (index) that MySQL actually decided to use. If MySQL decides to use one of the `possible_keys` indexes to look up rows, that index is listed as the key value.

It is possible that `key` will name an index that is not present in the `possible_keys` value. This can happen if none of the `possible_keys` indexes are suitable for looking up rows, but all the columns selected by the query are columns of some other index. That is, the named index covers the selected columns, so although it is not used to determine which rows to retrieve, an index scan is more efficient than a data row scan.

For `InnoDB`, a secondary index might cover the selected columns even if the query also selects the primary key because `InnoDB` stores the primary key value with each secondary index. If `key` is `NULL`, MySQL found no index to use for executing the query more efficiently.

To force MySQL to use or ignore an index listed in the `possible_keys` column, use `FORCE INDEX`, `USE INDEX`, or `IGNORE INDEX` in your query. See [Section 13.2.9.3, “Index Hint Syntax”](#).

For `MyISAM` and `NDB` tables, running `ANALYZE TABLE` helps the optimizer choose better indexes. For `NDB` tables, this also improves performance of distributed pushed-down joins. For `MyISAM` tables, `myisamchk --analyze` does the same as `ANALYZE TABLE`. See [Section 7.6, “MyISAM Table Maintenance and Crash Recovery”](#).

- `key_len`

The `key_len` column indicates the length of the key that MySQL decided to use. The length is `NULL` if the `key` column says `NULL`. Note that the value of `key_len` enables you to determine how many parts of a multiple-part key MySQL actually uses.

- `ref`

The `ref` column shows which columns or constants are compared to the index named in the `key` column to select rows from the table.

- `rows`

The `rows` column indicates the number of rows MySQL believes it must examine to execute the query.

For `InnoDB` tables, this number is an estimate, and may not always be exact.

- `filtered`

The `filtered` column indicates an estimated percentage of table rows that will be filtered by the table condition. That is, `rows` shows the estimated number of rows examined and `rows × filtered / 100` shows the number of rows that will be joined with previous tables. This column is displayed if you use `EXPLAIN EXTENDED`.

- `Extra`

This column contains additional information about how MySQL resolves the query. For descriptions of the different values, see [EXPLAIN Extra Information](#).

EXPLAIN Join Types

The `type` column of `EXPLAIN` output describes how tables are joined. The following list describes the join types, ordered from the best type to the worst:

- `system` [837]

The table has only one row (= system table). This is a special case of the `const` [837] join type.

- `const` [837]

The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. `const` [837] tables are very fast because they are read only once.

`const` [837] is used when you compare all parts of a `PRIMARY KEY` or `UNIQUE` index to constant values. In the following queries, `tbl_name` can be used as a `const` [837] table:

```
SELECT * FROM tbl_name WHERE primary_key=1;

SELECT * FROM tbl_name
WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- `eq_ref` [837]

One row is read from this table for each combination of rows from the previous tables. Other than the `system` [837] and `const` [837] types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a `PRIMARY KEY` or `UNIQUE NOT NULL` index.

`eq_ref` [837] can be used for indexed columns that are compared using the `=` operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table. In the following examples, MySQL can use an `eq_ref` [837] join to process `ref_table`:

```
SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- `ref` [837]

All rows with matching index values are read from this table for each combination of rows from the previous tables. `ref [837]` is used if the join uses only a leftmost prefix of the key or if the key is not a `PRIMARY KEY` or `UNIQUE` index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

`ref [837]` can be used for indexed columns that are compared using the `=` or `<=>` operator. In the following examples, MySQL can use a `ref [837]` join to process `ref_table`:

```
SELECT * FROM ref_table WHERE key_column=expr;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

- `fulltext [838]`

The join is performed using a `FULLTEXT` index.

- `ref_or_null [838]`

This join type is like `ref [837]`, but with the addition that MySQL does an extra search for rows that contain `NULL` values. This join type optimization is used most often in resolving subqueries. In the following examples, MySQL can use a `ref_or_null [838]` join to process `ref_table`:

```
SELECT * FROM ref_table
  WHERE key_column=expr OR key_column IS NULL;
```

See [Section 8.13.4, “IS NULL Optimization”](#).

- `index_merge [838]`

This join type indicates that the Index Merge optimization is used. In this case, the `key` column in the output row contains a list of indexes used, and `key_len` contains a list of the longest key parts for the indexes used. For more information, see [Section 8.13.2, “Index Merge Optimization”](#).

- `unique_subquery [838]`

This type replaces `ref [837]` for some `IN` subqueries of the following form:

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

`unique_subquery [838]` is just an index lookup function that replaces the subquery completely for better efficiency.

- `index_subquery [838]`

This join type is similar to `unique_subquery [838]`. It replaces `IN` subqueries, but it works for nonunique indexes in subqueries of the following form:

```
value IN (SELECT key_column FROM single_table WHERE some_expr)
```

- `range [838]`

Only rows that are in a given range are retrieved, using an index to select the rows. The `key` column in the output row indicates which index is used. The `key_len` contains the longest key part that was used. The `ref` column is `NULL` for this type.

[range \[838\]](#) can be used when a key column is compared to a constant using any of the [= \[1089\]](#), [<> \[1090\]](#), [> \[1090\]](#), [>= \[1090\]](#), [< \[1090\]](#), [<= \[1090\]](#), [IS NULL \[1091\]](#), [<=> \[1090\]](#), [BETWEEN \[1091\]](#), or [IN\(\) \[1092\]](#) operators:

```
SELECT * FROM tbl_name
  WHERE key_column = 10;

SELECT * FROM tbl_name
  WHERE key_column BETWEEN 10 and 20;

SELECT * FROM tbl_name
  WHERE key_column IN (10,20,30);

SELECT * FROM tbl_name
  WHERE key_part1 = 10 AND key_part2 IN (10,20,30);
```

- [index \[839\]](#)

The [index](#) join type is the same as [ALL](#), except that the index tree is scanned. This occurs two ways:

- If the index is a covering index for the queries and can be used to satisfy all data required from the table, only the index tree is scanned. In this case, the [Extra](#) column says [Using index](#). An index-only scan usually is faster than [ALL](#) because the size of the index usually is smaller than the table data.
- A full table scan is performed using reads from the index to look up data rows in index order. [Uses index](#) does not appear in the [Extra](#) column.

MySQL can use this join type when the query uses only columns that are part of a single index.

- [ALL](#)

A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked [const \[837\]](#), and usually *very bad* in all other cases. Normally, you can avoid [ALL](#) by adding indexes that enable row retrieval from the table based on constant values or column values from earlier tables.

EXPLAIN Extra Information

The [Extra](#) column of [EXPLAIN](#) output contains additional information about how MySQL resolves the query. The following list explains the values that can appear in this column. If you want to make your queries as fast as possible, look out for [Extra](#) values of [Using filesort](#) and [Using temporary](#).

- [Child of 'table' pushed join@1](#)

This table is referenced as the child of *table* in a join that can be pushed down to the NDB kernel. Applies only in MySQL Cluster NDB 7.2 and later, when pushed-down joins are enabled. See the description of the [ndb_join_pushdown \[2151\]](#) server system variable for more information and examples.

- [const row not found](#)

For a query such as `SELECT ... FROM tbl_name`, the table was empty.

- [Distinct](#)

MySQL is looking for distinct values, so it stops searching for more rows for the current row combination after it has found the first matching row.

- Full scan on NULL key

This occurs for subquery optimization as a fallback strategy when the optimizer cannot use an index-lookup access method.

- Impossible HAVING

The `HAVING` clause is always false and cannot select any rows.

- Impossible WHERE

The `WHERE` clause is always false and cannot select any rows.

- Impossible WHERE noticed after reading const tables

MySQL has read all `const` [837] (and `system` [837]) tables and notice that the `WHERE` clause is always false.

- No matching min/max row

No row satisfies the condition for a query such as `SELECT MIN(...) FROM ... WHERE condition`.

- no matching row in const table

For a query with a join, there was an empty table or a table with no rows satisfying a unique index condition.

- No tables used

The query has no `FROM` clause, or has a `FROM DUAL` clause.

- Not exists

MySQL was able to do a `LEFT JOIN` optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the `LEFT JOIN` criteria. Here is an example of the type of query that can be optimized this way:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
WHERE t2.id IS NULL;
```

Assume that `t2.id` is defined as `NOT NULL`. In this case, MySQL scans `t1` and looks up the rows in `t2` using the values of `t1.id`. If MySQL finds a matching row in `t2`, it knows that `t2.id` can never be `NULL`, and does not scan through the rest of the rows in `t2` that have the same `id` value. In other words, for each row in `t1`, MySQL needs to do only a single lookup in `t2`, regardless of how many rows actually match in `t2`.

- Range checked for each record (index map: N)

MySQL found no good index to use, but found that some of indexes might be used after column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a `range` [838] or `index_merge` [838] access method to retrieve rows. This is not very fast, but is faster than performing a join with no index at all. The applicability criteria are as described in [Section 8.13.1, "Range Optimization"](#), and [Section 8.13.2, "Index Merge Optimization"](#), with the exception that all column values for the preceding table are known and considered to be constants.

Indexes are numbered beginning with 1, in the same order as shown by `SHOW INDEX` for the table. The index map value `N` is a bitmask value that indicates which indexes are candidates. For example, a value of `0x19` (binary 11001) means that indexes 1, 4, and 5 will be considered.

- `Scanned N databases`

This indicates how many directory scans the server performs when processing a query for `INFORMATION_SCHEMA` tables, as described in [Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”](#). The value of `N` can be 0, 1, or `all`.

- `Select tables optimized away`

The query contained only aggregate functions (`MIN()` [1206], `MAX()` [1205]) that were all resolved using an index, or `COUNT(*)` [1204] for `MyISAM`, and no `GROUP BY` clause. The optimizer determined that only one row should be returned.

- `Skip_open_table, Open_frm_only, Open_trigger_only, Open_full_table`

These values indicate file-opening optimizations that apply to queries for `INFORMATION_SCHEMA` tables, as described in [Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”](#).

- `Skip_open_table`: Table files do not need to be opened. The information has already become available within the query by scanning the database directory.
- `Open_frm_only`: Only the table's `.frm` file need be opened.
- `Open_trigger_only`: Only the table's `.TRG` file need be opened.
- `Open_full_table`: The unoptimized information lookup. The `.frm`, `.MYD`, and `.MYI` files must be opened.

- `unique row not found`

For a query such as `SELECT ... FROM tbl_name`, no rows satisfy the condition for a `UNIQUE` index or `PRIMARY KEY` on the table.

- `Using filesort`

MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the `WHERE` clause. The keys then are sorted and the rows are retrieved in sorted order. See [Section 8.13.9, “`ORDER BY` Optimization”](#).

- `Using index`

The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

If the `Extra` column also says `Using where`, it means the index is being used to perform lookups of key values. Without `Using where`, the optimizer may be reading the index to avoid reading data rows but not using it for lookups. For example, if the index is a covering index for the query, the optimizer may scan it without using it for lookups.

For `InnoDB` tables that have a user-defined clustered index, that index can be used even when `Using index` is absent from the `Extra` column. This is the case if `type` is `index` [839] and `key` is `PRIMARY`.

- `Using index for group-by`

Similar to the `Using index` table access method, `Using index for group-by` indicates that MySQL found an index that can be used to retrieve all columns of a `GROUP BY` or `DISTINCT` query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way

so that for each group, only a few index entries are read. For details, see [Section 8.13.10](#), “[GROUP BY Optimization](#)”.

- [Using join buffer](#)

Tables from earlier joins are read in portions into the join buffer, and then their rows are used from the buffer to perform the join with the current table.

- [Using sort_union\(...\)](#), [Using union\(...\)](#), [Using intersect\(...\)](#)

These indicate how index scans are merged for the [index_merge \[838\]](#) join type. See [Section 8.13.2](#), “[Index Merge Optimization](#)”.

- [Using temporary](#)

To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains [GROUP BY](#) and [ORDER BY](#) clauses that list columns differently.

- [Using where](#)

A [WHERE](#) clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the [Extra](#) value is not [Using where](#) and the table join type is [ALL](#) or [index \[839\]](#). Even if you are using an index for all parts of a [WHERE](#) clause, you may see [Using where](#) if the column can be [NULL](#).

- [Using where with pushed condition](#)

This item applies to [NDBCLUSTER](#) tables *only*. It means that MySQL Cluster is using the Condition Pushdown optimization to improve the efficiency of a direct comparison between a nonindexed column and a constant. In such cases, the condition is “pushed down” to the cluster's data nodes and is evaluated on all data nodes simultaneously. This eliminates the need to send nonmatching rows over the network, and can speed up such queries by a factor of 5 to 10 times over cases where Condition Pushdown could be but is not used. For more information, see [Section 8.13.3](#), “[Engine Condition Pushdown Optimization](#)”.

EXPLAIN Output Interpretation

You can get a good indication of how good a join is by taking the product of the values in the [rows](#) column of the [EXPLAIN](#) output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the [max_join_size \[522\]](#) system variable, this row product also is used to determine which multiple-table [SELECT](#) statements to execute and which to abort. See [Section 8.11.2](#), “[Tuning Server Parameters](#)”.

The following example shows how a multiple-table join can be optimized progressively based on the information provided by [EXPLAIN](#).

Suppose that you have the [SELECT](#) statement shown here and that you plan to examine it using [EXPLAIN](#):

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
             tt.ProjectReference, tt.EstimatedShipDate,
             tt.ActualShipDate, tt.ClientID,
             tt.ServiceCodes, tt.RepetitiveID,
             tt.CurrentProcess, tt.CurrentDPPerson,
             tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
             et_1.COUNTRY, do.CUSTNAME
FROM tt, et, et AS et_1, do
WHERE tt.SubmitTime IS NULL
      AND tt.ActualPC = et.EMPLOYID
      AND tt.AssignedPC = et_1.EMPLOYID
```

```
AND tt.ClientID = do.CUSTNMBR;
```

For this example, make the following assumptions:

- The columns being compared have been declared as follows.

Table	Column	Data Type
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- The tables have the following indexes.

Table	Index
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The `tt.ActualPC` values are not evenly distributed.

Initially, before any optimizations have been performed, the `EXPLAIN` statement produces the following information:

```
table type possible_keys key key_len ref rows Extra
et ALL PRIMARY NULL NULL NULL 74
do ALL PRIMARY NULL NULL NULL 2135
et_1 ALL PRIMARY NULL NULL NULL 74
tt ALL AssignedPC, NULL NULL NULL 3872
ClientID,
ActualPC
Range checked for each record (index map: 0x23)
```

Because `type` is `ALL` for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is $74 \times 2135 \times 74 \times 3872 = 45,268,558,720$ rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. `tt.ActualPC` is declared as `CHAR(10)` and `et.EMPLOYID` is `CHAR(15)`, so there is a length mismatch.

To fix this disparity between column lengths, use `ALTER TABLE` to lengthen `ActualPC` from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now `tt.ActualPC` and `et.EMPLOYID` are both `VARCHAR(15)`. Executing the `EXPLAIN` statement again produces this result:

```
table type possible_keys key key_len ref rows Extra
tt ALL AssignedPC, NULL NULL NULL 3872 Using
```

```

do      ALL      PRIMARY      NULL      NULL      NULL      2135
Range checked for each record (index map: 0x1)
et_1    ALL      PRIMARY      NULL      NULL      NULL      74
Range checked for each record (index map: 0x1)
et      eq_ref    PRIMARY      PRIMARY  15      tt.ActualPC  1

```

This is not perfect, but is much better: The product of the `rows` values is less by a factor of 74. This version executes in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the `tt.AssignedPC = et_1.EMPLOYID` and `tt.ClientID = do.CUSTNMBR` comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
->      MODIFY ClientID VARCHAR(15);
```

After that modification, `EXPLAIN` produces the output shown here:

```

table type  possible_keys  key      key_len  ref      rows  Extra
et     ALL     PRIMARY       NULL     NULL     NULL     74
tt     ref     AssignedPC,   ActualPC 15      et.EMPLOYID 52    Using
      ClientID,
      ActualPC
et_1   eq_ref  PRIMARY       PRIMARY  15      tt.AssignedPC 1
do     eq_ref  PRIMARY       PRIMARY  15      tt.ClientID  1

```

At this point, the query is optimized almost as well as possible. The remaining problem is that, by default, MySQL assumes that values in the `tt.ActualPC` column are evenly distributed, and that is not the case for the `tt` table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

With the additional index information, the join is perfect and `EXPLAIN` produces this result:

```

table type  possible_keys  key      key_len  ref      rows  Extra
tt     ALL     AssignedPC,   NULL     NULL     NULL     3872  Using
      ClientID,
      ActualPC
et     eq_ref  PRIMARY       PRIMARY  15      tt.ActualPC  1
et_1   eq_ref  PRIMARY       PRIMARY  15      tt.AssignedPC 1
do     eq_ref  PRIMARY       PRIMARY  15      tt.ClientID  1

```

Note that the `rows` column in the output from `EXPLAIN` is an educated guess from the MySQL join optimizer. Check whether the numbers are even close to the truth by comparing the `rows` product with the actual number of rows that the query returns. If the numbers are quite different, you might get better performance by using `STRAIGHT_JOIN` in your `SELECT` statement and trying to list the tables in a different order in the `FROM` clause.

It is possible in some cases to execute statements that modify data when `EXPLAIN SELECT` is used with a subquery; for more information, see [Section 13.2.10.8, "Subqueries in the FROM Clause"](#).

8.8.3. EXPLAIN EXTENDED Output Format

When `EXPLAIN` is used with the `EXTENDED` keyword, the output includes a `filtered` column not otherwise displayed. This column indicates the estimated percentage of table rows that will be filtered by the table condition. In addition, the statement produces extra information that can be viewed by issuing a `SHOW WARNINGS` statement following the `EXPLAIN` statement. The `Message` value in `SHOW WARNINGS` output displays how the optimizer qualifies table and column names in the `SELECT` statement, what the `SELECT` looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process. Here is an example:

```
mysql> EXPLAIN EXTENDED
```

```

-> SELECT t1.a, t1.a IN (SELECT t2.a FROM t2) FROM t1\G
***** 1. row *****
      id: 1
    select_type: PRIMARY
      table: t1
      type: index
possible_keys: NULL
      key: PRIMARY
     key_len: 4
       ref: NULL
      rows: 4
  filtered: 100.00
    Extra: Using index
***** 2. row *****
      id: 2
    select_type: DEPENDENT SUBQUERY
      table: t2
      type: index_subquery
possible_keys: a
      key: a
     key_len: 5
       ref: func
      rows: 2
  filtered: 100.00
    Extra: Using index
2 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
      Level: Note
      Code: 1003
Message: select `test`.`t1`.`a` AS `a`,
           <in_optimizer>(`test`.`t1`.`a`,
           <exists>( <index_lookup>( <cache>(`test`.`t1`.`a`)
           in t2 on a checking NULL having
           <is_not_null_test>(`test`.`t2`.`a`))) AS `t1.a`
           IN (SELECT t2.a FROM t2)` from `test`.`t1`
1 row in set (0.00 sec)

```

Because the statement displayed by `SHOW WARNINGS` may contain special markers to provide information about query rewriting or optimizer actions, the statement is not necessarily valid SQL and is not intended to be executed. The output may also include rows with `Message` values that provide additional non-SQL explanatory notes about actions taken by the optimizer.

The following list describes special markers that can appear in `EXTENDED` output displayed by `SHOW WARNINGS`:

- `<cache>(expr)`

The expression (such as a scalar subquery) is executed once and the resulting value is saved in memory for later use.

- `<exists>(query fragment)`

The subquery predicate is converted to an `EXISTS` predicate and the subquery is transformed so that it can be used together with the `EXISTS` predicate.

- `<in_optimizer>(query fragment)`

This is an internal optimizer object with no user significance.

- `<index_lookup>(query fragment)`

The query fragment is processed using an index lookup to find qualifying rows.

- `<is_not_null_test>(expr)`

A test to verify that the expression does not evaluate to `NULL`.

- `<primary_index_lookup>(query fragment)`

The query fragment is processed using a primary key lookup to find qualifying rows.

- `<ref_null_helper>(expr)`

This is an internal optimizer object with no user significance.

8.8.4. Estimating Query Performance

In most cases, you can estimate query performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: $\log(\text{row_count}) / \log(\text{index_block_length} / 3 * 2 / (\text{index_length} + \text{data_pointer_length})) + 1$.

In MySQL, an index block is usually 1,024 bytes and the data pointer is usually four bytes. For a 500,000-row table with a key value length of three bytes (the size of `MEDIUMINT`), the formula indicates $\log(500,000) / \log(1024/3*2/(3+4)) + 1 = 4$ seeks.

This index would require storage of about $500,000 * 7 * 3/2 = 5.2\text{MB}$ (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and so need only one or two calls to read data to find the row.

For writes, however, you need four seek requests to find where to place a new index value and normally two seeks to update the index and write the row.

Note that the preceding discussion does not mean that your application performance slowly degenerates by $\log N$. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are bound only by disk seeks (which increase by $\log N$). To avoid this, increase the key cache size as the data grows. For `MyISAM` tables, the key cache size is controlled by the `key_buffer_size` [508] system variable. See [Section 8.11.2, “Tuning Server Parameters”](#).

8.8.5. Controlling the Query Optimizer

MySQL provides optimizer control through system variables that affect how query plans are evaluated and which switchable optimizations are enabled.

8.8.5.1. Controlling Query Plan Evaluation

The task of the query optimizer is to find an optimal plan for executing an SQL query. Because the difference in performance between “good” and “bad” plans can be orders of magnitude (that is, seconds versus hours or even days), most query optimizers, including that of MySQL, perform a more or less exhaustive search for an optimal plan among all possible query evaluation plans. For join queries, the number of possible plans investigated by the MySQL optimizer grows exponentially with the number of tables referenced in a query. For small numbers of tables (typically less than 7 to 10) this is not a problem. However, when larger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server’s performance.

A more flexible method for query optimization enables the user to control how exhaustive the optimizer is in its search for an optimal query evaluation plan. The general idea is that the fewer plans that are investigated by the optimizer, the less time it spends in compiling a query. On the other hand, because the optimizer skips some plans, it may miss finding an optimal plan.

The behavior of the optimizer with respect to the number of plans it evaluates can be controlled using two system variables:

- The `optimizer_prune_level` [535] variable tells the optimizer to skip certain plans based on estimates of the number of rows accessed for each table. Our experience shows that this kind of “educated guess” rarely misses optimal plans, and may dramatically reduce query compilation times. That is why this option is on (`optimizer_prune_level=1`) by default. However, if you believe that the optimizer missed a better query plan, this option can be switched off (`optimizer_prune_level=0`) with the risk that query compilation may take much longer. Note that, even with the use of this heuristic, the optimizer still explores a roughly exponential number of plans.
- The `optimizer_search_depth` [536] variable tells how far into the “future” of each incomplete plan the optimizer should look to evaluate whether it should be expanded further. Smaller values of `optimizer_search_depth` [536] may result in orders of magnitude smaller query compilation times. For example, queries with 12, 13, or more tables may easily require hours and even days to compile if `optimizer_search_depth` [536] is close to the number of tables in the query. At the same time, if compiled with `optimizer_search_depth` [536] equal to 3 or 4, the optimizer may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for `optimizer_search_depth` [536], this variable can be set to 0 to tell the optimizer to determine the value automatically.

8.8.5.2. Controlling Switchable Optimizations

The `optimizer_switch` [536] system variable enables control over optimizer behavior. Its value is a set of flags, each of which has a value of `on` or `off` to indicate whether the corresponding optimizer behavior is enabled or disabled. This variable has global and session values and can be changed at runtime. The global default can be set at server startup.

To see the current set of optimizer flags, select the variable value:

```
mysql> SELECT @@optimizer_switch\G
***** 1. row *****
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on
```

To change the value of `optimizer_switch` [536], assign a value consisting of a comma-separated list of one or more commands:

```
SET [GLOBAL|SESSION] optimizer_switch='command[,command]...';
```

Each `command` value should have one of the forms shown in the following table.

Command Syntax	Meaning
<code>default</code>	Reset every optimization to its default value
<code>opt_name=default</code>	Set the named optimization to its default value
<code>opt_name=off</code>	Disable the named optimization
<code>opt_name=on</code>	Enable the named optimization

The order of the commands in the value does not matter, although the `default` command is executed first if present. Setting an `opt_name` flag to `default` sets it to whichever of `on` or `off` is its default value. Specifying any given `opt_name` more than once in the value is not permitted and causes an error. Any errors in the value cause the assignment to fail with an error, leaving the value of `optimizer_switch` [536] unchanged.

The following table lists the permissible `opt_name` flag names, grouped by optimization strategy.

Optimization	Flag Name	Meaning
Engine Condition Pushdown	<code>engine_condition_pushdown</code>	Controls engine condition pushdown
Index Merge	<code>index_merge</code>	Controls all Index Merge optimizations
	<code>index_merge_intersection</code>	Controls the Index Merge Intersection Access optimization
	<code>index_merge_sort_union</code>	Controls the Index Merge Sort-Union Access optimization
	<code>index_merge_union</code>	Controls the Index Merge Union Access optimization

The flag for engine condition pushdown was added in MySQL 5.5.3.

For more information about individual optimization strategies, see the following sections:

- [Section 8.13.3, “Engine Condition Pushdown Optimization”](#)
- [Section 8.13.2, “Index Merge Optimization”](#)

When you assign a value to `optimizer_switch` [536], flags that are not mentioned keep their current values. This makes it possible to enable or disable specific optimizer behaviors in a single statement without affecting other behaviors. The statement does not depend on what other optimizer flags exist and what their values are. Suppose that all Index Merge optimizations are enabled:

```
mysql> SELECT @@optimizer_switch\G
***** 1. row *****
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on
```

If the server is using the Index Merge Union or Index Merge Sort-Union access methods for certain queries and you want to check whether the optimizer will perform better without them, set the variable value like this:

```
mysql> SET optimizer_switch='index_merge_union=off,index_merge_sort_union=off';

mysql> SELECT @@optimizer_switch\G
***** 1. row *****
@@optimizer_switch: index_merge=on,index_merge_union=off,
                    index_merge_sort_union=off,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on
```

8.9. Buffering and Caching

MySQL uses several strategies that cache information in memory buffers to increase performance.

Depending on your database architecture, you balance the size and layout of these areas, to provide the most performance benefit without wasting memory or exceeding available memory. When you set up or resize these memory areas, test the resulting performance using the techniques from [Section 8.12, “Measuring Performance \(Benchmarking\)”](#).

8.9.1. The InnoDB Buffer Pool

InnoDB maintains a storage area called the [buffer pool](#) for caching data and indexes in memory. Knowing how the InnoDB buffer pool works, and taking advantage of it to keep frequently accessed data in memory, is an important aspect of MySQL tuning.

Guidelines

Ideally, you set the size of the buffer pool to as large a value as practical, leaving enough memory for other processes on the server to run without excessive paging. The larger the buffer pool, the more `InnoDB` acts like an in-memory database, reading data from disk once and then accessing the data from memory during subsequent reads. The buffer pool even caches data changed by insert and update operations, so that disk writes can be grouped together for better performance.

Depending on the typical workload on your system, you might adjust the proportions of the parts within the buffer pool. You can tune the way the buffer pool chooses which blocks to cache once it fills up, to keep frequently accessed data in memory despite sudden spikes of activity for operations such as backups or reporting.

With 64-bit systems with large memory sizes, you can split the buffer pool into multiple parts, to minimize contention for the memory structures among concurrent operations. For details, see [Section 14.4.7.18, “Improvements to Performance from Multiple Buffer Pools”](#).

Internal Details

`InnoDB` manages the pool as a list, using a variation of the least recently used (LRU) algorithm. When room is needed to add a new block to the pool, `InnoDB` evicts the least recently used block and adds the new block to the middle of the list. This “midpoint insertion strategy” treats the list as two sublists:

- At the head, a sublist of “new” (or “young”) blocks that were accessed recently.
- At the tail, a sublist of “old” blocks that were accessed less recently.

This algorithm keeps blocks that are heavily used by queries in the new sublist. The old sublist contains less-used blocks; these blocks are candidates for [eviction](#).

The LRU algorithm operates as follows by default:

- 3/8 of the buffer pool is devoted to the old sublist.
- The midpoint of the list is the boundary where the tail of the new sublist meets the head of the old sublist.
- When `InnoDB` reads a block into the buffer pool, it initially inserts it at the midpoint (the head of the old sublist). A block can be read in because it is required for a user-specified operation such as an SQL query, or as part of a [read-ahead](#) operation performed automatically by `InnoDB`.
- Accessing to a block in the old sublist makes it “young”, moving it to the head of the buffer pool (the head of the new sublist). If the block was read in because it was required, the first access occurs immediately and the block is made young. If the block was read in due to read-ahead, the first access does not occur immediately (and might not occur at all before the block is evicted).
- As the database operates, blocks in the buffer pool that are not accessed “age” by moving toward the tail of the list. Blocks in both the new and old sublists age as other blocks are made new. Blocks in the old sublist also age as blocks are inserted at the midpoint. Eventually, a block that remains unused for long enough reaches the tail of the old sublist and is evicted.

By default, blocks read by queries immediately move into the new sublist, meaning they will stay in the buffer pool for a long time. A table scan (such as performed for a `mysqldump` operation, or a `SELECT` statement with no `WHERE` clause) can bring a large amount of data into the buffer pool and evict an equivalent amount of older data, even if the new data is never used again. Similarly, blocks that are loaded by the read-ahead background thread and then accessed only once move to the head of the new list. These situations can push frequently used blocks to the old sublist, where they become subject to eviction.

Configuration Options

Several InnoDB system variables control the size of the buffer pool and let you tune the LRU algorithm:

- [innodb_buffer_pool_size \[1549\]](#)

Specifies the size of the buffer pool. If your buffer pool is small and you have sufficient memory, making the pool larger can improve performance by reducing the amount of disk I/O needed as queries access InnoDB tables.

- [innodb_buffer_pool_instances \[1549\]](#)

Divides the buffer pool into a user-specified number of separate regions, each with its own LRU list and related data structures, to reduce contention during concurrent memory read and write operations. This option takes effect only when you set the [innodb_buffer_pool_size](#) to a size of 1 gigabyte or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of [innodb_buffer_pool_instances \[1549\]](#) and [innodb_buffer_pool_size \[1549\]](#) so that each buffer pool instance is at least 1 gigabyte.

- [innodb_old_blocks_pct \[1566\]](#)

Specifies the approximate percentage of the buffer pool that InnoDB uses for the old block sublist. The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool).

- [innodb_old_blocks_time \[1566\]](#)

Specifies how long in milliseconds (ms) a block inserted into the old sublist must stay there after its first access before it can be moved to the new sublist. The default value is 0: A block inserted into the old sublist moves immediately to the new sublist the first time it is accessed, no matter how soon after insertion the access occurs. If the value is greater than 0, blocks remain in the old sublist until an access occurs at least that many ms after the first access. For example, a value of 1000 causes blocks to stay in the old sublist for 1 second after the first access before they become eligible to move to the new sublist.

Setting [innodb_old_blocks_time \[1566\]](#) greater than 0 prevents one-time table scans from flooding the new sublist with blocks used only for the scan. Rows in a block read in for a scan are accessed many times in rapid succession, but the block is unused after that. If [innodb_old_blocks_time \[1566\]](#) is set to a value greater than time to process the block, the block remains in the “old” sublist and ages to the tail of the list to be evicted quickly. This way, blocks used only for a one-time scan do not act to the detriment of heavily used blocks in the new sublist.

[innodb_old_blocks_time \[1566\]](#) can be set at runtime, so you can change it temporarily while performing operations such as table scans and dumps:

```
SET GLOBAL innodb_old_blocks_time = 1000; ... perform queries that scan tables ...
SET GLOBAL innodb_old_blocks_time = 0;
```

This strategy does not apply if your intent is to “warm up” the buffer pool by filling it with a table's content. For example, benchmark tests often perform a table or index scan at server startup, because that data would normally be in the buffer pool after a period of normal use. In this case, leave [innodb_old_blocks_time \[1566\]](#) set to 0, at least until the warmup phase is complete.

Monitoring the Buffer Pool

The output from the InnoDB Standard Monitor contains several fields in the [BUFFER POOL AND MEMORY](#) section that pertain to operation of the buffer pool LRU algorithm:

- [Old database pages](#): The number of pages in the old sublist of the buffer pool.

- **Pages made young, not young**: The number of old pages that were moved to the head of the buffer pool (the new sublist), and the number of pages that have remained in the old sublist without being made new.
- **young/s non-young/s**: The number of accesses to old pages that have resulted in making them young or not. This metric differs from that of the previous item in two ways. First, it relates only to old pages. Second, it is based on number of accesses to pages and not the number of pages. (There can be multiple accesses to a given page, all of which are counted.)
- **young-making rate**: Hits that cause blocks to move to the head of the buffer pool.
- **not**: Hits that do not cause blocks to move to the head of the buffer pool (due to the delay not being met).

The **young-making** rate and **not** rate will not normally add up to the overall buffer pool hit rate. Hits for blocks in the old sublist cause them to move to the new sublist, but hits to blocks in the new sublist cause them to move to the head of the list only if they are a certain distance from the head.

The preceding information from the Monitor can help you make LRU tuning decisions:

- If you see very low **young/s** values when you do not have large scans going on, that indicates that you might need to either reduce the delay time, or increase the percentage of the buffer pool used for the old sublist. Increasing the percentage makes the old sublist larger, so blocks in that sublist take longer to move to the tail and be evicted. This increases the likelihood that they will be accessed again and be made young.
- If you do not see a lot of **non-young/s** when you are doing large table scans (and lots of **young/s**), to tune your delay value to be larger.

For more information about InnoDB Monitors, see [Section 14.3.14.2, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”](#).

8.9.2. The **MyISAM** Key Cache

To minimize disk I/O, the **MyISAM** storage engine exploits a strategy that is used by many database management systems. It employs a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (or *key buffer*) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.
- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system file system cache.

This section first describes the basic operation of the **MyISAM** key cache. Then it discusses features that improve key cache performance and that enable you to better control cache operation:

- Multiple sessions can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

To control the size of the key cache, use the **key_buffer_size** [508] system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the **key_buffer_size** [508] value is too small to allocate the minimal number of block buffers (8).

When the key cache is not operational, index files are accessed using only the native file system buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the MyISAM index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are nonleaf nodes.)

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks) and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered “dirty.” In this case, prior to being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an *LRU (Least Recently Used)* strategy: When choosing a block for replacement, it selects the least recently used index block. To make this choice easier, the key cache module maintains all used blocks in a special list (*LRU chain*) ordered by time of use. When a block is accessed, it is the most recently used and is placed at the end of the list. When blocks need to be replaced, blocks at the beginning of the list are the least recently used and become the first candidates for eviction.

The InnoDB storage engine also uses an LRU algorithm, to manage its buffer pool. See [Section 8.9.1, “The InnoDB Buffer Pool”](#).

8.9.2.1. Shared Key Cache Access

Threads can access key cache buffers simultaneously, subject to the following conditions:

- A buffer that is not being updated can be accessed by multiple sessions.
- A buffer that is being updated causes sessions that need to use it to wait until the update is complete.
- Multiple sessions can initiate requests that result in cache block replacements, as long as they do not interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache enables the server to improve throughput significantly.

8.9.2.2. Multiple Key Caches

Shared access to the key cache improves performance but does not eliminate contention among sessions entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL also provides multiple key caches. This feature enables you to assign different table indexes to different key caches.

Where there are multiple key caches, the server must know which cache to use when processing queries for a given MyISAM table. By default, all MyISAM table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the `CACHE INDEX` statement (see [Section 13.7.6.2, “CACHE INDEX Syntax”](#)). For example, the following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+-----+-----+-----+-----+
| Table | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
```

```

| test.t1 | assign_to_keycache | status | OK |
| test.t2 | assign_to_keycache | status | OK |
| test.t3 | assign_to_keycache | status | OK |
+-----+-----+-----+-----+

```

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a `SET GLOBAL` parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

Note that you cannot destroy the default key cache. Any attempt to do this will be ignored:

```
mysql> SET GLOBAL key_buffer_size = 0;

mysql> SHOW VARIABLES LIKE 'key_buffer_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| key_buffer_size | 8384512 |
+-----+-----+
```

Key cache variables are structured system variables that have a name and components. For `keycache1.key_buffer_size`, `keycache1` is the cache variable name and `key_buffer_size` [508] is the cache component. See [Section 5.1.5.1, “Structured System Variables”](#), for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, you can use a strategy that involves three key caches:

- A “hot” key cache that takes up 20% of the space allocated for all key caches. Use this for tables that are heavily used for searches but that are not updated.
- A “cold” key cache that takes up 20% of the space allocated for all key caches. Use this cache for medium-sized, intensively modified tables, such as temporary tables.
- A “warm” key cache that takes up 60% of the key cache space. Employ this as the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Statements that access tables assigned to one cache do not compete with statements that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.
- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to nonleaf nodes of the index B-tree remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read in from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

The `CACHE INDEX` statement sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an `init-file` option that names a file containing `CACHE INDEX` statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysql_d_init.sql
```

The statements in `mysql_d_init.sql` are executed each time the server starts. The file should contain one SQL statement per line. The following example assigns several tables each to `hot_cache` and `cold_cache`:

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

8.9.2.3. Midpoint Insertion Strategy

By default, the key cache management system uses a simple LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the *midpoint insertion strategy*.

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sublist and a warm sublist. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not “too short,” always containing at least `key_cache_division_limit` [510] percent of the key cache blocks. `key_cache_division_limit` [510] is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sublist. After a certain number of hits (accesses of the block), it is promoted to the hot sublist. At present, the number of hits required to promote a block (3) is the same for all index blocks.

A block promoted into the hot sublist is placed at the end of the list. The block then circulates within this sublist. If the block stays at the beginning of the sublist for a long enough time, it is demoted to the warm sublist. This time is determined by the value of the `key_cache_age_threshold` [509] component of the key cache.

The threshold value prescribes that, for a key cache containing N blocks, the block at the beginning of the hot sublist not accessed within the last $N * \text{key_cache_age_threshold} / 100$ hits is to be moved to the beginning of the warm sublist. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sublist.

The midpoint insertion strategy enables you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the `key_cache_division_limit` [510] value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the `key_cache_division_limit` [510] set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sublist during an index scan operation as well.

8.9.2.4. Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its nonleaf nodes, it makes sense to preload the key cache with index blocks before

starting to use it. Preloading enables you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks will stay in the cache, because there are enough buffers for all of them, they are fetched from disk in random order, and not sequentially.

To preload an index into a cache, use the `LOAD INDEX INTO CACHE` statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
```

Table	Op	Msg_type	Msg_text
test.t1	preload_keys	status	OK
test.t2	preload_keys	status	OK

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from `t1`, but only blocks for the nonleaf nodes from `t2`.

If an index has been assigned to a key cache using a `CACHE INDEX` statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

8.9.2.5. Key Cache Block Size

It is possible to specify the size of the block buffers for an individual key cache using the `key_cache_block_size` [509] variable. This permits tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

To control the size of blocks in the `.MYI` index file of `MyISAM` tables, use the `--myisam-block-size` [454] option at server startup.

8.9.2.6. Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the `key_buffer_size` [508] or `key_cache_block_size` [509] key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you change other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native file system caching. File system caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. After the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of file system caching for the indexes ceases.

8.9.3. The MySQL Query Cache

The query cache stores the text of a `SELECT` statement together with the corresponding result that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again. The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client.

The query cache can be useful in an environment where you have tables that do not change very often and for which the server receives many identical queries. This is a typical situation for many Web servers that generate many dynamic pages based on database content. For example, when an order form queries a table to display the lists of all US states or all countries in the world, those values can be retrieved from the query cache. Although the values would probably be retrieved from memory in any case (from the `InnoDB` buffer pool or `MyISAM` key cache), using the query cache avoids the overhead of processing the query, deciding whether to use a table scan, and locating the data block for each row.

The query cache always contains current and reliable data. Any insert, update, delete, or other modification to a table causes any relevant entries in the query cache to be flushed.



Note

The query cache does not work in an environment where you have multiple `mysqld` servers updating the same `MyISAM` tables.

The query cache is used for prepared statements under the conditions described in [Section 8.9.3.1, “How the Query Cache Operates”](#).



Note

As of MySQL 5.5.23, the query cache is not supported for partitioned tables, and is automatically disabled for queries involving partitioned tables. The query cache cannot be enabled for such queries. (Bug #53775)

Some performance data for the query cache follows. These results were generated by running the MySQL benchmark suite on a Linux Alpha 2x500MHz system with 2GB RAM and a 64MB query cache.

- If all the queries you are performing are simple (such as selecting a row from a table with one row), but still differ so that the queries cannot be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. In real life, queries tend to be much more complicated, so the overhead normally is significantly lower.
- Searches for a single row in a single-row table are 238% faster with the query cache than without it. This can be regarded as close to the minimum speedup to be expected for a query that is cached.

To disable the query cache at server startup, set the `query_cache_size` [541] system variable to 0. By disabling the query cache code, there is no noticeable overhead.

The query cache offers the potential for substantial performance improvement, but do not assume that it will do so under all circumstances. With some query cache configurations or server workloads, you might actually see a performance decrease:

- Be cautious about sizing the query cache excessively large, which increases the overhead required to maintain the cache, possibly beyond the benefit of enabling it. Sizes in tens of megabytes are usually beneficial. Sizes in the hundreds of megabytes might not be.
- Server workload has a significant effect on query cache efficiency. A query mix consisting almost entirely of a fixed set of `SELECT` statements is much more likely to benefit from enabling the cache than a mix in which frequent `INSERT` statements cause continual invalidation of results in the cache. In some cases, a workaround is to use the `SQL_NO_CACHE` option to prevent results from even entering the cache for `SELECT` statements that use frequently modified tables. (See [Section 8.9.3.2, “Query Cache `SELECT` Options”](#).)

To verify that enabling the query cache is beneficial, test the operation of your MySQL server with the cache enabled and disabled. Then retest periodically because query cache efficiency may change as server workload changes.

8.9.3.1. How the Query Cache Operates

This section describes how the query cache works when it is operational. [Section 8.9.3.3, “Query Cache Configuration”](#), describes how to control whether it is operational.

Incoming queries are compared to those in the query cache before parsing, so the following two queries are regarded as different by the query cache:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Queries must be *exactly* the same (byte for byte) to be seen as identical. In addition, query strings that are identical may be treated as different for other reasons. Queries that use different databases, different protocol versions, or different default character sets are considered different queries and are cached separately.

The cache is not used for queries of the following types:

- Queries that are a subquery of an outer query
- Queries executed within the body of a stored function, trigger, or event

Before a query result is fetched from the query cache, MySQL checks whether the user has `SELECT` privilege for all databases and tables involved. If this is not the case, the cached result is not used.

If a query result is returned from query cache, the server increments the `Qcache_hits` [605] status variable, not `Com_select`. See [Section 8.9.3.4, “Query Cache Status and Maintenance”](#).

If a table changes, all cached queries that use the table become invalid and are removed from the cache. This includes queries that use `MERGE` tables that map to the changed table. A table can be changed by many types of statements, such as `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE TABLE`, `ALTER TABLE`, `DROP TABLE`, or `DROP DATABASE`.

The query cache also works within transactions when using `InnoDB` tables.

In MySQL 5.5, the result from a `SELECT` query on a view is cached.

The query cache works for `SELECT SQL_CALC_FOUND_ROWS ...` queries and stores a value that is returned by a following `SELECT FOUND_ROWS()` query. `FOUND_ROWS()` [1193] returns the correct value even if the preceding query was fetched from the cache because the number of found rows is also stored in the cache. The `SELECT FOUND_ROWS()` query itself cannot be cached.

Prepared statements that are issued using the binary protocol using `mysql_stmt_prepare()` and `mysql_stmt_execute()` (see [Section 22.8.4, “C API Prepared Statements”](#)), are subject to limitations on caching. Comparison with statements in the query cache is based on the text of the statement after expansion of `?` parameter markers. The statement is compared only with other cached statements that were executed using the binary protocol. That is, for query cache purposes, prepared statements issued using the binary protocol are distinct from prepared statements issued using the text protocol (see [Section 13.5, “SQL Syntax for Prepared Statements”](#)).

A query cannot be cached if it contains any of the functions shown in the following table.

<code>BENCHMARK()</code> [1190]	<code>CONNECTION_ID()</code> [1191]	<code>CONVERT_TZ()</code> [1134]
<code>CURDATE()</code> [1135]	<code>CURRENT_DATE()</code> [1135]	<code>CURRENT_TIME()</code> [1135]

CURRENT_TIMESTAMP() [1135]	CURTIME() [1135]	DATABASE() [1193]
ENCRYPT() [1187] with one parameter	FOUND_ROWS() [1193]	GET_LOCK() [1199]
LAST_INSERT_ID() [1194]	LOAD_FILE() [1105]	MASTER_POS_WAIT() [1200]
NOW() [1143]	RAND() [1128]	RELEASE_LOCK() [1201]
SLEEP() [1201]	SYSDATE() [1146]	UNIX_TIMESTAMP() [1150] with no parameters
USER() [1197]	UUID() [1201]	UUID_SHORT() [1202]

A query also is not cached under these conditions:

- It refers to user-defined functions (UDFs) or stored functions.
- It refers to user variables or local stored program variables.
- It refers to tables in the `mysql`, `INFORMATION_SCHEMA`, or `performance_schema` database.
- (*MySQL 5.5.23 and later:*) It refers to any partitioned tables.
- It is of any of the following forms:

```
SELECT ... LOCK IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value. See the Connector/ODBC section of [Chapter 22, Connectors and APIs](#).

Statements within transactions that use [SERIALIZABLE \[1402\]](#) isolation level also cannot be cached because they use `LOCK IN SHARE MODE` locking.

- It uses `TEMPORARY` tables.
- It does not use any tables.
- It generates warnings.
- The user has a column-level privilege for any of the involved tables.

8.9.3.2. Query Cache `SELECT` Options

Two query cache-related options may be specified in `SELECT` statements:

- `SQL_CACHE`

The query result is cached if it is cacheable and the value of the `query_cache_type [542]` system variable is `ON` or `DEMAND`.

- `SQL_NO_CACHE`

The query result is not cached.

Examples:

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

8.9.3.3. Query Cache Configuration

The `have_query_cache` [505] server system variable indicates whether the query cache is available:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES  |
+-----+-----+
```

When using a standard MySQL binary, this value is always `YES`, even if query caching is disabled.

Several other system variables control query cache operation. These can be set in an option file or on the command line when starting `mysqld`. The query cache system variables all have names that begin with `query_cache_`. They are described briefly in [Section 5.1.4, “Server System Variables”](#), with additional configuration information given here.

To set the size of the query cache, set the `query_cache_size` [541] system variable. Setting it to 0 disables the query cache. The default size is 0, so the query cache is disabled by default. To reduce overhead significantly, also start the server with `query_cache_type=0` [542] if you will not be using the query cache.



Note

When using the Windows Configuration Wizard to install or configure MySQL, the default value for `query_cache_size` [541] will be configured automatically for you based on the different configuration types available. When using the Windows Configuration Wizard, the query cache may be enabled (that is, set to a nonzero value) due to the selected configuration. The query cache is also controlled by the setting of the `query_cache_type` [542] variable. Check the values of these variables as set in your `my.ini` file after configuration has taken place.

When you set `query_cache_size` [541] to a nonzero value, keep in mind that the query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value too small, you'll get a warning, as in this example:

```
mysql> SET GLOBAL query_cache_size = 40000;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1282
Message: Query cache failed to set size 39936;
        new query cache size is 0

mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_size | 41984 |
+-----+-----+
```

For the query cache to actually be able to hold any query results, its size must be set larger:

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SHOW VARIABLES LIKE 'query_cache_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_size | 999424 |
+-----+-----+
1 row in set (0.00 sec)
```

The `query_cache_size` [541] value is aligned to the nearest 1024 byte block. The value reported may therefore be different from the value that you assign.

If the query cache size is greater than 0, the `query_cache_type` [542] variable influences how it works. This variable can be set to the following values:

- A value of `0` or `OFF` prevents caching or retrieval of cached results.
- A value of `1` or `ON` enables caching except of those statements that begin with `SELECT SQL_NO_CACHE`.
- A value of `2` or `DEMAND` causes caching of only those statements that begin with `SELECT SQL_CACHE`.

If `query_cache_size` [541] is 0, you should also set `query_cache_type` [542] variable to 0. In this case, the server does not acquire the query cache mutex at all, which means that the query cache cannot be enabled at runtime and there is reduced overhead in query execution.

Setting the GLOBAL `query_cache_type` [542] value determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the SESSION `query_cache_type` [542] value. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

If you set `query_cache_type` [542] at server startup (rather than at runtime with a `SET` statement), only the numeric values are permitted.

To control the maximum size of individual query results that can be cached, set the `query_cache_limit` [540] system variable. The default value is 1MB.

Be careful not to set the size of the cache too large. Due to the need for threads to lock the cache during updates, you may see lock contention issues with a very large cache.



Note

You can set the maximum size that can be specified for the query cache at run time with the `SET` statement by using the `--maximum-query-cache-size=32M` option on the command line or in the configuration file.

When a query is to be cached, its result (the data sent to the client) is stored in the query cache during result retrieval. Therefore the data usually is not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (timewise), the query cache allocates blocks with a minimum size given by the `query_cache_min_res_unit` [541] system variable. When a query is executed, the last result block is trimmed to the actual data size so that unused memory is freed. Depending on the types of queries your server executes, you might find it helpful to tune the value of `query_cache_min_res_unit` [541]:

- The default value of `query_cache_min_res_unit` [541] is 4KB. This should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation, as indicated by a large number of free blocks. Fragmentation can force the query cache to

prune (delete) queries from the cache due to lack of memory. In this case, decrease the value of `query_cache_min_res_unit` [541]. The number of free blocks and queries removed due to pruning are given by the values of the `Qcache_free_blocks` [604] and `Qcache_lowmem_prunes` [605] status variables.

- If most of your queries have large results (check the `Qcache_total_blocks` [605] and `Qcache_queries_in_cache` [605] status variables), you can increase performance by increasing `query_cache_min_res_unit` [541]. However, be careful to not make it too large (see the previous item).

8.9.3.4. Query Cache Status and Maintenance

To check whether the query cache is present in your MySQL server, use the following statement:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES |
+-----+-----+
```

You can defragment the query cache to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The `RESET QUERY CACHE` statement removes all query results from the query cache. The `FLUSH TABLES` statement also does this.

To monitor query cache performance, use `SHOW STATUS` to view the cache status variables:

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_free_blocks | 36 |
| Qcache_free_memory | 138488 |
| Qcache_hits | 79570 |
| Qcache_inserts | 27087 |
| Qcache_lowmem_prunes | 3114 |
| Qcache_not_cached | 22989 |
| Qcache_queries_in_cache | 415 |
| Qcache_total_blocks | 912 |
+-----+-----+
```

Descriptions of each of these variables are given in [Section 5.1.6, “Server Status Variables”](#). Some uses for them are described here.

The total number of `SELECT` queries is given by this formula:

```
Com_select
+ Qcache_hits
+ queries with errors found by parser
```

The `Com_select` value is given by this formula:

```
Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during the column-privileges check
```

The query cache uses variable-length blocks, so `Qcache_total_blocks` [605] and `Qcache_free_blocks` [604] may indicate query cache memory fragmentation. After `FLUSH QUERY CACHE`, only a single free block remains.

Every cached query requires a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query requires one block. However, if two or more queries use the same table, only one table block needs to be allocated.

The information provided by the `Qcache_lowmem_prunes` [605] status variable can help you tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a least recently used (LRU) strategy to decide which queries to remove from the cache. Tuning information is given in [Section 8.9.3.3, “Query Cache Configuration”](#).

8.10. Optimizing Locking Operations

When your database is busy with multiple sessions reading and writing data, the mechanism that controls access to data files and memory areas can become a consideration for performance tuning. Otherwise, sessions can spend time waiting for access to resources when they could be running concurrently.

MySQL manages contention for table contents using locking:

- Internal locking is performed within the MySQL server itself to manage contention for table contents by multiple threads. This type of locking is internal because it is performed entirely by the server and involves no other programs. See [Section 8.10.1, “Internal Locking Methods”](#).
- External locking occurs when the server and other programs lock `MyISAM` table files to coordinate among themselves which program can access the tables at which time. See [Section 8.10.5, “External Locking”](#).

8.10.1. Internal Locking Methods

This section discusses internal locking; that is, locking performed within the MySQL server to manage contention for table contents by multiple sessions.

MySQL uses [row-level locking](#) for `InnoDB` tables to support simultaneous write access by multiple sessions, making them suitable for multi-user, highly concurrent, and OLTP applications. MySQL uses [table-level locking](#) for `MyISAM`, `MEMORY`, and `MERGE` tables, allowing only one session to update those tables at a time, making them more suitable for read-only, read-mostly, or single-user applications.

Considerations for Row Locking

Advantages of row-level locking:

- Fewer lock conflicts when different sessions access different rows.
- Fewer changes for rollbacks.
- Possible to lock a single row for a long time.

Disadvantages of row-level locking:

- Requires more memory than table-level locks.
- Slower than table-level locks when used on a large part of the table because you must acquire many more locks.
- Slower than other locks if you often do `GROUP BY` operations on a large part of the data or if you must scan the entire table frequently.

Considerations for Table Locking

Table locking in MySQL is deadlock-free for storage engines that use table-level locking. Deadlock avoidance is managed by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order.

MySQL grants table write locks as follows:

1. If there are no locks on the table, put a write lock on it.
2. Otherwise, put the lock request in the write lock queue.

MySQL grants table read locks as follows:

1. If there are no write locks on the table, put a read lock on it.
2. Otherwise, put the lock request in the read lock queue.

Table updates are given higher priority than table retrievals. Therefore, when a lock is released, the lock is made available to the requests in the write lock queue and then to the requests in the read lock queue. This ensures that updates to a table are not “starved” even if there is heavy `SELECT` activity for the table. However, if you have many updates for a table, `SELECT` statements wait until there are no more updates.

For information on altering the priority of reads and writes, see [Section 8.10.2, “Table Locking Issues”](#).

You can analyze the table lock contention on your system by checking the `Table_locks_immediate` [610] and `Table_locks_waited` [610] status variables, which indicate the number of times that requests for table locks could be granted immediately and the number that had to wait, respectively:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 1151552 |
| Table_locks_waited | 15324 |
+-----+-----+
```

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no free blocks in the middle of the data file, rows are always inserted at the end of the data file. In this case, you can freely mix concurrent `INSERT` and `SELECT` statements for a `MyISAM` table without locks. That is, you can insert rows into a `MyISAM` table at the same time other clients are reading from it. Holes can result from rows having been deleted from or updated in the middle of the table. If there are holes, concurrent inserts are disabled but are enabled again automatically when all holes have been filled with new data.. This behavior is altered by the `concurrent_insert` [490] system variable. See [Section 8.10.3, “Concurrent Inserts”](#).

If you acquire a table lock explicitly with `LOCK TABLES`, you can request a `READ LOCAL` lock rather than a `READ` lock to enable other sessions to perform concurrent inserts while you have the table locked.

To perform many `INSERT` and `SELECT` operations on a table `real_table` when concurrent inserts are not possible, you can insert rows into a temporary table `temp_table` and update the real table with the rows from the temporary table periodically. This can be done with the following code:

```
mysql> LOCK TABLES real_table WRITE, temp_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM temp_table;
mysql> DELETE FROM temp_table;
mysql> UNLOCK TABLES;
```

[InnoDB](#) uses row locks. Deadlocks are possible for [InnoDB](#) because it automatically acquires locks during the processing of SQL statements, not at the start of the transaction.

Choosing the Type of Locking

Generally, table locks are superior to row-level locks in the following cases:

- Most statements for the table are reads.
- Statements for the table are a mix of reads and writes, where writes are updates or deletes for a single row that can be fetched with one key read:

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- [SELECT](#) combined with concurrent [INSERT](#) statements, and very few [UPDATE](#) or [DELETE](#) statements.
- Many scans or [GROUP BY](#) operations on the entire table without any writers.

With higher-level locks, you can more easily tune applications by supporting locks of different types, because the lock overhead is less than for row-level locks.

Options other than row-level locking:

- Versioning (such as that used in MySQL for concurrent inserts) where it is possible to have one writer at the same time as many readers. This means that the database or table supports different views for the data depending on when access begins. Other common terms for this are “time travel,” “copy on write,” or “copy on demand.”
- Copy on demand is in many cases superior to row-level locking. However, in the worst case, it can use much more memory than using normal locks.
- Instead of using row-level locks, you can employ application-level locks, such as those provided by [GET_LOCK\(\)](#) [1199] and [RELEASE_LOCK\(\)](#) [1201] in MySQL. These are advisory locks, so they work only with applications that cooperate with each other. See [Section 12.15](#), “Miscellaneous Functions”.

8.10.2. Table Locking Issues

[InnoDB](#) tables use row-level locking so that multiple sessions and applications can read from and write to the same table simultaneously, without making each other wait or producing inconsistent results. For this storage engine, avoid using the [LOCK TABLES](#) statement, because it does not offer any extra protection, but instead reduces concurrency. The automatic row-level locking makes these tables suitable for your busiest databases with your most important data, while also simplifying application logic since you do not need to lock and unlock tables. Consequently, the [InnoDB](#) storage engine is the default in MySQL 5.5 and higher.

MySQL uses table locking (instead of page, row, or column locking) for all storage engines except [InnoDB](#) and [NDBCLUSTER](#). The locking operations themselves do not have much overhead. But because only one session can write to a table at any one time, for best performance with these other storage engines, use them primarily for tables that are queried often and rarely inserted into or updated.

Performance Considerations Favoring [InnoDB](#)

When choosing whether to create a table using [InnoDB](#) or a different storage engine, keep in mind the following disadvantages of table locking:

- Table locking enables many sessions to read from a table at the same time, but if a session wants to write to a table, it must first get exclusive access, meaning it might have to wait for other sessions to

finish with the table first. During the update, all other sessions that want to access this particular table must wait until the update is done.

- Table locking causes problems when a session is waiting because the disk is full and free space needs to become available before the session can proceed. In this case, all sessions that want to access the problem table are also put in a waiting state until more disk space is made available.
- A `SELECT` statement that takes a long time to run prevents other sessions from updating the table in the meantime, making the other sessions appear slow or unresponsive. While a session is waiting to get exclusive access to the table for updates, other sessions that issue `SELECT` statements will queue up behind it, reducing concurrency even for read-only sessions.

Workarounds for Locking Performance Issues

The following items describe some ways to avoid or reduce contention caused by table locking:

- Consider switching the table to the `InnoDB` storage engine, either using `CREATE TABLE ... ENGINE=INNODB` during setup, or using `ALTER TABLE ... ENGINE=INNODB` for an existing table. See [Section 14.3, “The InnoDB Storage Engine”](#) for more details about this storage engine.
- Optimize `SELECT` statements to run faster so that they lock tables for a shorter time. You might have to create some summary tables to do this.
- Start `mysqld` with `--low-priority-updates` [452]. For storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`), this gives all statements that update (modify) a table lower priority than `SELECT` statements. In this case, the second `SELECT` statement in the preceding scenario would execute before the `UPDATE` statement, and would not wait for the first `SELECT` to finish.
- To specify that all updates issued in a specific connection should be done with low priority, set the `low_priority_updates` [517] server system variable equal to 1.
- To give a specific `INSERT`, `UPDATE`, or `DELETE` statement lower priority, use the `LOW_PRIORITY` attribute.
- To give a specific `SELECT` statement higher priority, use the `HIGH_PRIORITY` attribute. See [Section 13.2.9, “SELECT Syntax”](#).
- Start `mysqld` with a low value for the `max_write_lock_count` [526] system variable to force MySQL to temporarily elevate the priority of all `SELECT` statements that are waiting for a table after a specific number of inserts to the table occur. This permits `READ` locks after a certain number of `WRITE` locks.
- If you have problems with `INSERT` combined with `SELECT`, consider switching to `MyISAM` tables, which support concurrent `SELECT` and `INSERT` statements. (See [Section 8.10.3, “Concurrent Inserts”](#).)
- If you mix inserts and deletes on the same table, `INSERT DELAYED` may be of great help. See [Section 13.2.5.2, “INSERT DELAYED Syntax”](#).
- If you have problems with mixed `SELECT` and `DELETE` statements, the `LIMIT` option to `DELETE` may help. See [Section 13.2.2, “DELETE Syntax”](#).
- Using `SQL_BUFFER_RESULT` with `SELECT` statements can help to make the duration of table locks shorter. See [Section 13.2.9, “SELECT Syntax”](#).
- Splitting table contents into separate tables may help, by allowing queries to run against columns in one table, while updates are confined to columns in a different table.
- You could change the locking code in `mysys/thr_lock.c` to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

8.10.3. Concurrent Inserts

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no holes in the data file (deleted rows in the middle), an `INSERT` statement can be executed to add rows to the end of the table at the same time that `SELECT` statements are reading rows from the table. If there are multiple `INSERT` statements, they are queued and performed in sequence, concurrently with the `SELECT` statements. The results of a concurrent `INSERT` may not be visible immediately.

The `concurrent_insert` [490] system variable can be set to modify the concurrent-insert processing. By default, the variable is set to `AUTO` (or 1) and concurrent inserts are handled as just described. If `concurrent_insert` [490] is set to `NEVER` (or 0), concurrent inserts are disabled. If the variable is set to `ALWAYS` (or 2), concurrent inserts at the end of the table are permitted even for tables that have deleted rows. See also the description of the `concurrent_insert` [490] system variable.

Under circumstances where concurrent inserts can be used, there is seldom any need to use the `DELAYED` modifier for `INSERT` statements. See [Section 13.2.5.2, “INSERT DELAYED Syntax”](#).

If you are using the binary log, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. See [Section 5.2.4, “The Binary Log”](#). In addition, for those statements a read lock is placed on the selected-from table such that inserts into that table are blocked. The effect is that concurrent inserts for that table must wait as well.

With `LOAD DATA INFILE`, if you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other sessions can retrieve data from the table while `LOAD DATA` is executing. Use of the `CONCURRENT` option affects the performance of `LOAD DATA` a bit, even if no other session is using the table at the same time.

If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` [452] option if the server was started with that option. It also causes concurrent inserts not to be used.

For `LOCK TABLE`, the difference between `READ LOCAL` and `READ` is that `READ LOCAL` permits nonconflicting `INSERT` statements (concurrent inserts) to execute while the lock is held. However, this cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock.

8.10.4. Metadata Locking Within Transactions

To ensure transaction serializability, the server must not permit one session to perform a data definition language (DDL) statement on a table that is used in an uncompleted transaction in another session.

As of MySQL 5.5.3, the server achieves this by acquiring metadata locks on tables used within a transaction and deferring release of those locks until the transaction ends. A metadata lock on a table prevents changes to the table's structure. This locking approach has the implication that a table that is being used by a transaction within one session cannot be used in DDL statements by other sessions until the transaction ends. For example, if a table `t1` is in use by a transaction, another session that attempts to execute `DROP TABLE t1` blocks until the transaction ends.

If the server acquires metadata locks for a statement that is syntactically valid but fails during execution, it does not release the locks early. Lock release is still deferred to the end of the transaction because the failed statement is written to the binary log and the locks protect log consistency.

Metadata locks acquired during a `PREPARE` statement are released once the statement has been prepared, even if preparation occurs within a multiple-statement transaction.

Before MySQL 5.5.3, when a transaction acquired a metadata lock for a table used within a statement, it released the lock at the end of the statement. This approach had the disadvantage that if a DDL statement occurred for a table that was being used by another session in an active transaction, statements could be written to the binary log in the wrong order.

8.10.5. External Locking

External locking is the use of file system locking to manage contention for [MyISAM](#) database tables by multiple processes. External locking is used in situations where a single process such as the MySQL server cannot be assumed to be the only process that requires access to tables. Here are some examples:

- If you run multiple servers that use the same database directory (not recommended), each server must have external locking enabled.
- If you use [myisamchk](#) to perform table maintenance operations on [MyISAM](#) tables, you must either ensure that the server is not running, or that the server has external locking enabled so that it locks table files as necessary to coordinate with [myisamchk](#) for access to the tables. The same is true for use of [myisampack](#) to pack [MyISAM](#) tables.

If the server is run with external locking enabled, you can use [myisamchk](#) at any time for read operations such as checking tables. In this case, if the server tries to update a table that [myisamchk](#) is using, the server will wait for [myisamchk](#) to finish before it continues.

If you use [myisamchk](#) for write operations such as repairing or optimizing tables, or if you use [myisampack](#) to pack tables, you *must* always ensure that the [mysqld](#) server is not using the table. If you don't stop [mysqld](#), at least do a [mysqladmin flush-tables](#) before you run [myisamchk](#). Your tables *may become corrupted* if the server and [myisamchk](#) access the tables simultaneously.

With external locking in effect, each process that requires access to a table acquires a file system lock for the table files before proceeding to access the table. If all necessary locks cannot be acquired, the process is blocked from accessing the table until the locks can be obtained (after the process that currently holds the locks releases them).

External locking affects server performance because the server must sometimes wait for other processes before it can access tables.

External locking is unnecessary if you run a single server to access a given data directory (which is the usual case) and if no other programs such as [myisamchk](#) need to modify tables while the server is running. If you only *read* tables with other programs, external locking is not required, although [myisamchk](#) might report warnings if the server changes tables while [myisamchk](#) is reading them.

With external locking disabled, to use [myisamchk](#), you must either stop the server while [myisamchk](#) executes or else lock and flush the tables before running [myisamchk](#). (See [Section 8.11.1, "System Factors and Startup Parameter Tuning"](#).) To avoid this requirement, use the [CHECK TABLE](#) and [REPAIR TABLE](#) statements to check and repair [MyISAM](#) tables.

For [mysqld](#), external locking is controlled by the value of the [skip_external_locking](#) [552] system variable. When this variable is enabled, external locking is disabled, and vice versa. From MySQL 4.0 on, external locking is disabled by default.

Use of external locking can be controlled at server startup by using the [--external-locking](#) [444] or [--skip-external-locking](#) [460] option.

If you do use external locking option to enable updates to [MyISAM](#) tables from many MySQL processes, you must ensure that the following conditions are satisfied:

- Do not use the query cache for queries that use tables that are updated by another process.

- Do not start the server with the `--delay-key-write=ALL` [441] option or use the `DELAY_KEY_WRITE=1` table option for any shared tables. Otherwise, index corruption can occur.

The easiest way to satisfy these conditions is to always use `--external-locking` [444] together with `--delay-key-write=OFF` [441] and `--query-cache-size=0` [541]. (This is not done by default because in many setups it is useful to have a mixture of the preceding options.)

8.11. Optimizing the MySQL Server

This section discusses optimization techniques for the database server, primarily dealing with system configuration rather than tuning SQL statements. The information in this section is appropriate for DBAs who want to ensure performance and scalability across the servers they manage; for developers constructing installation scripts that include setting up the database; and people running MySQL themselves for development, testing, and so on who want to maximize their own productivity.

8.11.1. System Factors and Startup Parameter Tuning

We start with system-level factors, because some of these decisions must be made very early to achieve large performance gains. In other cases, a quick look at this section may suffice. However, it is always nice to have a sense of how much can be gained by changing factors that apply at this level.

The operating system to use is very important. To get the best use of multiple-CPU machines, you should use Solaris (because its threads implementation works well) or Linux (because the 2.4 and later kernels have good SMP support). Note that older Linux kernels have a 2GB filesize limit by default. If you have such a kernel and a need for files larger than 2GB, get the Large File Support (LFS) patch for the ext2 file system. Other file systems such as ReiserFS and XFS do not have this 2GB limitation.

Before using MySQL in production, test it on your intended platform.

Other tips:

- If you have enough RAM, you could remove all swap devices. Some operating systems use a swap device in some contexts even if you have free memory.
- Avoid external locking for `MyISAM` tables. Since MySQL 4.0, the default has been for external locking to be disabled on all systems. The `--external-locking` [444] and `--skip-external-locking` [460] options explicitly enable and disable external locking.

Note that disabling external locking does not affect MySQL's functionality as long as you run only one server. Just remember to take down the server (or lock and flush the relevant tables) before you run `myisamchk`. On some systems it is mandatory to disable external locking because it does not work, anyway.

The only case in which you cannot disable external locking is when you run multiple MySQL *servers* (not clients) on the same data, or if you run `myisamchk` to check (not repair) a table without telling the server to flush and lock the tables first. Note that using multiple MySQL servers to access the same data concurrently is generally *not* recommended, except when using MySQL Cluster.

The `LOCK TABLES` and `UNLOCK TABLES` statements use internal locking, so you can use them even if external locking is disabled.

8.11.2. Tuning Server Parameters

You can determine the default buffer sizes used by the `mysqld` server using this command:

```
shell> mysqld --verbose --help
```

This command produces a list of all `mysqld` options and configurable system variables. The output includes the default variable values and looks something like this:

```

abort-slave-event-count          0
allow-suspicious-udfs           FALSE
archive                          ON
auto-increment-increment        1
auto-increment-offset           1
autocommit                       TRUE
automatic-sp-privileges          TRUE
back-log                         50
basedir                          /home/jon/bin/mysql-5.5-cluster/
big-tables                       FALSE
bind-address                     (No default value)
binlog-cache-size                32768
binlog-direct-non-transactional-updates FALSE
binlog-format                    STATEMENT
binlog-row-event-max-size        1024
binlog-stmt-cache-size          32768
blackhole                       ON
bulk-insert-buffer-size          8388608
character-set-client-handshake    TRUE
character-set-filesystem         binary
character-set-server            latin1
character-sets-dir               /home/jon/bin/mysql-5.5-cluster/share/charsets/
chroot                          (No default value)
collation-server                 latin1_swedish_ci
completion-type                  NO_CHAIN
concurrent-insert                AUTO
connect-timeout                  10
console                         FALSE
datadir                         (No default value)
date-format                      %Y-%m-%d
datetime-format                  %Y-%m-%d %H:%i:%s
default-storage-engine           InnoDB
default-time-zone                (No default value)
default-week-format              0
delay-key-write                  ON
delayed-insert-limit             100
delayed-insert-timeout           300
delayed-queue-size               1000
des-key-file                     (No default value)
disconnect-slave-event-count     0
div-precision-increment          4
engine-condition-pushdown        TRUE
event-scheduler                  OFF
expire-logs-days                 0
external-locking                 FALSE
flush                            FALSE
flush-time                       0
ft-boolean-syntax                + -><()~*:""&|
ft-max-word-len                  84
ft-min-word-len                  4
ft-query-expansion-limit         20
ft-stopword-file                 (No default value)
gdb                              FALSE
general-log                      FALSE
general-log-file                 /home/jon/bin/mysql-5.5-cluster/data/torsk.log
group-concat-max-len             1024
help                             TRUE
ignore-builtin-innodb            FALSE
init-connect                     (No default value)
init-rpl-role                    MASTER
init-slave                       (No default value)
innodb                           ON
innodb-adaptive-flushing         TRUE

```

Tuning Server Parameters

innodb-adaptive-hash-index	TRUE
innodb-additional-mem-pool-size	8388608
innodb-autoextend-increment	8
innodb-autoinc-lock-mode	1
innodb-buffer-pool-instances	1
innodb-buffer-pool-size	134217728
innodb-change-buffering	all
innodb-checksums	TRUE
innodb-cmp	ON
innodb-cmp-reset	ON
innodb-cmpmem	ON
innodb-cmpmem-reset	ON
innodb-commit-concurrency	0
innodb-concurrency-tickets	500
innodb-data-file-path	(No default value)
innodb-data-home-dir	(No default value)
innodb-doublewrite	TRUE
innodb-fast-shutdown	1
innodb-file-format	Antelope
innodb-file-format-check	TRUE
innodb-file-format-max	Antelope
innodb-file-io-threads	4
innodb-file-per-table	FALSE
innodb-flush-log-at-trx-commit	1
innodb-flush-method	(No default value)
innodb-force-recovery	0
innodb-io-capacity	200
innodb-large-prefix	FALSE
innodb-lock-wait-timeout	50
innodb-lock-waits	ON
innodb-locks	ON
innodb-locks-unsafe-for-binlog	FALSE
innodb-log-buffer-size	8388608
innodb-log-file-size	5242880
innodb-log-files-in-group	2
innodb-log-group-home-dir	(No default value)
innodb-max-dirty-pages-pct	75
innodb-max-purge-lag	0
innodb-mirrored-log-groups	1
innodb-old-blocks-pct	37
innodb-old-blocks-time	0
innodb-open-files	300
innodb-purge-batch-size	20
innodb-purge-threads	0
innodb-read-ahead-threshold	56
innodb-read-io-threads	4
innodb-replication-delay	0
innodb-rollback-on-timeout	FALSE
innodb-rollback-segments	128
innodb-spin-wait-delay	6
innodb-stats-method	nulls_equal
innodb-stats-on-metadata	TRUE
innodb-stats-sample-pages	8
innodb-status-file	FALSE
innodb-strict-mode	FALSE
innodb-support-xa	TRUE
innodb-sync-spin-loops	30
innodb-table-locks	TRUE
innodb-thread-concurrency	0
innodb-thread-sleep-delay	10000
innodb-trx	ON
innodb-use-native-aio	TRUE
innodb-use-sys-malloc	TRUE
innodb-write-io-threads	4
interactive-timeout	28800
join-buffer-size	131072
keep-files-on-create	FALSE

Tuning Server Parameters

key-buffer-size	8388608
key-cache-age-threshold	300
key-cache-block-size	1024
key-cache-division-limit	100
language	/home/jon/bin/mysql-5.5-cluster/share/
large-pages	FALSE
lc-messages	en_US
lc-messages-dir	/home/jon/bin/mysql-5.5-cluster/share/
lc-time-names	en_US
local-infile	TRUE
lock-wait-timeout	31536000
log	/home/jon/bin/mysql-5.5-cluster/data/torsk.log
log-bin	mister-pibb
log-bin-index	(No default value)
log-bin-trust-function-creators	FALSE
log-error	
log-isam	myisam.log
log-output	FILE
log-queries-not-using-indexes	FALSE
log-short-format	FALSE
log-slave-updates	FALSE
log-slow-admin-statements	FALSE
log-slow-queries	/home/jon/bin/mysql-5.5-cluster/data/torsk-slow.log
log-slow-slave-statements	FALSE
log-tc	tc.log
log-tc-size	24576
log-warnings	1
long-query-time	10
low-priority-updates	FALSE
lower-case-table-names	0
master-info-file	master.info
master-retry-count	86400
max-allowed-packet	1048576
max-binlog-cache-size	18446744073709547520
max-binlog-dump-events	0
max-binlog-size	1073741824
max-binlog-stmt-cache-size	18446744073709547520
max-connect-errors	10
max-connections	151
max-delayed-threads	20
max-error-count	64
max-heap-table-size	16777216
max-join-size	18446744073709551615
max-length-for-sort-data	1024
max-long-data-size	1048576
max-prepared-stmt-count	16382
max-relay-log-size	0
max-seeks-for-key	18446744073709551615
max-sort-length	1024
max-sp-recursion-depth	0
max-tmp-tables	32
max-user-connections	0
max-write-lock-count	18446744073709551615
memlock	FALSE
min-examined-row-limit	0
multi-range-count	256
myisam-block-size	1024
myisam-data-pointer-size	6
myisam-max-sort-file-size	9223372036853727232
myisam-mmap-size	18446744073709551615
myisam-recover-options	OFF
myisam-repair-threads	1
myisam-sort-buffer-size	8388608
myisam-stats-method	nulls_unequal
myisam-use-mmap	FALSE
ndb-autoincrement-prefetch-sz	1
ndb-batch-size	32768

Tuning Server Parameters

ndb-blob-read-batch-bytes	65536
ndb-blob-write-batch-bytes	65536
ndb-cache-check-time	0
ndb-cluster-connection-pool	1
ndb-connectstring	(No default value)
ndb-deferred-constraints	0
ndb-distribution	KEYHASH
ndb-extra-logging	1
ndb-force-send	TRUE
ndb-index-stat-cache-entries	32
ndb-index-stat-enable	TRUE
ndb-index-stat-option	loop_checkon=1000ms,loop_idle=1000ms,loop_busy=100ms,update_
ndb-index-stat-update-freq	20
ndb-join-pushdown	TRUE
ndb-log-apply-status	FALSE
ndb-log-bin	TRUE
ndb-log-binlog-index	TRUE
ndb-log-empty-epochs	FALSE
ndb-log-orig	FALSE
ndb-log-transaction-id	FALSE
ndb-log-update-as-write	TRUE
ndb-log-updated-only	TRUE
ndb-mgmd-host	(No default value)
ndb-nodeid	0
ndb-optimization-delay	10
ndb-optimized-node-selection	3
ndb-report-thresh-binlog-epoch-slip	3
ndb-report-thresh-binlog-mem-usage	10
ndb-table-no-logging	FALSE
ndb-table-temporary	FALSE
ndb-use-copying-alter-table	FALSE
ndb-use-exact-count	FALSE
ndb-use-transactions	TRUE
ndb-wait-connected	0
ndb-wait-setup	15
ndbcluster	ON
ndbinfo	ON
ndbinfo-database	ndbinfo
ndbinfo-max-bytes	0
ndbinfo-max-rows	10
ndbinfo-show-hidden	FALSE
ndbinfo-table-prefix	ndb\$
net-buffer-length	16384
net-read-timeout	30
net-retry-count	10
net-write-timeout	60
new	FALSE
old	FALSE
old-alter-table	FALSE
old-passwords	FALSE
old-style-user-limits	FALSE
open-files-limit	1024
optimizer-prune-level	1
optimizer-search-depth	62
optimizer-switch	index_merge=on,index_merge_union=on,index_merge_sort_union=0
partition	ON
performance-schema	FALSE
performance-schema-events-waits-history-long-size	10000
performance-schema-events-waits-history-size	10
performance-schema-max-cond-classes	80
performance-schema-max-cond-instances	1000
performance-schema-max-file-classes	50
performance-schema-max-file-handles	32768
performance-schema-max-file-instances	10000
performance-schema-max-mutex-classes	200
performance-schema-max-mutex-instances	1000000
performance-schema-max-rwlock-classes	30

Tuning Server Parameters

performance-schema-max-rwlock-instances	1000000
performance-schema-max-table-handles	100000
performance-schema-max-table-instances	50000
performance-schema-max-thread-classes	50
performance-schema-max-thread-instances	1000
pid-file	/home/jon/bin/mysql-5.5-cluster/data/torsk.pid
plugin-dir	/home/jon/bin/mysql-5.5-cluster/lib/plugin
plugin-load	(No default value)
port	3306
port-open-timeout	0
preload-buffer-size	32768
profiling-history-size	15
query-alloc-block-size	8192
query-cache-limit	1048576
query-cache-min-res-unit	4096
query-cache-size	0
query-cache-type	ON
query-cache-wlock-invalidate	FALSE
query-prealloc-size	8192
range-alloc-block-size	4096
read-buffer-size	131072
read-only	FALSE
read-rnd-buffer-size	262144
relay-log	(No default value)
relay-log-index	(No default value)
relay-log-info-file	relay-log.info
relay-log-purge	TRUE
relay-log-recovery	FALSE
relay-log-space-limit	0
replicate-same-server-id	FALSE
report-host	(No default value)
report-password	(No default value)
report-port	3306
report-user	(No default value)
rpl-recovery-rank	0
safe-user-create	FALSE
secure-auth	FALSE
secure-file-priv	(No default value)
server-id	1
server-id-bits	32
show-slave-auth-info	FALSE
skip-grant-tables	FALSE
skip-name-resolve	FALSE
skip-networking	FALSE
skip-show-database	FALSE
skip-slave-start	FALSE
slave-allow-batching	FALSE
slave-compressed-protocol	FALSE
slave-exec-mode	STRICT
slave-load-tmpdir	/tmp
slave-net-timeout	3600
slave-skip-errors	(No default value)
slave-transaction-retries	10
slave-type-conversions	
slow-launch-time	2
slow-query-log	FALSE
slow-query-log-file	/home/jon/bin/mysql-5.5-cluster/data/torsk-slow.log
socket	/tmp/mysql.sock
sort-buffer-size	2097152
sporadic-binlog-dump-fail	FALSE
sql-mode	
ssl	FALSE
ssl-ca	(No default value)
ssl-capath	(No default value)
ssl-cert	(No default value)
ssl-cipher	(No default value)
ssl-key	(No default value)

super-large-pages	FALSE
symbolic-links	TRUE
sync-binlog	0
sync_frm	TRUE
sync-master-info	0
sync-relay-log	0
sync-relay-log-info	0
sysdate-is-now	FALSE
table-cache	400
table-definition-cache	400
table-open-cache	400
tc-heuristic-recover	COMMIT
temp-pool	TRUE
thread-cache-size	0
thread-concurrency	10
thread-handling	one-thread-per-connection
thread-stack	262144
time-format	%H:%i:%s
timed-mutexes	FALSE
tmp-table-size	16777216
tmpdir	/tmp
transaction-alloc-block-size	8192
transaction-isolation	REPEATABLE-READ
transaction-prealloc-size	4096
updatable-views-with-limit	YES
verbose	TRUE
wait-timeout	28800

For a `mysqld` server that is currently running, you can see the current values of its system variables by connecting to it and issuing this statement:

```
mysql> SHOW VARIABLES;
```

You can also see some statistical and status indicators for a running server by issuing this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also can be obtained using `mysqladmin`:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

For a full description of all system and status variables, see [Section 5.1.4, “Server System Variables”](#), and [Section 5.1.6, “Server Status Variables”](#).

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally you get better performance by giving MySQL more memory.

When tuning a MySQL server, the two most important variables to configure are `key_buffer_size` [508] and `table_open_cache` [561]. You should first feel confident that you have these set appropriately before trying to change any other variables.

The following examples indicate some typical variable values for different runtime configurations.

- If you have at least 256MB of memory and many tables and want maximum performance with a moderate number of clients, use something like this:

```
shell> mysqld_safe --key_buffer_size=64M --table_open_cache=256 \
--sort_buffer_size=4M --read_buffer_size=1M &
```

- If you have only 128MB of memory and only a few tables, but you still do a lot of sorting, you can use something like this:

```
shell> mysqld_safe --key_buffer_size=16M --sort_buffer_size=1M
```

If there are very many simultaneous connections, swapping problems may occur unless `mysqld` has been configured to use very little memory for each connection. `mysqld` performs better if you have enough memory for all connections.

- With little memory and lots of connections, use something like this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
        --read_buffer_size=100K &
```

Or even this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
        --table_open_cache=32 --read_buffer_size=8K \
        --net_buffer_length=1K &
```

If you are performing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, increase the value of `read_rnd_buffer_size` [545] to speed up the reading of rows following sorting operations.

You can make use of the example option files included with your MySQL distribution; see [Section 5.1.2, “Server Configuration Defaults”](#).

If you specify an option on the command line for `mysqld` or `mysqld_safe`, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file.

To see the effects of a parameter change, do something like this:

```
shell> mysqld --key_buffer_size=32M --verbose --help
```

The variable values are listed near the end of the output. Make sure that the `--verbose` [467] and `--help` [434] options are last. Otherwise, the effect of any options listed after them on the command line are not reflected in the output.

For information on tuning the `InnoDB` storage engine, see [Section 14.3.14.1, “InnoDB Performance Tuning Tips”](#).

8.11.3. Optimizing Disk I/O

The `SHOW COLUMNS` and The `DESCRIBE` statements use `BLOB` as the type for some columns, thus the temporary table used for the results is an on-disk table.

The tips in [Section 8.5.7, “Optimizing InnoDB Disk I/O”](#) help you get more I/O performance out of your existing storage configuration. This section describes ways to configure your storage devices when you can devote more and faster storage hardware to the database server.

- Disk seeks are a huge performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.
- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:
 - Using symbolic links

This means that, for `MyISAM` tables, you symlink the index file and data files from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read

times better, assuming that the disk is not used for other purposes as well. See [Section 8.11.3.1, “Using Symbolic Links”](#).

- **Striping**

Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the N -th block on the $(N \text{ MOD } \textit{number_of_disks})$ disk, and so on. This means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See [Section 8.12.3, “Using Your Own Benchmarks”](#).

The speed difference for striping is *very* dependent on the parameters. Depending on how you set the striping parameters and number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.

- For reliability, you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need $2 \times N$ drives to hold N drives of data. This is probably the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.
- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID N disk. RAID N can be a problem if you have many writes, due to the time required to update the parity bits.
- On Linux, you can get much better performance by using `hdparm` to configure your disk's interface. (Up to 100% under load is not uncommon.) The following `hdparm` options should be quite good for MySQL, and probably for many other applications:

```
hdparm -m 16 -d 1
```

Note that performance and reliability when using this command depend on your hardware, so we strongly suggest that you test your system thoroughly after using `hdparm`. Please consult the `hdparm` manual page for more information. If `hdparm` is not used wisely, file system corruption may result, so back up everything before experimenting!

- You can also set the parameters for the file system that the database uses:

If you do not need to know when files were last accessed (which is not really useful on a database server), you can mount your file systems with the `-o noatime` option. That skips updates to the last access time in inodes on the file system, which avoids some disk seeks.

On many operating systems, you can set a file system to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you better performance without sacrificing too much reliability. (This flag is on by default on Linux.)

8.11.3.1. Using Symbolic Links

You can move databases or tables from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disks.

The recommended way to do this is to symlink entire database directories to a different disk. Symlink `MyISAM` tables only as a last resort.

To determine the location of your data directory, use this statement:

```
SHOW VARIABLES LIKE 'datadir';
```

8.11.3.1.1. Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is first to create a directory on some disk where you have free space and then to create a soft link to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL does not support linking one directory to multiple databases. Replacing a database directory with a symbolic link works as long as you do not make a symbolic link between databases. Suppose that you have a database `db1` under the MySQL data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

The result is that, or any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one client updates `db1.tbl_a` and another client updates `db2.tbl_a`, problems are likely to occur.

8.11.3.1.2. Using Symbolic Links for MyISAM Tables on Unix

Symlinks are fully supported only for MyISAM tables. For files used by tables for other storage engines, you may get strange problems if you try to use symbolic links.

Do not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`.) To determine whether your system supports symbolic links, check the value of the `have_symlink` [505] system variable using this statement:

```
SHOW VARIABLES LIKE 'have_symlink';
```

The handling of symbolic links for MyISAM tables works as follows:

- In the data directory, you always have the table format (`.frm`) file, the data (`.MYD`) file, and the index (`.MYI`) file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The format file cannot.
- You can symlink the data file and the index file independently to different directories.
- To instruct a running MySQL server to perform the symlinking, use the `DATA DIRECTORY` and `INDEX DIRECTORY` options to `CREATE TABLE`. See [Section 13.1.17, “CREATE TABLE Syntax”](#). Alternatively, if `mysqld` is not running, symlinking can be accomplished manually using `ln -s` from the command line.



Note

The path used with either or both of the `DATA DIRECTORY` and `INDEX DIRECTORY` options may not include the MySQL `data` directory. (Bug #32167)

- `myisamchk` does not replace a symlink with the data file or index file. It works directly on the file to which the symlink points. Any temporary files are created in the directory where the data file or index file is located. The same is true for the `ALTER TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.



Note

When you drop a table that is using symlinks, *both the symlink and the file to which the symlink points are dropped*. This is an extremely good reason *not* to run `mysqld` as the system `root` or permit system users to have write access to MySQL database directories.

- If you rename a table with `ALTER TABLE ... RENAME` or `RENAME TABLE` and you do not move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you use `ALTER TABLE ... RENAME` or `RENAME TABLE` to move a table to another database, the table is moved to the other database directory. If the table name changed, the symlinks in the new database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you are not using symlinks, start `mysqld` with the `--skip-symbolic-links` [462] option to ensure that no one can use `mysqld` to drop or rename a file outside of the data directory.

These table symlink operations are not supported:

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.
- As indicated previously, only the data and index files can be symbolic links. The `.frm` file must *never* be a symbolic link. Attempting to do this (for example, to make one table name a synonym for another) produces incorrect results. Suppose that you have a database `db1` under the MySQL data directory, a table `tb11` in this database, and in the `db1` directory you make a symlink `tb12` that points to `tb11`:

```
shell> cd /path/to/datadir/db1
shell> ln -s tb11.frm tb12.frm
shell> ln -s tb11.MYD tb12.MYD
shell> ln -s tb11.MYI tb12.MYI
```

Problems result if one thread reads `db1.tb11` and another thread updates `db1.tb12`:

- The query cache is “fooled” (it has no way of knowing that `tb11` has not been updated, so it returns outdated results).
- `ALTER` statements on `tb12` fail.

8.11.3.1.3. Using Symbolic Links for Databases on Windows

On Windows, symbolic links can be used for database directories. This enables you to put a database directory at a different location (for example, on a different disk) by setting up a symbolic link to it. Use of database symlinks on Windows is similar to their use on Unix, although the procedure for setting up the link differs.

Suppose that you want to place the database directory for a database named `mydb` at `D:\data\mydb`. To do this, create a symbolic link in the MySQL data directory that points to `D:\data\mydb`. However, before creating the symbolic link, make sure that the `D:\data\mydb` directory exists by creating it if necessary. If you already have a database directory named `mydb` in the data directory, move it to `D:\data`. Otherwise, the symbolic link will be ineffective. To avoid problems, make sure that the server is not running when you move the database directory.

The procedure for creating the database symbolic link depends on your version of Windows.

Windows Vista, Windows Server 2008, or newer have native symbolic link support, so you can create a symlink using the `mklink` command. This command requires administrative privileges.

1. Change location into the data directory:

```
C:\> cd \path\to\datadir
```

2. In the data directory, create a symlink named `mydb` that points to the location of the database directory:

```
C:\> mklink /d mydb D:\data\mydb
```

After this, all tables created in the database `mydb` are created in `D:\data\mydb`.

Alternatively, on any version of Windows supported by MySQL, you can create a symbolic link to a MySQL database by creating a `.sym` file in the data directory that contains the path to the destination directory. The file should be named `db_name.sym`, where `db_name` is the database name.

Support for database symbolic links on Windows using `.sym` files is enabled by default. If you do not need `.sym` file symbolic links, you can disable support for them by starting `mysqld` with the `--skip-symbolic-links` [462] option. To determine whether your system supports `.sym` file symbolic links, check the value of the `have_symlink` [505] system variable using this statement:

```
SHOW VARIABLES LIKE 'have_symlink';
```

To create a `.sym` file symlink, use this procedure:

1. Change location into the data directory:

```
C:\> cd \path\to\datadir
```

2. In the data directory, create a text file named `mydb.sym` that contains this path name: `D:\data\mydb\`



Note

The path name to the new database and tables should be absolute. If you specify a relative path, the location will be relative to the `mydb.sym` file.

After this, all tables created in the database `mydb` are created in `D:\data\mydb`.

The following limitations apply to the use of `.sym` files for database symbolic linking on Windows. These limitations do not apply for symlinks created using `mklink`.

- The symbolic link is not used if a directory with the same name as the database exists in the MySQL data directory.
- The `--innodb_file_per_table` [1555] option cannot be used.
- If you run `mysqld` as a service, you cannot use a mapped drive to a remote server as the destination of the symbolic link. As a workaround, you can use the full path (`\\servername\path\`).

8.11.4. Optimizing Memory Use

8.11.4.1. How MySQL Uses Memory

The following list indicates some of the ways that the `mysqld` server uses memory. Where applicable, the name of the system variable relevant to the memory use is given:

- All threads share the `MyISAM` key buffer; its size is determined by the `key_buffer_size` [508] variable. Other buffers used by the server are allocated as needed. See [Section 8.11.2, “Tuning Server Parameters”](#).
- Each thread that is used to manage client connections uses some thread-specific space. The following list indicates these and which variables control their size:
 - A stack (variable `thread_stack` [566])
 - A connection buffer (variable `net_buffer_length` [531])

- A result buffer (variable `net_buffer_length` [531])

The connection buffer and result buffer each begin with a size equal to `net_buffer_length` [531] bytes, but are dynamically enlarged up to `max_allowed_packet` [519] bytes as needed. The result buffer shrinks to `net_buffer_length` [531] bytes after each SQL statement. While a statement is running, a copy of the current statement string is also allocated.

- All threads share the same base memory.
- When a thread is no longer needed, the memory allocated to it is released and returned to the system unless the thread goes back into the thread cache. In that case, the memory remains allocated.
- The `myisam_use_mmap` [531] system variable can be set to 1 to enable memory-mapping for all MyISAM tables.
- Each request that performs a sequential scan of a table allocates a *read buffer* (variable `read_buffer_size` [544]).
- When reading rows in an arbitrary sequence (for example, following a sort), a *random-read buffer* (variable `read_rnd_buffer_size` [545]) may be allocated to avoid disk seeks.
- All joins are executed in a single pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based hash tables. Temporary tables with a large row length (calculated as the sum of all column lengths) or that contain BLOB columns are stored on disk.

If an internal in-memory temporary table becomes too large, MySQL handles this automatically by changing the table from in-memory to on-disk format, to be handled by the MyISAM storage engine. You can increase the permissible temporary table size as described in [Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#).

- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See [Section C.5.4.4, “Where MySQL Stores Temporary Files”](#).
- Almost all parsing and calculating is done in thread-local and reusable memory pools. No memory overhead is needed for small items, so the normal slow memory allocation and freeing is avoided. Memory is allocated only for unexpectedly large strings.
- For each MyISAM table that is opened, the index file is opened once; the data file is opened once for each concurrently running thread. For each concurrent thread, a table structure, column structures for each column, and a buffer of size $3 * N$ are allocated (where N is the maximum row length, not counting BLOB columns). A BLOB column requires five to eight bytes plus the length of the BLOB data. The MyISAM storage engine maintains one extra row buffer for internal use.
- For each table having BLOB columns, a buffer is enlarged dynamically to read in larger BLOB values. If you scan a table, a buffer as large as the largest BLOB value is allocated.
- Handler structures for all in-use tables are saved in a cache and managed as a FIFO. The initial cache size is taken from the value of the `table_open_cache` [561] system variable. If a table has been used by two running threads at the same time, the cache contains two entries for the table. See [Section 8.4.3.1, “How MySQL Opens and Closes Tables”](#).
- A `FLUSH TABLES` statement or `mysqladmin flush-tables` command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. `FLUSH TABLES` does not return until all tables have been closed.
- The server caches information in memory as a result of `GRANT`, `CREATE USER`, `CREATE SERVER`, and `INSTALL PLUGIN` statements. This memory is not released by the corresponding `REVOKE`, `DROP`

`USER`, `DROP SERVER`, and `UNINSTALL PLUGIN` statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with `FLUSH PRIVILEGES`.

`ps` and other system status programs may report that `mysqld` uses a lot of memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of `ps` counts the unused memory between stacks as used memory. To verify this, check available swap with `swap -s`. We test `mysqld` with several memory-leakage detectors (both commercial and Open Source), so there should be no memory leaks.

8.11.4.2. Enabling Large Page Support

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

In MySQL, large pages can be used by InnoDB, to allocate memory for its buffer pool and additional memory pool.

Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a “super large pages” feature enables uses of pages up to 256MB. This feature is available for recent SPARC platforms. It can be enabled or disabled by using the `--super-large-pages [462]` or `--skip-super-large-pages [462]` option.

MySQL also supports the Linux implementation of large page support (which is called HugeTLB in Linux).

Before large pages can be used on Linux, the kernel must be enabled to support them and it is necessary to configure the HugeTLB memory pool. For reference, the HugeTBL API is documented in the [Documentation/vm/hugetlbpage.txt](#) file of your Linux sources.

The kernel for some recent systems such as Red Hat Enterprise Linux appear to have the large pages feature enabled by default. To check whether this is true for your kernel, use the following command and look for output lines containing “huge”:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      0
HugePages_Free:       0
HugePages_Rsvd:       0
HugePages_Surp:       0
Hugepagesize:         4096 kB
```

The nonempty command output indicates that large page support is present, but the zero values indicate that no pages are configured for use.

If your kernel needs to be reconfigured to support large pages, consult the [hugetlbpage.txt](#) file for instructions.

Assuming that your Linux kernel has large page support enabled, configure it for use by MySQL using the following commands. Normally, you put these in an `rc` file or equivalent startup file that is executed during the system boot sequence, so that the commands execute each time the system starts. The commands should execute early in the boot sequence, before the MySQL server starts. Be sure to change the allocation numbers and the group number as appropriate for your system.

```
# Set the number of pages to be used.
# Each page is normally 2MB, so a value of 20 = 40MB.
# This command actually allocates memory, so this much
# memory must be available.
```

```
echo 20 > /proc/sys/vm/nr_hugepages

# Set the group number that is permitted to access this
# memory (102 in this case). The mysql user must be a
# member of this group.
echo 102 > /proc/sys/vm/hugetlb_shm_group

# Increase the amount of shmem permitted per segment
# (12G in this case).
echo 1560281088 > /proc/sys/kernel/shmmax

# Increase total amount of shared memory. The value
# is the number of pages. At 4KB/page, 4194304 = 16GB.
echo 4194304 > /proc/sys/kernel/shmall
```

For MySQL usage, you normally want the value of `shmmax` to be close to the value of `shmall`.

To verify the large page configuration, check `/proc/meminfo` again as described previously. Now you should see some nonzero values:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:       0
HugePages_Surp:       0
Hugepagesize:         4096 kB
```

The final step to make use of the `hugetlb_shm_group` is to give the `mysql` user an “unlimited” value for the memlock limit. This can be done either by editing `/etc/security/limits.conf` or by adding the following command to your `mysqld_safe` script:

```
ulimit -l unlimited
```

Adding the `ulimit` command to `mysqld_safe` causes the `root` user to set the memlock limit to `unlimited` before switching to the `mysql` user. (This assumes that `mysqld_safe` is started by `root`.)

Large page support in MySQL is disabled by default. To enable it, start the server with the `--large-pages [446]` option. For example, you can use the following lines in your server's `my.cnf` file:

```
[mysqld]
large-pages
```

With this option, `InnoDB` uses large pages automatically for its buffer pool and additional memory pool. If `InnoDB` cannot do this, it falls back to use of traditional memory and writes a warning to the error log:
`Warning: Using conventional memory pool`

To verify that large pages are being used, check `/proc/meminfo` again:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:       2
HugePages_Surp:       0
Hugepagesize:         4096 kB
```

8.11.5. Optimizing Network Use

8.11.5.1. How MySQL Uses Threads for Client Connections

Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this

manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests. The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

By default, connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

In the default connection thread model, there are as many threads as there are clients currently connected, which has some disadvantages when server workload must scale to handle large numbers of connections. For example, thread creation and disposal becomes expensive. Also, each thread requires server and kernel resources, such as stack space. To accommodate a large number of simultaneous connections, the stack size per thread must be kept small, leading to a situation where it is either too small or the server consumes large amounts of memory. Exhaustion of other resources can occur as well, and scheduling overhead can become significant.

As of MySQL 5.5.16, commercial distributions of MySQL include a thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. It implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections. See [Section 8.11.6, “The Thread Pool Plugin”](#).

To control and monitor how the server manages threads that handle client connections, several system and status variables are relevant. (See [Section 5.1.4, “Server System Variables”](#), and [Section 5.1.6, “Server Status Variables”](#).)

The thread cache has a size determined by the `thread_cache_size` [561] system variable. The default value is 0 (no caching), which causes a thread to be set up for each new connection and disposed of when the connection terminates. Set `thread_cache_size` [561] to *N* to enable *N* inactive connection threads to be cached. `thread_cache_size` [561] can be set at server startup or changed while the server runs. A connection thread becomes inactive when the client connection with which it was associated terminates.

To monitor the number of threads in the cache and how many threads have been created because a thread could not be taken from the cache, monitor the `Threads_cached` [610] and `Threads_created` [610] status variables.

You can set `max_connections` [519] at server startup or at runtime to control the maximum number of clients that can connect simultaneously.

When the thread stack is too small, this limits the complexity of the SQL statements which the server can handle, the recursion depth of stored procedures, and other memory-consuming actions. To set a stack size of *N* bytes for each thread, start the server with `--thread_stack=N` [566].

8.11.5.2. DNS Lookup Optimization and the Host Cache

The MySQL server maintains a host cache in memory that contains information about clients: IP address, host name, and error information. The server uses this cache for nonlocal TCP connections. It does not use the cache for TCP connections established using a loopback interface address (127.0.0.1 or ::1), or for connections established using a Unix socket file, named pipe, or shared memory.

For each new client connection, the server uses the client IP address to check whether the client host name is in the host cache. If not, the server attempts to resolve the host name. First, it resolves the IP address to a host name and resolves that host name back to an IP address. Then it compares the result to

the original IP address to ensure that they are the same. The server stores information about the result of this operation in the host cache. If the cache is full, the least recently used entry is discarded.

The server performs host name resolution using the thread-safe `gethostbyaddr_r()` and `gethostbyname_r()` calls if the operating system supports them. Otherwise, the thread performing the lookup locks a mutex and calls `gethostbyaddr()` and `gethostbyname()` instead. In this case, no other thread can resolve host names that are not in the host cache until the thread holding the mutex lock releases it.

The server uses the host cache for several purposes:

- By caching the results of IP-to-host name lookups, the server avoids doing a DNS lookup for each client connection. Instead, for a given host, it needs to perform a lookup only for the first connection from that host.
- The cache contains information about errors that occur during the connection process. Some errors are considered “blocking.” If too many of these occur successively from a given host without a successful connection, the server blocks further connections from that host. The `max_connect_errors` [519] system variable determines the number of permitted errors before blocking occurs. See Section C.5.2.6, “Host ‘host_name’ is blocked”.

To unblock blocked hosts, flush the host cache by issuing a `FLUSH HOSTS` statement or executing a `mysqladmin flush-hosts` command.

It is possible for a blocked host to become unblocked even without `FLUSH HOSTS` if activity from other hosts has occurred since the last connection attempt from the blocked host. This can occur because the server discards the least recently used cache entry to make room for a new entry if the cache is full when a connection arrives from a client IP not in the cache. If the discarded entry is for a blocked host, that host becomes unblocked.

The host cache is enabled by default. To disable it, start the server with the `--skip-host-cache` [461] option.

To disable DNS host name lookups, start the server with the `--skip-name-resolve` [461] option. In this case, the server uses only IP addresses and not host names to match connecting hosts to rows in the MySQL grant tables. Only accounts specified in those tables using IP addresses can be used.

If you have a very slow DNS and many hosts, you might be able to improve performance either by disabling DNS lookups with `--skip-name-resolve` [461] or by increasing the `HOST_CACHE_SIZE` define (default value: 128) and recompiling the server

To disallow TCP/IP connections entirely, start the server with the `--skip-networking` [461] option.

Some connection errors are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions).

8.11.6. The Thread Pool Plugin



Note

MySQL Thread Pool is a commercial extension. To learn more about commercial products (MySQL Enterprise Edition), see <http://www.mysql.com/products/>.

As of MySQL 5.5.16, commercial distributions of MySQL include MySQL Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using

one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. The thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

The thread pool addresses several problems of the one thread per connection model:

- Too many thread stacks make CPU caches almost useless in highly parallel execution workloads. The thread pool promotes thread stack reuse to minimize the CPU cache footprint.
- With too many threads executing in parallel, context switching overhead is high. This also presents a challenging task to the operating system scheduler. The thread pool controls the number of active threads to keep the parallelism within the MySQL server at a level that it can handle and that is appropriate for the server host on which MySQL is executing.
- Too many transactions executing in parallel increases resource contention. In [InnoDB](#), this increases the time spent holding central mutexes. The thread pool controls when transactions start to ensure that not too many execute in parallel.

The thread pool plugin is a commercial feature. It is not included in MySQL community distributions.

On Windows, the thread pool plugin requires Windows Vista or newer. On Linux, the plugin requires kernel 2.6.9 or newer.

Additional Resources

[Section B.14, “MySQL 5.5 FAQ: MySQL Enterprise Scalability Thread Pool”](#)

8.11.6.1. Thread Pool Components and Installation

The thread pool feature comprises these components:

- A plugin library object file contains a plugin for the thread pool code and plugins for several [INFORMATION_SCHEMA](#) tables.

For a detailed description of how the thread pool works, see [Section 8.11.6.2, “Thread Pool Operation”](#).

The [INFORMATION_SCHEMA](#) tables are named [TP_THREAD_STATE](#), [TP_THREAD_GROUP_STATE](#), and [TP_THREAD_GROUP_STATS](#). These tables provide information about thread pool operation. For more information, see [Section 20.30, “Thread Pool INFORMATION_SCHEMA Tables”](#).

- Several system variables are related to the thread pool. The [thread_handling](#) [562] system variable has a value of `loaded-dynamically` when the server successfully loads the thread pool plugin.

The other related variables are implemented by the thread pool plugin; they are not available unless it is enabled:

- [thread_pool_algorithm](#) [563]: The concurrency algorithm to use for scheduling.
- [thread_pool_high_priority_connection](#) [563]: How to schedule statement execution for a session.
- [thread_pool_prio_kickup_timer](#) [565]: How long before the thread pool moves a statement awaiting execution from the low-priority queue to the high-priority queue.
- [thread_pool_max_unused_threads](#) [564]: How many sleeping threads to permit.

- `thread_pool_size` [565]: The number of thread groups in the thread pool. This is the most important parameter controlling thread pool performance.
- `thread_pool_stall_limit` [566]: The time before an executing statement is considered to be stalled.

If any variable implemented by the plugin is set to an illegal value at startup, plugin initialization fails and the plugin does not load.

For information about setting thread pool parameters, see [Section 8.11.6.3, “Thread Pool Tuning”](#).

- The Performance Schema exposes information about the thread pool and may be used to investigate operational performance. For more information, see [Chapter 21, MySQL Performance Schema](#).

To be usable by the server, the thread pool library object file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` [537] system variable). To enable thread pool capability, load the plugins to be used by starting the server with the `--plugin-load` [457] option. For example, if you name just the plugin object file, the server loads all plugins that it contains (that is, the thread pool plugin and all the `INFORMATION_SCHEMA` tables). To do this, put these lines in your `my.cnf` file:

```
[mysqld]
plugin-load=thread_pool.so
```

If object files have a suffix different from `.so` on your system, substitute the correct suffix (for example, `.dll` on Windows).

If necessary, set the value of the `plugin_dir` [537] system variable to tell the server the location of the plugin directory.

If desired, you can load individual plugins from the library file. To load the thread pool plugin but not the `INFORMATION_SCHEMA` tables, use an option like this:

```
[mysqld]
plugin-load=thread_pool=thread_pool.so
```

To load the thread pool plugin and only the `TP_THREAD_STATE INFORMATION_SCHEMA` table, use an option like this:

```
[mysqld]
plugin-load=thread_pool=thread_pool.so;TP_THREAD_STATE=thread_pool.so
```

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement. See [Section 5.1.8.2, “Obtaining Server Plugin Information”](#).

If the server loads the thread plugin successfully, it sets the `thread_handling` system variable to `dynamically-loaded`. If the plugin fails to load, the server writes a message to the error log.

8.11.6.2. Thread Pool Operation

The thread pool consists of a number of thread groups, each of which manages a set of client connections. As connections are established, the thread pool assigns them to thread groups in round-robin fashion.

The number of thread groups is configurable using the `thread_pool_size` [565] system variable. The default number of groups is 16. For guidelines on setting this variable, see [Section 8.11.6.3, “Thread Pool Tuning”](#).

The maximum number of threads per group is 4096 (or 4095 on some systems where one thread is used internally).

The thread pool separates connections and threads, so there is no fixed relationship between connections and the threads that execute statements received from those connections. This differs from the default thread-handling model that associates one thread with one connection such that the thread executes all statements from the connection.

The thread pool tries to ensure a maximum of one thread executing in each group at any time, but sometimes permits more threads to execute temporarily for best performance. The algorithm works in the following manner:

- Each thread group has a listener thread that listens for incoming statements from the connections assigned to the group. When a statement arrives, the thread group either begins executing it immediately or queues it for later execution:
 - Immediate execution occurs if the statement is the only one received and no statements are queued or currently executing.
 - Queuing occurs if the statement cannot begin executing immediately.
- If immediate execution occurs, execution is performed by the listener thread. (This means that temporarily no thread in the group is listening.) If the statement finishes quickly, the executing thread returns to listening for statements. Otherwise, the thread pool considers the statement stalled and starts another thread as a listener thread (creating it if necessary). To ensure that no thread group becomes blocked by stalled statements, the thread pool has a background thread that regularly monitors thread group states.

By using the listening thread to execute a statement that can begin immediately, there is no need to create an additional thread if the statement finishes quickly. This ensures the most efficient execution possible in the case of a low number of concurrent threads.

When the thread pool plugin starts, it creates one thread per group (the listener thread), plus the background thread. Additional threads are created as necessary to execute statements.

- The value of the `thread_pool_stall_limit` [566] system variable determines the meaning of “finishes quickly” in the previous item. The default time before threads are considered stalled is 60ms but can be set to a maximum of 6s. This parameter is configurable to enable you to strike a balance appropriate for the server work load. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.
- The thread pool focuses on limiting the number of concurrent short-running statements. Before an executing statement reaches the stall time, it prevents other statements from beginning to execute. If the statement executes past the stall time, it is permitted to continue but no longer prevents other statements from starting. In this way, the thread pool tries to ensure that in each thread group there is never more than one short-running statement, although there might be multiple long-running statements. It is undesirable to let long-running statements prevent other statements from executing because there is no limit on the amount of waiting that might be necessary. For example, on a replication master, a thread that is sending binary log events to a slave effectively runs forever.
- A statement becomes blocked if it encounters a disk I/O operation or a user level lock (row lock or table lock). The block would cause the thread group to become unused, so there are callbacks to the thread pool to ensure that the thread pool can immediately start a new thread in this group to execute another statement. When a blocked thread returns, the thread pool permits it to restart immediately.
- There are two queues, a high-priority queue and a low-priority queue. The first statement in a transaction goes to the low-priority queue. Any following statements for the transaction go to the high-priority queue if the transaction is ongoing (statements for it have begun executing),

or to the low-priority queue otherwise. Queue assignment can be affected by enabling the `thread_pool_high_priority_connection` [563] system variable, which causes all queued statements for a session to go into the high-priority queue.

Statements for a nontransactional storage engine, or a transactional engine if `autocommit` [484] is enabled, are treated as low-priority statements because in this case each statement is a transaction. Thus, given a mix of statements for `InnoDB` and `MyISAM` tables, the thread pool prioritizes those for `InnoDB` over those for `MyISAM` unless `autocommit` [484] is enabled. With `autocommit` [484] enabled, all statements will be low priority.

- When the thread group selects a queued statement for execution, it first looks in the high-priority queue, then in the low-priority queue. If a statement is found, it is removed from its queue and begins to execute.
- If a statement stays in the low-priority queue too long, the thread pool moves to the high-priority queue. The value of the `thread_pool_prio_kickup_timer` [565] system variable controls the time before movement. For each thread group, a maximum of one statement per 10ms or 100 per second will be moved from the low-priority queue to the high-priority queue.
- The thread pool reuses the most active threads to obtain a much better use of CPU caches. This is a small adjustment that has a great impact on performance.
- While a thread executes a statement from a user connection, Performance Schema instrumentation accounts thread activity to the user connection. Otherwise, Performance Schema accounts activity to the thread pool.

Here are examples of conditions under which a thread group might have multiple threads started to execute statements:

- One thread begins executing a statement, but runs long enough to be considered stalled. The thread group permits another thread to begin executing another statement even though the first thread is still executing.
- One thread begins executing a statement, then becomes blocked and reports this back to the thread pool. The thread group permits another thread to begin executing another statement.
- One thread begins executing a statement, becomes blocked, but does not report back that it is blocked because the block does not occur in code that has been instrumented with thread pool callbacks. In this case, the thread appears to the thread group to be still running. If the block lasts long enough for the statement to be considered stalled, the group permits another thread to begin executing another statement.

The thread pool is designed to be scalable across an increasing number of connections. It is also designed to avoid deadlocks that can arise from limiting the number of actively executing statements. It is important that threads that do not report back to the thread pool do not prevent other statements from executing and thus cause the thread pool to become deadlocked. Examples of such statements follow:

- Long-running statements. These would lead to all resources used by only a few statements and they could prevent all others from accessing the server.
- Binary log dump threads that read the binary log and send it to slaves. This is a kind of long-running “statement” that runs for a very long time, and that should not prevent other statements from executing.
- Statements blocked on a row lock, table lock, sleep, or any other blocking activity that has not been reported back to the thread pool by MySQL Server or a storage engine.

In each case, to prevent deadlock, the statement is moved to the stalled category when it does not complete quickly, so that the thread group can permit another statement to begin executing. With this

design, when a thread executes or becomes blocked for an extended time, the thread pool moves the thread to the stalled category and for the rest of the statement's execution, it does not prevent other statements from executing.

The maximum number of threads that can occur is the sum of `max_connections` [519] and `thread_pool_size` [565]. This can happen in a situation where all connections are in execution mode and an extra thread is created per group to listen for more statements. This is not necessarily a state that happens often, but it is theoretically possible.

8.11.6.3. Thread Pool Tuning

This section provides guidelines on setting thread pool system variables for best performance, measured using a metric such as transactions per second.

`thread_pool_size` [565] is the most important parameter controlling thread pool performance. It can be set only at server startup. Our experience in testing the thread pool indicates the following:

- If the primary storage engine is `InnoDB`, the optimal `thread_pool_size` [565] setting is likely to be between 16 and 36, with the most common optimal values tending to be from 24 to 36. We have not seen any situation where the setting has been optimal beyond 36. There may be special cases where a value smaller than 16 is optimal.

For workloads such as DBT2 and Sysbench, the optimum for `InnoDB` seems to be usually around 36. For very write-intensive workloads, the optimal setting can sometimes be lower.

- If the primary storage engine is `MyISAM`, the `thread_pool_size` [565] setting should be fairly low. We tend to get optimal performance for values from 4 to 8. Higher values tend to have a slightly negative but not dramatic impact on performance.

Another system variable, `thread_pool_stall_limit` [566], is important for handling of blocked and long-running statements. If all calls that block the MySQL Server are reported to the thread pool, it would always know when execution threads are blocked. However, this may not always be true. For example, blocks could occur in code that has not been instrumented with thread pool callbacks. For such cases, the thread pool must be able to identify threads that appear to be blocked. This is done by means of a timeout, the length of which can be tuned using the `thread_pool_stall_limit` [566] system variable. This parameter ensures that the server does not become completely blocked. The value of `thread_pool_stall_limit` [566] has an upper limit of 6 seconds to prevent the risk of a deadlocked server.

`thread_pool_stall_limit` [566] also enables the thread pool to handle long-running statements. If a long-running statement was permitted to block a thread group, all other connections assigned to the group would be blocked and unable to start execution until the long-running statement completed. In the worst case, this could take hours or even days.

The value of `thread_pool_stall_limit` [566] should be chosen such that statements that execute longer than its value are considered stalled. Stalled statements generate a lot of extra overhead since they involve extra context switches and in some cases even extra thread creations. On the other hand, setting the `thread_pool_stall_limit` [566] parameter too high means that long-running statements will block a number of short-running statements for longer than necessary. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.

Suppose a server executes a workload where 99.9% of the statements complete within 100ms even when the server is loaded, and the remaining statements take between 100ms and 2 hours fairly evenly spread. In this case, it would make sense to set `thread_pool_stall_limit` [566] to 10 (meaning 100ms). The default value of 60ms is okay for servers that primarily execute very simple statements.

The `thread_pool_stall_limit` [566] parameter can be changed at runtime to enable you to strike a balance appropriate for the server work load. Assuming that the `TP_THREAD_GROUP_STATS` table is enabled, you can use the following query to determine the fraction of executed statements that stalled:

```
SELECT SUM(STALLED_QUERIES_EXECUTED) / SUM(QUERIES_EXECUTED)
FROM information_schema.TP_THREAD_GROUP_STATS;
```

This number should be as low as possible. To decrease the likelihood of statements stalling, increase the value of `thread_pool_stall_limit` [566].

When a statement arrives, what is the maximum time it can be delayed before it actually starts executing? Suppose that the following conditions apply:

- There are 200 statements queued in the low-priority queue.
- There are 10 statements queued in the high-priority queue.
- `thread_pool_prio_kickup_timer` [565] is set to 10000 (10 seconds).
- `thread_pool_stall_limit` [566] is set to 100 (1 second).

In the worst case, the 10 high-priority statements represent 10 transactions that continue executing for a long time. Thus, in the worst case, no statements will be moved to the high-priority queue because it will always already contain statements awaiting execution. After 10 seconds, the new statement is eligible to be moved to the high-priority queue. However, before it can be moved, all the statements before it must be moved as well. This could take another 2 seconds because a maximum of 100 statements per second are moved to the high-priority queue. Now when the statement reaches the high-priority queue, there could potentially be many long-running statements ahead of it. In the worst case, every one of those will become stalled and it will take 1 second for each statement before the next statement is retrieved from the high-priority queue. Thus, in this scenario, it will take 222 seconds before the new statement starts executing.

This example shows a worst case for an application. How to handle it depends on the application. If the application has high requirements for the response time, it should most likely throttle users at a higher level itself. Otherwise, it can use the thread pool configuration parameters to set some kind of a maximum waiting time.

8.12. Measuring Performance (Benchmarking)

To measure performance, consider the following factors:

- Whether you are measuring the speed of a single operation on a quiet system, or how a set of operations (a “workload”) works over a period of time. With simple tests, you usually test how changing one aspect (a configuration setting, the set of indexes on a table, the SQL clauses in a query) affects performance. Benchmarks are typically long-running and elaborate performance tests, where the results could dictate high-level choices such as hardware and storage configuration, or how soon to upgrade to a new MySQL version.
- For benchmarking, sometimes you must simulate a heavy database workload to get an accurate picture.
- Performance can vary depending on so many different factors that a difference of a few percentage points might not be a decisive victory. The results might shift the opposite way when you test in a different environment.
- Certain MySQL features help or do not help performance depending on the workload. For completeness, always test performance with those features turned on and turned off. The two most important features to try with each workload are the [MySQL query cache](#), and the [adaptive hash index](#) for [InnoDB](#) tables.

This section progresses from simple and direct measurement techniques that a single developer can do, to more complicated ones that require additional expertise to perform and interpret the results.

8.12.1. Measuring the Speed of Expressions and Functions

To measure the speed of a specific MySQL expression or function, invoke the `BENCHMARK()` [1190] function using the `mysql` client program. Its syntax is `BENCHMARK(loop_count,expression)` [1190]. The return value is always zero, but `mysql` prints a line displaying approximately how long the statement took to execute. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+-----+
| BENCHMARK(1000000,1+1) |
+-----+
|                          0 |
+-----+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

The built-in MySQL functions are typically highly optimized, but there may be some exceptions. `BENCHMARK()` [1190] is an excellent tool for finding out if some function is a problem for your queries.

8.12.2. The MySQL Benchmark Suite

This benchmark suite is meant to tell any user what operations a given SQL implementation performs well or poorly. You can get a good idea for how the benchmarks work by looking at the code and results in the `sql-bench` directory in any MySQL source distribution.

Note that this benchmark is single-threaded, so it measures the minimum time for the operations performed. We plan to add multi-threaded tests to the benchmark suite in the future.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions. You can either download a released distribution from <http://dev.mysql.com/downloads/>, or use the current development source tree. (See Section 2.10.3, “Installing MySQL from a Development Source Tree”.)
- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You also need the server-specific DBD drivers for each of the servers you want to test. For example, to test MySQL, PostgreSQL, and DB2, you must have the `DBD::mysql`, `DBD::Pg`, and `DBD::DB2` modules installed. See Section 2.14, “Perl Installation Notes”.

After you obtain a MySQL source distribution, you can find the benchmark suite located in its `sql-bench` directory. To run the benchmark tests, build MySQL, and then change location into the `sql-bench` directory and execute the `run-all-tests` script:

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

`server_name` should be the name of one of the supported servers. To get a list of all options and supported servers, invoke this command:

```
shell> perl run-all-tests --help
```

The `crash-me` script also is located in the `sql-bench` directory. `crash-me` tries to determine what features a database system supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What data types are supported.
- How many indexes are supported.
- What functions are supported.
- How big a query can be.
- How big a `VARCHAR` column can be.

For more information about benchmark results, visit <http://www.mysql.com/why-mysql/benchmarks/>.

8.12.3. Using Your Own Benchmarks

Benchmark your application and database to find out where the bottlenecks are. After fixing one bottleneck (or by replacing it with a “dummy” module), you can proceed to identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck and decide how to solve it if someday you really need the extra performance.

For examples of portable benchmark programs, look at those in the MySQL benchmark suite. See [Section 8.12.2, “The MySQL Benchmark Suite”](#). You can take any program from this suite and modify it for your own needs. By doing this, you can try different solutions to your problem and test which really is fastest for you.

Another free benchmark suite is the Open Source Database Benchmark, available at <http://osdb.sourceforge.net/>.

It is very common for a problem to occur only when the system is very heavily loaded, even if other aspects of the system were tested extensively. These performance problems are typically due to issues of basic database design (for example, table scans are not good under high load) or problems with the operating system or libraries. These problems are much easier to fix if isolated during pre-production testing.

To avoid problems like this, benchmark your whole application under the worst possible load:

- The `mysqlslap` program can be helpful for simulating a high load produced by multiple clients issuing queries simultaneously. See [Section 4.5.7, “mysqlslap — Load Emulation Client”](#).
- You can also try benchmarking packages such as SysBench and DBT2, available at <http://sourceforge.net/projects/sysbench/>, and <http://osldb.sourceforge.net/#dbt2>.

These programs or packages can bring a system to its knees, so be sure to use them only on your development systems.

8.12.4. Measuring Performance with `performance_schema`

You can query the tables in the `performance_schema` database to see real-time information about the performance characteristics of your server and the applications it is running. See [Chapter 21, MySQL Performance Schema](#) for details.

8.12.5. Examining Thread Information

As you monitor the performance of your MySQL server, examine the process list, which is the set of threads currently executing within the server. Process list information is available from these sources:

- The `SHOW [FULL] PROCESSLIST` statement: [Section 13.7.5.30, “SHOW PROCESSLIST Syntax”](#)
- The `SHOW PROFILE` statement: [Section 13.7.5.32, “SHOW PROFILES Syntax”](#)

- The `INFORMATION_SCHEMA.PROCESSLIST` table: [Section 20.14, “The INFORMATION_SCHEMA PROCESSLIST Table”](#)
- The `mysqladmin processlist` command: [Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”](#)

You can always view information about your own threads. To view information about threads being executed for other accounts, you must have the `PROCESS [696]` privilege.

Each process list entry contains several pieces of information:

- `Id` is the connection identifier for the client associated with the thread.
- `User` and `Host` indicate the account associated with the thread.
- `db` is the default database for the thread, or `NULL` if none is selected.
- `Command` and `State` indicate what the thread is doing.

Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

- `Time` indicates how long the thread has been in its current state. The thread's notion of the current time may be altered in some cases: The thread can change the time with `SET TIMESTAMP = value`. For a thread running on a slave that is processing events from the master, the thread time is set to the time found in the events and thus reflects current time on the master and not the slave.
- `Info` contains the text of the statement being executed by the thread, or `NULL` if it is not executing one. By default, this value contains only the first 100 characters of the statement. To see the complete statements, use `SHOW FULL PROCESSLIST`.

The following sections list the possible `Command` values, and `State` values grouped by category. The meaning for some of these values is self-evident. For others, additional description is provided.

8.12.5.1. Thread Command Values

A thread can have any of the following `Command` values:

- `Binlog Dump`
This is a thread on a master server for sending binary log contents to a slave server.
- `Change user`
The thread is executing a change-user operation.
- `Close stmt`
The thread is closing a prepared statement.
- `Connect`
A replication slave is connected to its master.
- `Connect Out`
A replication slave is connecting to its master.
- `Create DB`

The thread is executing a create-database operation.

- `Daemon`

This thread is internal to the server, not a thread that services a client connection.

- `Debug`

The thread is generating debugging information.

- `Delayed insert`

The thread is a delayed-insert handler.

- `Drop DB`

The thread is executing a drop-database operation.

- `Error`

- `Execute`

The thread is executing a prepared statement.

- `Fetch`

The thread is fetching the results from executing a prepared statement.

- `Field List`

The thread is retrieving information for table columns.

- `Init DB`

The thread is selecting a default database.

- `Kill`

The thread is killing another thread.

- `Long Data`

The thread is retrieving long data in the result of executing a prepared statement.

- `Ping`

The thread is handling a server-ping request.

- `Prepare`

The thread is preparing a prepared statement.

- `Processlist`

The thread is producing information about server threads.

- `Query`

The thread is executing a statement.

- `Quit`
The thread is terminating.
- `Refresh`
The thread is flushing table, logs, or caches, or resetting status variable or replication server information.
- `Register Slave`
The thread is registering a slave server.
- `Reset stmt`
The thread is resetting a prepared statement.
- `Set option`
The thread is setting or resetting a client statement-execution option.
- `Shutdown`
The thread is shutting down the server.
- `Sleep`
The thread is waiting for the client to send a new statement to it.
- `Statistics`
The thread is producing server-status information.
- `Table Dump`
The thread is sending table contents to a slave server.
- `Time`
Unused.

8.12.5.2. General Thread States

The following list describes thread `State` values that are associated with general query processing and not more specialized activities such as replication. Many of these are useful only for finding bugs in the server.

- `After create`
This occurs when the thread creates a table (including internal temporary tables), at the end of the function that creates the table. This state is used even if the table could not be created due to some error.
- `Analyzing`
The thread is calculating a `MyISAM` table key distributions (for example, for `ANALYZE TABLE`).
- `checking permissions`
The thread is checking whether the server has the required privileges to execute the statement.

- `Checking table`

The thread is performing a table check operation.

- `cleaning up`

The thread has processed one command and is preparing to free memory and reset certain state variables.

- `closing tables`

The thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, verify that you do not have a full disk and that the disk is not in very heavy use.

- `converting HEAP to MyISAM`

The thread is converting an internal temporary table from a `MEMORY` table to an on-disk `MyISAM` table.

- `copy to tmp table`

The thread is processing an `ALTER TABLE` statement. This state occurs after the table with the new structure has been created but before rows are copied into it.

- `Copying to group table`

If a statement has different `ORDER BY` and `GROUP BY` criteria, the rows are sorted by group and copied to a temporary table.

- `Copying to tmp table`

The server is copying to a temporary table in memory.

- `Copying to tmp table on disk`

The server is copying to a temporary table on disk. The temporary result set has become too large (see [Section 8.4.3.3, "How MySQL Uses Internal Temporary Tables"](#)). Consequently, the thread is changing the temporary table from in-memory to disk-based format to save memory.

- `Creating index`

The thread is processing `ALTER TABLE ... ENABLE KEYS` for a `MyISAM` table.

- `Creating sort index`

The thread is processing a `SELECT` that is resolved using an internal temporary table.

- `creating table`

The thread is creating a table. This includes creation of temporary tables.

- `Creating tmp table`

The thread is creating a temporary table in memory or on disk. If the table is created in memory but later is converted to an on-disk table, the state during that operation will be `Copying to tmp table on disk`.

- `deleting from main table`

The server is executing the first part of a multiple-table delete. It is deleting only from the first table, and saving columns and offsets to be used for deleting from the other (reference) tables.

- `deleting from reference tables`

The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- `discard_or_import_tablespace`

The thread is processing an `ALTER TABLE ... DISCARD TABLESPACE` or `ALTER TABLE ... IMPORT TABLESPACE` statement.

- `end`

This occurs at the end but before the cleanup of `ALTER TABLE`, `CREATE VIEW`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements.

- `executing`

The thread has begun executing a statement.

- `Execution of init_command`

The thread is executing statements in the value of the `init_command` system variable.

- `freeing items`

The thread has executed a command. Some freeing of items done during this state involves the query cache. This state is usually followed by `cleaning up`.

- `Flushing tables`

The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables.

- `FULLTEXT initialization`

The server is preparing to perform a natural-language full-text search.

- `init`

This occurs before the initialization of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements. Actions taken by the server in this state include flushing the binary log, the `InnoDB` log, and some query cache cleanup operations.

For the `end` state, the following operations could be happening:

- Removing query cache entries after data in a table is changed
- Writing an event to the binary log
- Freeing memory buffers, including for blobs

- `Killed`

Someone has sent a `KILL` statement to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- `Locked`

The query is locked by another query.

As of MySQL 5.5.3, this state was removed because it was equivalent to the `Table lock` state and no longer appears in `SHOW PROCESSLIST` output.

- `logging slow query`

The thread is writing a statement to the slow-query log.

- `NULL`

This state is used for the `SHOW PROCESSLIST` state.

- `login`

The initial state for a connection thread until the client has been authenticated successfully.

- `manage keys`

The server is enabling or disabling a table index.

- `Opening tables, Opening table`

The thread is trying to open a table. This is should be very fast procedure, unless something prevents opening. For example, an `ALTER TABLE` or a `LOCK TABLE` statement can prevent opening a table until the statement is finished. It is also worth checking that your `table_open_cache` [561] value is large enough.

- `optimizing`

The server is performing initial optimizations for a query.

- `preparing`

This state occurs during query optimization.

- `Purging old relay logs`

The thread is removing unneeded relay log files.

- `query end`

This state occurs after processing a query but before the `freeing items` state.

- `Reading from net`

The server is reading a packet from the network.

- `Removing duplicates`

The query was using `SELECT DISTINCT` in such a way that MySQL could not optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- `removing tmp table`

The thread is removing an internal temporary table after processing a `SELECT` statement. This state is not used if no temporary table was created.

- `rename`

The thread is renaming a table.

- `rename result table`

The thread is processing an `ALTER TABLE` statement, has created the new table, and is renaming it to replace the original table.

- `Reopen tables`

The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- `Repair by sorting`

The repair code is using a sort to create indexes.

- `Repair done`

The thread has completed a multi-threaded repair for a `MyISAM` table.

- `Repair with keycache`

The repair code is using creating keys one by one through the key cache. This is much slower than `Repair by sorting`.

- `Rolling back`

The thread is rolling back a transaction.

- `Saving state`

For `MyISAM` table operations such as repair or analysis, the thread is saving the new table state to the `.MYI` file header. State includes information such as number of rows, the `AUTO_INCREMENT` counter, and key distributions.

- `Searching rows for update`

The thread is doing a first phase to find all matching rows before updating them. This has to be done if the `UPDATE` is changing the index that is used to find the involved rows.

- `Sending data`

The thread is reading and processing rows for a `SELECT` statement, and sending data to the client. Because operations occurring during this state tend to perform large amounts of disk access (reads), it is often the longest-running state over the lifetime of a given query.

- `setup`

The thread is beginning an `ALTER TABLE` operation.

- `Sorting for group`

The thread is doing a sort to satisfy a `GROUP BY`.

- `Sorting for order`

The thread is doing a sort to satisfy a `ORDER BY`.

- `Sorting index`

The thread is sorting index pages for more efficient access during a `MyISAM` table optimization operation.

- `Sorting result`

For a `SELECT` statement, this is similar to `Creating sort index`, but for nontemporary tables.

- `statistics`

The server is calculating statistics to develop a query execution plan. If a thread is in this state for a long time, the server is probably disk-bound performing other work.

- `System lock`

The thread is going to request or is waiting for an internal or external system lock for the table. If this state is being caused by requests for external locks and you are not using multiple `mysqld` servers that are accessing the same `MyISAM` tables, you can disable external system locks with the `--skip-external-locking` [460] option. However, external locking is disabled by default, so it is likely that this option will have no effect. For `SHOW PROFILE`, this state means the thread is requesting the lock (not waiting for it).

- `Table lock`

The next thread state after `System lock`. The thread has acquired an external lock and is going to request an internal table lock.

This state was replaced in MySQL 5.5.6 with `waiting for table level lock`.

- `Updating`

The thread is searching for rows to update and is updating them.

- `updating main table`

The server is executing the first part of a multiple-table update. It is updating only the first table, and saving columns and offsets to be used for updating the other (reference) tables.

- `updating reference tables`

The server is executing the second part of a multiple-table update and updating the matched rows from the other tables.

- `User lock`

The thread is going to request or is waiting for an advisory lock requested with a `GET_LOCK()` [1199] call. For `SHOW PROFILE`, this state means the thread is requesting the lock (not waiting for it).

- `User sleep`

The thread has invoked a `SLEEP()` [1201] call.

- `Waiting for all running commits to finish`

A statement that causes an explicit or implicit commit is waiting for release of a read lock. This state was removed in MySQL 5.5.8; `Waiting for commit lock` is used instead.

- `Waiting for commit lock`

A statement that causes an explicit or implicit commit is waiting for release of a read lock or `FLUSH TABLES WITH READ LOCK`) is waiting for a commit lock. This state was added in MySQL 5.5.8.

- `Waiting for global read lock`

`FLUSH TABLES WITH READ LOCK`) is waiting for a global read lock.

- `Waiting for release of readlock`

The thread is waiting for a global read lock obtained by another thread (with `FLUSH TABLES WITH READ LOCK`) to be released. This state was removed in MySQL 5.5.8; `Waiting for global read lock` or `Waiting for commit lock` are used instead.

- `Waiting for tables`, `Waiting for table`, `Waiting for table flush`

The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

This notification takes place if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES tbl_name`, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

In MySQL 5.5.6, `Waiting for table` was replaced with `Waiting for table flush`.

- `Waiting for lock_type lock`

The server is waiting to acquire a lock, where `lock_type` indicates the type of lock:

- `Waiting for event metadata lock` (added in MySQL 5.5.8)
- `Waiting for global metadata lock` (replaced by `Waiting for global read lock` in MySQL 5.5.8)
- `Waiting for global read lock` (added in MySQL 5.5.8)
- `Waiting for schema metadata lock`
- `Waiting for stored function metadata lock`
- `Waiting for stored procedure metadata lock`
- `Waiting for table level lock`
- `Waiting for table metadata lock`
- `Waiting for trigger metadata lock` (added in MySQL 5.5.8)
- `Waiting on cond`

A generic state in which the thread is waiting for a condition to become true. No specific state information is available.

- `Waiting to get readlock`

The thread has issued a `FLUSH TABLES WITH READ LOCK` statement to obtain a global read lock and is waiting to obtain the lock. This state was removed in MySQL 5.5.8; `Waiting for global read lock` is used instead.

- `Writing to net`

The server is writing a packet to the network.

8.12.5.3. Delayed-Insert Thread States

These thread states are associated with processing for `DELAYED` inserts (see [Section 13.2.5.2](#), “`INSERT DELAYED Syntax`”). Some states are associated with connection threads that process `INSERT DELAYED` statements from clients. Other states are associated with delayed-insert handler threads that insert the rows. There is a delayed-insert handler thread for each table for which `INSERT DELAYED` statements are issued.

States associated with a connection thread that processes an `INSERT DELAYED` statement from the client:

- `allocating local table`

The thread is preparing to feed rows to the delayed-insert handler thread.

- `Creating delayed handler`

The thread is creating a handler for `DELAYED` inserts.

- `got handler lock`

This occurs before the `allocating local table` state and after the `waiting for handler lock` state, when the connection thread gets access to the delayed-insert handler thread.

- `got old table`

This occurs after the `waiting for handler open` state. The delayed-insert handler thread has signaled that it has ended its initialization phase, which includes opening the table for delayed inserts.

- `storing row into queue`

The thread is adding a new row to the list of rows that the delayed-insert handler thread must insert.

- `update`

The thread is getting ready to start updating the table.

- `waiting for delay_list`

This occurs during the initialization phase when the thread is trying to find the delayed-insert handler thread for the table, and before attempting to gain access to the list of delayed-insert threads.

- `waiting for handler insert`

An `INSERT DELAYED` handler has processed all pending inserts and is waiting for new ones.

- `waiting for handler lock`

This occurs before the `allocating local table` state when the connection thread waits for access to the delayed-insert handler thread.

- `waiting for handler open`

This occurs after the `Creating delayed handler` state and before the `got old table` state. The delayed-insert handler thread has just been started, and the connection thread is waiting for it to initialize.

States associated with a delayed-insert handler thread that inserts the rows:

- `insert`

The state that occurs just before inserting rows into the table.

- `reschedule`

After inserting a number of rows, the delayed-insert thread sleeps to let other threads do work.

- `upgrading lock`

A delayed-insert handler is trying to get a lock for the table to insert rows.

- `Waiting for INSERT`

A delayed-insert handler is waiting for a connection thread to add rows to the queue (see `storing row into queue`).

8.12.5.4. Query Cache Thread States

These thread states are associated with the query cache (see [Section 8.9.3, “The MySQL Query Cache”](#)).

- `checking privileges on cached query`

The server is checking whether the user has privileges to access a cached query result.

- `checking query cache for query`

The server is checking whether the current query is present in the query cache.

- `invalidating query cache entries`

Query cache entries are being marked invalid because the underlying tables have changed.

- `sending cached result to client`

The server is taking the result of a query from the query cache and sending it to the client.

- `storing result in query cache`

The server is storing the result of a query in the query cache.

- `Waiting for query cache lock`

This state occurs while a session is waiting to take the query cache lock. This can happen for any statement that needs to perform some query cache operation, such as an `INSERT` or `DELETE` that invalidates the query cache, a `SELECT` that looks for a cached entry, `RESET QUERY CACHE`, and so forth.

8.12.5.5. Replication Master Thread States

The following list shows the most common states you may see in the `State` column for the master's `Binlog Dump` thread. If you see no `Binlog Dump` threads on a master server, this means that replication is not running—that is, that no slaves are currently connected.

- `Sending binlog event to slave`

Binary logs consist of *events*, where an event is usually an update plus some other information. The thread has read an event from the binary log and is now sending it to the slave.

- `Finished reading one binlog; switching to next binlog`

The thread has finished reading a binary log file and is opening the next one to send to the slave.

- `Master has sent all binlog to slave; waiting for binlog to be updated`

The thread has read all outstanding updates from the binary logs and sent them to the slave. The thread is now idle, waiting for new events to appear in the binary log resulting from new updates occurring on the master.

- `Waiting to finalize termination`

A very brief state that occurs as the thread is stopping.

8.12.5.6. Replication Slave I/O Thread States

The following list shows the most common states you see in the `State` column for a slave server I/O thread. This state also appears in the `Slave_IO_State` column displayed by `SHOW SLAVE STATUS`, so you can get a good view of what is happening by using that statement.

- `Waiting for master update`

The initial state before `Connecting to master`.

- `Connecting to master`

The thread is attempting to connect to the master.

- `Checking master version`

A state that occurs very briefly, after the connection to the master is established.

- `Registering slave on master`

A state that occurs very briefly after the connection to the master is established.

- `Requesting binlog dump`

A state that occurs very briefly, after the connection to the master is established. The thread sends to the master a request for the contents of its binary logs, starting from the requested binary log file name and position.

- `Waiting to reconnect after a failed binlog dump request`

If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `CHANGE MASTER TO` statement.

- `Reconnecting after a failed binlog dump request`

The thread is trying to reconnect to the master.

- `Waiting for master to send event`

The thread has connected to the master and is waiting for binary log events to arrive. This can last for a long time if the master is idle. If the wait lasts for `slave_net_timeout` [1888] seconds, a timeout occurs. At that point, the thread considers the connection to be broken and makes an attempt to reconnect.

- `Queueing master event to the relay log`

The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- `Waiting to reconnect after a failed master event read`

An error occurred while reading (due to disconnection). The thread is sleeping for the number of seconds set by the `CHANGE MASTER TO` statement (default 60) before attempting to reconnect.

- `Reconnecting after a failed master event read`

The thread is trying to reconnect to the master. When connection is established again, the state becomes `Waiting for master to send event`.

- `Waiting for the slave SQL thread to free enough relay log space`

You are using a nonzero `relay_log_space_limit` [546] value, and the relay logs have grown large enough that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- `Waiting for slave mutex on exit`

A state that occurs briefly as the thread is stopping.

8.12.5.7. Replication Slave SQL Thread States

The following list shows the most common states you may see in the `State` column for a slave server SQL thread:

- `Waiting for the next event in relay log`

The initial state before `Reading event from the relay log`.

- `Reading event from the relay log`

The thread has read an event from the relay log so that the event can be processed.

- `Making temp file`

The thread is executing a `LOAD DATA INFILE` statement and is creating a temporary file containing the data from which the slave will read rows.

- `Slave has read all relay log; waiting for the slave I/O thread to update it`

The thread has processed all events in the relay log files, and is now waiting for the I/O thread to write new events to the relay log.

- `Waiting for slave mutex on exit`

A very brief state that occurs as the thread is stopping.

The `State` column for the I/O thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and is executing it.

8.12.5.8. Replication Slave Connection Thread States

These thread states occur on a replication slave but are associated with connection threads, not with the I/O or SQL threads.

- `Changing master`

The thread is processing a `CHANGE MASTER TO` statement.

- `Killing slave`

The thread is processing a `STOP SLAVE` statement.

- `Opening master dump table`

This state occurs after `Creating table from master dump`.

- `Reading master dump table data`

This state occurs after `Opening master dump table`.

- `Rebuilding the index on master dump table`

This state occurs after `Reading master dump table data`.

8.12.5.9. MySQL Cluster Thread States

- `Committing events to binlog`

- `Opening mysql.ndb_apply_status`

- `Processing events`

The thread is processing events for binary logging.

- `Processing events from schema table`

The thread is doing the work of schema replication.

- `Shutting down`

- `Syncing ndb table schema operation and binlog`

This is used to have a correct binary log of schema operations for NDB.

- `Waiting for event from ndbcluster`

The server is acting as an SQL node in a MySQL Cluster, and is connected to a cluster management node.

- `Waiting for first event from ndbcluster`

- `Waiting for ndbcluster binlog update to reach current position`

- `Waiting for ndbcluster to start`

- `Waiting for schema epoch`

The thread is waiting for a schema epoch (that is, a global checkpoint).

8.12.5.10. Event Scheduler Thread States

These states occur for the Event Scheduler thread, threads that are created to execute scheduled events, or threads that terminate the scheduler.

- `Clearing`
The scheduler thread or a thread that was executing an event is terminating and is about to end.
- `Initialized`
The scheduler thread or a thread that will execute an event has been initialized.
- `Waiting for next activation`
The scheduler has a nonempty event queue but the next activation is in the future.
- `Waiting for scheduler to stop`
The thread issued `SET GLOBAL event_scheduler=OFF` and is waiting for the scheduler to stop.
- `Waiting on empty queue`
The scheduler's event queue is empty and it is sleeping.

8.13. Internal Details of MySQL Optimizations

This background information helps you to understand some of the terms you see in the `EXPLAIN` plan output. If you are rewriting queries to make them more efficient, or writing your own application logic for lookups, joining, or sorting, use this information to determine which optimizations you write yourself and which you can rely on MySQL to perform.

8.13.1. Range Optimization

The `range` [838] access method uses a single index to retrieve a subset of table rows that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. The following sections give a detailed description of how intervals are extracted from the `WHERE` clause.

8.13.1.1. The Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the `WHERE` clause, so we speak of *range conditions* rather than “intervals.”

The definition of a range condition for a single-part index is as follows:

- For both `BTREE` and `HASH` indexes, comparison of a key part with a constant value is a range condition when using the `=` [1089], `<=>` [1090], `IN()` [1092], `IS NULL` [1091], or `IS NOT NULL` [1091] operators.
- Additionally, for `BTREE` indexes, comparison of a key part with a constant value is a range condition when using the `>` [1090], `<` [1090], `>=` [1090], `<=` [1090], `BETWEEN` [1091], `!=` [1090], or `<>` [1090] operators, or `LIKE` [1111] comparisons if the argument to `LIKE` [1111] is a constant string that does not start with a wildcard character.
- For all types of indexes, multiple range conditions combined with `OR` [1095] or `AND` [1094] form a range condition.

“Constant value” in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a `const` [837] or `system` [837] table from the same join

- The result of an uncorrelated subquery
- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the `WHERE` clause:

```
SELECT * FROM t1
WHERE key_col > 1
AND key_col < 10;

SELECT * FROM t1
WHERE key_col = 1
OR key_col IN (15,18,20);

SELECT * FROM t1
WHERE key_col LIKE 'ab%'
OR key_col BETWEEN 'bar' AND 'foo';
```

Note that some nonconstant values may be converted to constants during the constant propagation phase.

MySQL tries to extract range conditions from the `WHERE` clause for each of the possible indexes. During the extraction process, conditions that cannot be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

Consider the following statement, where `key1` is an indexed column and `nonkey` is not indexed:

```
SELECT * FROM t1 WHERE
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z');
```

The extraction process for key `key1` is as follows:

1. Start with original `WHERE` clause:

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z')
```

2. Remove `nonkey = 4` and `key1 LIKE '%b'` because they cannot be used for a range scan. The correct way to remove them is to replace them with `TRUE`, so that we do not miss any matching rows when doing the range scan. Having replaced them with `TRUE`, we get:

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
(key1 < 'bar' AND TRUE) OR
(key1 < 'uux' AND key1 > 'z')
```

3. Collapse conditions that are always true or false:

- `(key1 LIKE 'abcde%' OR TRUE)` is always true
- `(key1 < 'uux' AND key1 > 'z')` is always false

Replacing these conditions with constants, we get:

```
(key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
```

Removing unnecessary `TRUE` and `FALSE` constants, we obtain:

```
(key1 < 'abc') OR (key1 < 'bar')
```

4. Combining overlapping intervals into one yields the final condition to be used for the range scan:

```
(key1 < 'bar')
```

In general (and as demonstrated by the preceding example), the condition used for a range scan is less restrictive than the `WHERE` clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full `WHERE` clause.

The range condition extraction algorithm can handle nested `AND` [1094]/`OR` [1095] constructs of arbitrary depth, and its output does not depend on the order in which conditions appear in `WHERE` clause.

Currently, MySQL does not support merging multiple ranges for the `range` [838] access method for spatial indexes. To work around this limitation, you can use a `UNION` with identical `SELECT` statements, except that you put each spatial predicate in a different `SELECT`.

8.13.1.2. The Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index rows to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as `key1(key_part1, key_part2, key_part3)`, and the following set of key tuples listed in key order:

<i>key_part1</i>	<i>key_part2</i>	<i>key_part3</i>
NULL	1	'abc'
NULL	1	'xyz'
NULL	2	'foo'
1	1	'abc'
1	1	'xyz'
1	2	'abc'
2	1	'aaa'

The condition `key_part1 = 1` defines this interval:

```
(1,-inf,-inf) <= (key_part1,key_part2,key_part3) < (1,+inf,+inf)
```

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition `key_part3 = 'abc'` does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For `HASH` indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

```
key_part1 cmp const1
AND key_part2 cmp const2
AND ...
AND key_partN cmp constN;
```

Here, `const1`, `const2`, ... are constants, `cmp` is one of the `=` [1089], `<=>` [1090], or `IS NULL` [1091] comparison operators, and the conditions cover all index parts. (That is, there are `N` conditions, one for each part of an `N`-part index.) For example, the following is a range condition for a three-part `HASH` index:

```
key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
```

For the definition of what is considered to be a constant, see [Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”](#).

- For a `BTREE` index, an interval might be usable for conditions combined with `AND` [1094], where each condition compares a key part with a constant value using `=` [1089], `<=>` [1090], `IS NULL` [1091], `>` [1090], `<` [1090], `>=` [1090], `<=` [1090], `!=` [1090], `<>` [1090], `BETWEEN` [1091], or `LIKE` `'pattern'` [1111] (where `'pattern'` does not start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all rows that match the condition (or two intervals if `<>` [1090] or `!=` [1090] is used).

The optimizer attempts to use additional key parts to determine the interval as long as the comparison operator is `=` [1089], `<=>` [1090], or `IS NULL` [1091]. If the operator is `>` [1090], `<` [1090], `>=` [1090], `<=` [1090], `!=` [1090], `<>` [1090], `BETWEEN` [1091], or `LIKE` [1111], the optimizer uses it but considers no more key parts. For the following expression, the optimizer uses `=` [1089] from the first comparison. It also uses `>=` [1090] from the second comparison but considers no further key parts and does not use the third comparison for interval construction:

```
key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
```

The single interval is:

```
('foo',10,-inf) < (key_part1,key_part2,key_part3) < ('foo',+inf,+inf)
```

It is possible that the created interval contains more rows than the initial condition. For example, the preceding interval includes the value `('foo', 11, 0)`, which does not satisfy the original condition.

- If conditions that cover sets of rows contained within intervals are combined with `OR` [1095], they form a condition that covers a set of rows contained within the union of their intervals. If the conditions are combined with `AND` [1094], they form a condition that covers a set of rows contained within the intersection of their intervals. For example, for this condition on a two-part index:

```
(key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
```

The intervals are:

```
(1,-inf) < (key_part1,key_part2) < (1,2)
(5,-inf) < (key_part1,key_part2)
```

In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The `key_len` column in the `EXPLAIN` output indicates the maximum length of the key prefix used.

In some cases, `key_len` may indicate that a key part was used, but that might be not what you would expect. Suppose that `key_part1` and `key_part2` can be `NULL`. Then the `key_len` column displays two key part lengths for the following condition:

```
key_part1 >= 1 AND key_part2 < 2
```

But, in fact, the condition is converted to this:

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

[Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”](#), describes how optimizations are performed to combine or eliminate intervals for range conditions on a single-part index. Analogous steps are performed for range conditions on multiple-part indexes.

8.13.2. Index Merge Optimization

The *Index Merge* method is used to retrieve rows with several [range \[838\]](#) scans and to merge their results into one. The merge can produce unions, intersections, or unions-of-intersections of its underlying scans. This access method merges index scans from a single table; it does not merge scans across multiple tables.

In `EXPLAIN` output, the Index Merge method appears as `index_merge [838]` in the `type` column. In this case, the `key` column contains a list of indexes used, and `key_len` contains a list of the longest key parts for those indexes.

Examples:

```
SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;

SELECT * FROM tbl_name
  WHERE (key1 = 10 OR key2 = 20) AND non_key=30;

SELECT * FROM t1, t2
  WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
  AND t2.key1=t1.some_col;

SELECT * FROM t1, t2
  WHERE t1.key1=1
  AND (t2.key1=t1.some_col OR t2.key2=t1.some_col2);
```

The Index Merge method has several access algorithms (seen in the `Extra` field of `EXPLAIN` output):

- Using `intersect(...)`
- Using `union(...)`
- Using `sort_union(...)`

The following sections describe these methods in greater detail.



Note

The Index Merge optimization algorithm has the following known deficiencies:

- If your query has a complex `WHERE` clause with deep `AND [1094]`/`OR [1095]` nesting and MySQL doesn't choose the optimal plan, try distributing terms using the following identity laws:

```
(x AND y) OR z = (x OR z) AND (y OR z)
(x OR y) AND z = (x AND z) OR (y AND z)
```

- Index Merge is not applicable to full-text indexes. We plan to extend it to cover these in a future MySQL release.
- If a range scan is possible on some key, the optimizer will not consider using Index Merge Union or Index Merge Sort-Union algorithms. For example, consider this query:

```
SELECT * FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

For this query, two plans are possible:

- An Index Merge scan using the `(goodkey1 < 10 OR goodkey2 < 20)` condition.
- A range scan using the `badkey < 30` condition.

However, the optimizer considers only the second plan.

The choice between different possible variants of the Index Merge access method and other access methods is based on cost estimates of various available options.

8.13.2.1. The Index Merge Intersection Access Algorithm

This access algorithm can be employed when a `WHERE` clause was converted to several range conditions on different keys combined with `AND` [1094], and each condition is one of the following:

- In this form, where the index has exactly N parts (that is, all index parts are covered):

```
key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
```

- Any range condition over a primary key of an `InnoDB` table.

Examples:

```
SELECT * FROM innodb_table WHERE primary_key < 10 AND key_coll=20;
```

```
SELECT * FROM tbl_name
WHERE (key1_part1=1 AND key1_part2=2) AND key2=2;
```

The Index Merge intersection algorithm performs simultaneous scans on all used indexes and produces the intersection of row sequences that it receives from the merged index scans.

If all columns used in the query are covered by the used indexes, full table rows are not retrieved (`EXPLAIN` output contains `Using index` in `Extra` field in this case). Here is an example of such a query:

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

If the used indexes don't cover all columns used in the query, full rows are retrieved only when the range conditions for all used keys are satisfied.

If one of the merged conditions is a condition over a primary key of an `InnoDB` table, it is not used for row retrieval, but is used to filter out rows retrieved using other conditions.

8.13.2.2. The Index Merge Union Access Algorithm

The applicability criteria for this algorithm are similar to those for the Index Merge method intersection algorithm. The algorithm can be employed when the table's `WHERE` clause was converted to several range conditions on different keys combined with `OR` [1095], and each condition is one of the following:

- In this form, where the index has exactly N parts (that is, all index parts are covered):

```
key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
```

- Any range condition over a primary key of an `InnoDB` table.
- A condition for which the Index Merge method intersection algorithm is applicable.

Examples:

```
SELECT * FROM t1 WHERE key1=1 OR key2=2 OR key3=3;
```

```
SELECT * FROM innodb_table WHERE (key1=1 AND key2=2) OR
(key3='foo' AND key4='bar') AND key5=5;
```

8.13.2.3. The Index Merge Sort-Union Access Algorithm

This access algorithm is employed when the `WHERE` clause was converted to several range conditions combined by `OR` [1095], but for which the Index Merge method union algorithm is not applicable.

Examples:

```
SELECT * FROM tbl_name WHERE key_col1 < 10 OR key_col2 < 20;

SELECT * FROM tbl_name
WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col=30;
```

The difference between the sort-union algorithm and the union algorithm is that the sort-union algorithm must first fetch row IDs for all rows and sort them before returning any rows.

8.13.3. Engine Condition Pushdown Optimization

This optimization improves the efficiency of direct comparisons between a nonindexed column and a constant. In such cases, the condition is “pushed down” to the storage engine for evaluation. This optimization can be used only by the `NDBCLUSTER` storage engine.

For MySQL Cluster, this optimization can eliminate the need to send nonmatching rows over the network between the cluster's data nodes and the MySQL Server that issued the query, and can speed up queries where it is used by a factor of 5 to 10 times over cases where condition pushdown could be but is not used.

Suppose that a MySQL Cluster table is defined as follows:

```
CREATE TABLE t1 (
  a INT,
  b INT,
  KEY(a)
) ENGINE=NDBCLUSTER;
```

Condition pushdown can be used with queries such as the one shown here, which includes a comparison between a nonindexed column and a constant:

```
SELECT a, b FROM t1 WHERE b = 10;
```

The use of condition pushdown can be seen in the output of `EXPLAIN`:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: t1
       type: ALL
possible_keys: NULL
         key: NULL
      key_len: NULL
         ref: NULL
        rows: 10
   Extra: Using where with pushed condition
```

However, condition pushdown *cannot* be used with either of these two queries:

```
SELECT a,b FROM t1 WHERE a = 10;
SELECT a,b FROM t1 WHERE b + 1 = 10;
```

Condition pushdown is not applicable to the first query because an index exists on column `a`. (An index access method would be more efficient and so would be chosen in preference to condition pushdown.) Condition pushdown cannot be employed for the second query because the comparison involving the

nonindexed column `b` is indirect. (However, condition pushdown could be applied if you were to reduce `b + 1 = 10` to `b = 9` in the `WHERE` clause.)

Condition pushdown may also be employed when an indexed column is compared with a constant using a `>` or `<` operator:

```
mysql> EXPLAIN SELECT a, b FROM t1 WHERE a < 2\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: t1
      type: range
possible_keys: a
      key: a
      key_len: 5
      ref: NULL
      rows: 2
      Extra: Using where with pushed condition
```

Other supported comparisons for condition pushdown include the following:

- `column [NOT] LIKE pattern`
pattern must be a string literal containing the pattern to be matched; for syntax, see [Section 12.5.1, “String Comparison Functions”](#).
- `column IS [NOT] NULL`
- `column IN (value_list)`
 Each item in the *value_list* must be a constant, literal value.
- `column BETWEEN constant1 AND constant2`
constant1 and *constant2* must each be a constant, literal value.

In all of the cases in the preceding list, it is possible for the condition to be converted into the form of one or more direct comparisons between a column and a constant.

Engine condition pushdown is enabled by default. To disable it at server startup, set the `optimizer_switch` [536] system variable. For example, in a `my.cnf` file, use these lines:

```
[mysqld]
optimizer_switch=engine_condition_pushdown=off
```

At runtime, disable condition pushdown like this:

```
SET optimizer_switch='engine_condition_pushdown=off';
```

Before MySQL 5.5.3, disable condition pushdown using the `engine_condition_pushdown` [497] system variable. At server startup:

```
[mysqld]
engine_condition_pushdown=0
```

At runtime, use either of these statements:

```
SET engine_condition_pushdown=OFF;
```

```
SET engine_condition_pushdown=0;
```

Limitations. Condition pushdown is subject to the following limitations:

- Condition pushdown is supported only by the `NDBCLUSTER` storage engine.
- Columns may be compared with constants only; however, this includes expressions which evaluate to constant values.
- Columns used in comparisons cannot be of any of the `BLOB` or `TEXT` types.
- A string value to be compared with a column must use the same collation as the column.
- Joins are not directly supported; conditions involving multiple tables are pushed separately where possible. Use `EXPLAIN EXTENDED` to determine which conditions are actually pushed down.

8.13.4. IS NULL Optimization

MySQL can perform the same optimization on `col_name IS NULL [1091]` that it can use for `col_name = constant_value`. For example, MySQL can use indexes and ranges to search for `NULL` with `IS NULL [1091]`.

Examples:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a `WHERE` clause includes a `col_name IS NULL [1091]` condition for a column that is declared as `NOT NULL`, that expression is optimized away. This optimization does not occur in cases when the column might produce `NULL` anyway; for example, if it comes from a table on the right side of a `LEFT JOIN`.

MySQL can also optimize the combination `col_name = expr OR col_name IS NULL`, a form that is common in resolved subqueries. `EXPLAIN` shows `ref_or_null [838]` when this optimization is used.

This optimization can handle one `IS NULL [1091]` for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns `a` and `b` of table `t2`:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT * FROM t1, t2
WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1, t2
WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

`ref_or_null [838]` works by first doing a read on the reference key, and then a separate search for rows with a `NULL` key value.

Note that the optimization can handle only one `IS NULL [1091]` level. In the following query, MySQL uses key lookups only on the expression `(t1.a=t2.a AND t2.a IS NULL)` and is not able to use the key part on `b`:

```
SELECT * FROM t1, t2
```

```
WHERE (t1.a=t2.a AND t2.a IS NULL)
OR (t1.b=t2.b AND t2.b IS NULL);
```

8.13.5. LEFT JOIN and RIGHT JOIN Optimization

MySQL implements an `A LEFT JOIN B join_condition` as follows:

- Table `B` is set to depend on table `A` and all tables on which `A` depends.
- Table `A` is set to depend on all tables (except `B`) that are used in the `LEFT JOIN` condition.
- The `LEFT JOIN` condition is used to decide how to retrieve rows from table `B`. (In other words, any condition in the `WHERE` clause is not used.)
- All standard join optimizations are performed, with the exception that a table is always read after all tables on which it depends. If there is a circular dependence, MySQL issues an error.
- All standard `WHERE` optimizations are performed.
- If there is a row in `A` that matches the `WHERE` clause, but there is no row in `B` that matches the `ON` condition, an extra `B` row is generated with all columns set to `NULL`.
- If you use `LEFT JOIN` to find rows that do not exist in some table and you have the following test: `col_name IS NULL` in the `WHERE` part, where `col_name` is a column that is declared as `NOT NULL`, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the `LEFT JOIN` condition.

The implementation of `RIGHT JOIN` is analogous to that of `LEFT JOIN` with the roles of the tables reversed.

The join optimizer calculates the order in which tables should be joined. The table read order forced by `LEFT JOIN` or `STRAIGHT_JOIN` helps the join optimizer do its work much more quickly, because there are fewer table permutations to check. Note that this means that if you do a query of the following type, MySQL does a full scan on `b` because the `LEFT JOIN` forces it to be read before `d`:

```
SELECT *
FROM a JOIN b LEFT JOIN c ON (c.key=a.key)
LEFT JOIN d ON (d.key=a.key)
WHERE b.key=d.key;
```

The fix in this case is reverse the order in which `a` and `b` are listed in the `FROM` clause:

```
SELECT *
FROM b JOIN a LEFT JOIN c ON (c.key=a.key)
LEFT JOIN d ON (d.key=a.key)
WHERE b.key=d.key;
```

For a `LEFT JOIN`, if the `WHERE` condition is always false for the generated `NULL` row, the `LEFT JOIN` is changed to a normal join. For example, the `WHERE` clause would be false in the following query if `t2.column1` were `NULL`:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it is safe to convert the query to a normal join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

This can be made faster because MySQL can use table `t2` before table `t1` if doing so would result in a better query plan. To provide a hint about the table join order, use `STRAIGHT_JOIN`. (See [Section 13.2.9](#), “`SELECT Syntax`”.)

8.13.6. Nested-Loop Join Algorithms

MySQL executes joins between tables using a nested-loop algorithm or variations on it.

Nested-Loop Join Algorithm

A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join. This process is repeated as many times as there remain tables to be joined.

Assume that a join between three tables `t1`, `t2`, and `t3` is to be executed using the following join types:

Table	Join Type
t1	range
t2	ref
t3	ALL

If a simple NLJ algorithm is used, the join is processed like this:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions,
        send to client
    }
  }
}
```

Because the NLJ algorithm passes rows one at a time from outer loops to inner loops, it typically reads tables processed in the inner loops many times.

Block Nested-Loop Join Algorithm

A Block Nested-Loop (BNL) join algorithm uses buffering of rows read in outer loops to reduce the number of times that tables in inner loops must be read. For example, if 10 rows are read into a buffer and the buffer is passed to the next inner loop, each row read in the inner loop can be compared against all 10 rows in the buffer. This reduces the number of times the inner table must be read by an order of magnitude.

MySQL uses join buffering under these conditions:

- The `join_buffer_size` [507] system variable determines the size of each join buffer.
- Join buffering can be used when the join is of type `ALL` [839] or `index` [839] (in other words, when no possible keys can be used, and a full scan is done, of either the data or index rows, respectively), or `range` [838].
- One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.
- A join buffer is never allocated for the first nonconst table, even if it would be of type `ALL` [839] or `index` [839].
- A join buffer is allocated prior to executing the join and freed after the query is done.
- Only columns of interest to the join are stored in the join buffer, not whole rows.

For the example join described previously for the NLJ algorithm (without buffering), the join is done as follows using join buffering:

```

for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions,
            send to client
        }
      }
      empty buffer
    }
  }
}

if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions,
        send to client
    }
  }
}

```

If S is the size of each stored $t1, t2$ combination in the join buffer and C is the number of combinations in the buffer, the number of times table $t3$ is scanned is:

```
(S * C) / join_buffer_size + 1
```

The number of $t3$ scans decreases as the value of `join_buffer_size` [507] increases, up to the point when `join_buffer_size` [507] is large enough to hold all previous row combinations. At that point, there is no speed to be gained by making it larger.

8.13.7. Nested Join Optimization

The syntax for expressing joins permits nested joins. The following discussion refers to the join syntax described in [Section 13.2.9.2, “JOIN Syntax”](#).

The syntax of `table_factor` is extended in comparison with the SQL Standard. The latter accepts only `table_reference`, not a list of them inside a pair of parentheses. This is a conservative extension if we consider each comma in a list of `table_reference` items as equivalent to an inner join. For example:

```

SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
      ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

```

is equivalent to:

```

SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
      ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

```

In MySQL, `CROSS JOIN` is a syntactic equivalent to `INNER JOIN` (they can replace each other). In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause; `CROSS JOIN` is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. After removing parentheses and grouping operations to the left, the join expression:

```

t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
      ON t1.a=t2.a

```

transforms into the expression:

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
ON t2.b=t3.b OR t2.b IS NULL
```

Yet, the two expressions are not equivalent. To see this, suppose that the tables `t1`, `t2`, and `t3` have the following state:

- Table `t1` contains rows (1), (2)
- Table `t2` contains row (1,101)
- Table `t3` contains row (101)

In this case, the first expression returns a result set including the rows (1,1,101,101), (2,NULL,NULL,NULL), whereas the second expression returns the rows (1,1,101,101), (2,NULL,NULL,101):

```
mysql> SELECT *
-> FROM t1
-> LEFT JOIN
-> (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
-> ON t1.a=t2.a;
```

a	a	b	b
1	1	101	101
2	NULL	NULL	NULL

```
mysql> SELECT *
-> FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
-> LEFT JOIN t3
-> ON t2.b=t3.b OR t2.b IS NULL;
```

a	a	b	b
1	1	101	101
2	NULL	NULL	101

In the following example, an outer join operation is used together with an inner join operation:

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

That expression cannot be transformed into the following expression:

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3.
```

For the given table states, the two expressions return different sets of rows:

```
mysql> SELECT *
-> FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
```

a	a	b	b
1	1	101	101
2	NULL	NULL	NULL

```
mysql> SELECT *
-> FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
```

a	a	b	b
1	1	101	101

```
| 2 | NULL | NULL | 101 |
+-----+-----+-----+-----+
```

Therefore, if we omit parentheses in a join expression with outer join operators, we might change the result set for the original expression.

More exactly, we cannot ignore parentheses in the right operand of the left outer join operation and in the left operand of a right join operation. In other words, we cannot ignore parentheses for the inner table expressions of outer join operations. Parentheses for the other operand (operand for the outer table) can be ignored.

The following expression:

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

is equivalent to this expression:

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

for any tables `t1`, `t2`, `t3` and any condition `P` over attributes `t2.b` and `t3.b`.

Whenever the order of execution of the join operations in a join expression (*join_table*) is not from left to right, we talk about nested joins. Consider the following queries:

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
WHERE t1.a > 1
```

```
SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

Those queries are considered to contain these nested joins:

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

The nested join is formed in the first query with a left join operation, whereas in the second query it is formed with an inner join operation.

In the first query, the parentheses can be omitted: The grammatical structure of the join expression will dictate the same order of execution for join operations. For the second query, the parentheses cannot be omitted, although the join expression here can be interpreted unambiguously without them. (In our extended syntax the parentheses in `(t2, t3)` of the second query are required, although theoretically the query could be parsed without them: We still would have unambiguous syntactical structure for the query because `LEFT JOIN` and `ON` would play the role of the left and right delimiters for the expression `(t2,t3).`)

The preceding examples demonstrate these points:

- For join expressions involving only inner joins (and not outer joins), parentheses can be removed. You can remove parentheses and evaluate left to right (or, in fact, you can evaluate the tables in any order).
- The same is not true, in general, for outer joins or for outer joins mixed with inner joins. Removal of parentheses may change the result.

Queries with nested outer joins are executed in the same pipeline manner as queries with inner joins. More exactly, a variation of the nested-loop join algorithm is exploited. Recall by what algorithmic schema the nested-loop join executes a query. Suppose that we have a join query over 3 tables `T1`, `T2`, `T3` of the form:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
          INNER JOIN T3 ON P2(T2,T3)
WHERE P(T1,T2,T3).
```

Here, $P_1(T_1, T_2)$ and $P_2(T_2, T_3)$ are some join conditions (on expressions), whereas $P(t_1, t_2, t_3)$ is a condition over columns of tables T_1, T_2, T_3 .

The nested-loop join algorithm would execute this query in the following manner:

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

The notation $t_1 || t_2 || t_3$ means “a row constructed by concatenating the columns of rows t_1 , t_2 , and t_3 .” In some of the following examples, $NULL$ where a row name appears means that $NULL$ is used for each column of that row. For example, $t_1 || t_2 || NULL$ means “a row constructed by concatenating the columns of rows t_1 and t_2 , and $NULL$ for each column of t_3 .”

Now let's consider a query with nested outer joins:

```
SELECT * FROM T1 LEFT JOIN
          (T2 LEFT JOIN T3 ON P2(T2,T3))
          ON P1(T1,T2)
WHERE P(T1,T2,T3).
```

For this query, we modify the nested-loop pattern to get:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF P(t1,t2,NULL) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In general, for any nested loop for the first inner table in an outer join operation, a flag is introduced that is turned off before the loop and is checked after the loop. The flag is turned on when for the current row from the outer table a match from the table representing the inner operand is found. If at the end of the loop cycle the flag is still off, no match has been found for the current row of the outer table. In this case, the row is complemented by $NULL$ values for the columns of the inner tables. The result row is passed to

the final check for the output or into the next nested loop, but only if the row satisfies the join condition of all embedded outer joins.

In our example, the outer join table expressed by the following expression is embedded:

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

Note that for the query with inner joins, the optimizer could choose a different order of nested loops, such as this one:

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

For the queries with outer joins, the optimizer can choose only such an order where loops for outer tables precede loops for inner tables. Thus, for our query with outer joins, only one nesting order is possible. For the following query, the optimizer will evaluate two different nestings:

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
WHERE P(T1,T2,T3)
```

The nestings are these:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
    f1:=TRUE
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

and:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
    f1:=TRUE
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In both nestings, **T1** must be processed in the outer loop because it is used in an outer join. **T2** and **T3** are used in an inner join, so that join must be processed in the inner loop. However, because the join is an inner join, **T2** and **T3** can be processed in either order.

When discussing the nested-loop algorithm for inner joins, we omitted some details whose impact on the performance of query execution may be huge. We did not mention so-called “pushed-down” conditions. Suppose that our **WHERE** condition **P(T1, T2, T3)** can be represented by a conjunctive formula:

```
P(T1,T2,T3) = C1(T1) AND C2(T2) AND C3(T3).
```

In this case, MySQL actually uses the following nested-loop schema for the execution of the query with inner joins:

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2) {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

You see that each of the conjuncts **C1(T1)**, **C2(T2)**, **C3(T3)** are pushed out of the most inner loop to the most outer loop where it can be evaluated. If **C1(T1)** is a very restrictive condition, this condition pushdown may greatly reduce the number of rows from table **T1** passed to the inner loops. As a result, the execution time for the query may improve immensely.

For a query with outer joins, the **WHERE** condition is to be checked only after it has been found that the current row from the outer table has a match in the inner tables. Thus, the optimization of pushing conditions out of the inner nested loops cannot be applied directly to queries with outer joins. Here we have to introduce conditional pushed-down predicates guarded by the flags that are turned on when a match has been encountered.

For our example with outer joins with:

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

the nested-loop schema using guarded pushed-down conditions looks like this:

```
FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
    such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3
      such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
        IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
          t:=t1||t2||t3; OUTPUT t;
        }
        f2=TRUE;
        f1=TRUE;
      }
    }
  IF (!f2) {
    IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
      t:=t1||t2||NULL; OUTPUT t;
    }
    f1=TRUE;
  }
}
IF (!f1 && P(t1,NULL,NULL)) {
  t:=t1||NULL||NULL; OUTPUT t;
}
```

```
}
}
```

In general, pushed-down predicates can be extracted from join conditions such as `P1(T1, T2)` and `P(T2, T3)`. In this case, a pushed-down predicate is guarded also by a flag that prevents checking the predicate for the `NULL`-complemented row generated by the corresponding outer join operation.

Note that access by key from one inner table to another in the same nested join is prohibited if it is induced by a predicate from the `WHERE` condition. (We could use conditional key access in this case, but this technique is not employed yet in MySQL 5.5.)

8.13.8. Outer Join Simplification

Table expressions in the `FROM` clause of a query are simplified in many cases.

At the parser stage, queries with right outer joins operations are converted to equivalent queries containing only left join operations. In the general case, the conversion is performed according to the following rule:

```
(T1, ...) RIGHT JOIN (T2, ...) ON P(T1, ..., T2, ...) =
(T2, ...) LEFT JOIN (T1, ...) ON P(T1, ..., T2, ...)
```

All inner join expressions of the form `T1 INNER JOIN T2 ON P(T1, T2)` are replaced by the list `T1, T2, P(T1, T2)` being joined as a conjunct to the `WHERE` condition (or to the join condition of the embedding join, if there is any).

When the optimizer evaluates plans for join queries with outer join operation, it takes into consideration only the plans where, for each such operation, the outer tables are accessed before the inner tables. The optimizer options are limited because only such plans enables us to execute queries with outer joins operations by the nested loop schema.

Suppose that we have a query of the form:

```
SELECT * T1 LEFT JOIN T2 ON P1(T1, T2)
WHERE P(T1, T2) AND R(T2)
```

with `R(T2)` narrowing greatly the number of matching rows from table `T2`. If we executed the query as it is, the optimizer would have no other choice besides to access table `T1` before table `T2` that may lead to a very inefficient execution plan.

Fortunately, MySQL converts such a query into a query without an outer join operation if the `WHERE` condition is null-rejected. A condition is called null-rejected for an outer join operation if it evaluates to `FALSE` or to `UNKNOWN` for any `NULL`-complemented row built for the operation.

Thus, for this outer join:

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

Conditions such as these are null-rejected:

```
T2.B IS NOT NULL,
T2.B > 3,
T2.C <= T1.C,
T2.B < 2 OR T2.C > 1
```

Conditions such as these are not null-rejected:

```
T2.B IS NULL,
T1.B < 3 OR T2.B IS NOT NULL,
T1.B < 3 OR T2.B > 3
```

The general rules for checking whether a condition is null-rejected for an outer join operation are simple. A condition is null-rejected in the following cases:

- If it is of the form `A IS NOT NULL`, where `A` is an attribute of any of the inner tables
- If it is a predicate containing a reference to an inner table that evaluates to `UNKNOWN` when one of its arguments is `NULL`
- If it is a conjunction containing a null-rejected condition as a conjunct
- If it is a disjunction of null-rejected conditions

A condition can be null-rejected for one outer join operation in a query and not null-rejected for another. In the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
           LEFT JOIN T3 ON T3.B=T1.B
WHERE T3.C > 0
```

the `WHERE` condition is null-rejected for the second outer join operation but is not null-rejected for the first one.

If the `WHERE` condition is null-rejected for an outer join operation in a query, the outer join operation is replaced by an inner join operation.

For example, the preceding query is replaced with the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
           INNER JOIN T3 ON T3.B=T1.B
WHERE T3.C > 0
```

For the original query, the optimizer would evaluate plans compatible with only one access order `T1, T2, T3`. For the replacing query, it additionally considers the access sequence `T3, T1, T2`.

A conversion of one outer join operation may trigger a conversion of another. Thus, the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
           LEFT JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

will be first converted to the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
           INNER JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

which is equivalent to the query:

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

Now the remaining outer join operation can be replaced by an inner join, too, because the condition `T3.B=T2.B` is null-rejected and we get a query without outer joins at all:

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

Sometimes we succeed in replacing an embedded outer join operation, but cannot convert the embedding outer join. The following query:

```
SELECT * FROM T1 LEFT JOIN
           (T2 LEFT JOIN T3 ON T3.B=T2.B)
```

```

        ON T2.A=T1.A
WHERE T3.C > 0

```

is converted to:

```

SELECT * FROM T1 LEFT JOIN
    (T2 INNER JOIN T3 ON T3.B=T2.B)
    ON T2.A=T1.A
WHERE T3.C > 0,

```

That can be rewritten only to the form still containing the embedding outer join operation:

```

SELECT * FROM T1 LEFT JOIN
    (T2, T3)
    ON (T2.A=T1.A AND T3.B=T2.B)
WHERE T3.C > 0.

```

When trying to convert an embedded outer join operation in a query, we must take into account the join condition for the embedding outer join together with the `WHERE` condition. In the query:

```

SELECT * FROM T1 LEFT JOIN
    (T2 LEFT JOIN T3 ON T3.B=T2.B)
    ON T2.A=T1.A AND T3.C=T1.C
WHERE T3.D > 0 OR T1.D > 0

```

the `WHERE` condition is not null-rejected for the embedded outer join, but the join condition of the embedding outer join `T2.A=T1.A AND T3.C=T1.C` is null-rejected. So the query can be converted to:

```

SELECT * FROM T1 LEFT JOIN
    (T2, T3)
    ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
WHERE T3.D > 0 OR T1.D > 0

```

8.13.9. ORDER BY Optimization

In some cases, MySQL can use an index to satisfy an `ORDER BY` clause without doing any extra sorting.

The index can also be used even if the `ORDER BY` does not match the index exactly, as long as all of the unused portions of the index and all the extra `ORDER BY` columns are constants in the `WHERE` clause. The following queries use the index to resolve the `ORDER BY` part:

```

SELECT * FROM t1
ORDER BY key_part1, key_part2, ... ;

SELECT * FROM t1
WHERE key_part1=constant
ORDER BY key_part2;

SELECT * FROM t1
ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
WHERE key_part1=1
ORDER BY key_part1 DESC, key_part2 DESC;

```

In some cases, MySQL *cannot* use indexes to resolve the `ORDER BY`, although it still uses indexes to find the rows that match the `WHERE` clause. These cases include the following:

- You use `ORDER BY` on different keys:

```

SELECT * FROM t1 ORDER BY key1, key2;

```

- You use `ORDER BY` on nonconsecutive parts of a key:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
```

- You mix [ASC](#) and [DESC](#):

```
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
```

- The key used to fetch the rows is not the same as the one used in the [ORDER BY](#):

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
```

- You use [ORDER BY](#) with an expression that includes terms other than the key column name:

```
SELECT * FROM t1 ORDER BY ABS(key);
SELECT * FROM t1 ORDER BY -key;
```

- You are joining many tables, and the columns in the [ORDER BY](#) are not all from the first nonconstant table that is used to retrieve rows. (This is the first table in the [EXPLAIN](#) output that does not have a [const \[837\]](#) join type.)
- You have different [ORDER BY](#) and [GROUP BY](#) expressions.
- You index only a prefix of a column named in the [ORDER BY](#) clause. In this case, the index cannot be used to fully resolve the sort order. For example, if you have a [CHAR\(20\)](#) column, but index only the first 10 bytes, the index cannot distinguish values past the 10th byte and a [filesort](#) will be needed.
- The type of table index used does not store rows in order. For example, this is true for a [HASH](#) index in a [MEMORY](#) table.

Availability of an index for sorting may be affected by the use of column aliases. Suppose that the column [t1.a](#) is indexed. In this statement, the name of the column in the select list is [a](#). It refers to [t1.a](#), so for the reference to [a](#) in the [ORDER BY](#), the index can be used:

```
SELECT a FROM t1 ORDER BY a;
```

In this statement, the name of the column in the select list is also [a](#), but it is the alias name. It refers to [ABS\(a\)](#), so for the reference to [a](#) in the [ORDER BY](#), the index cannot be used:

```
SELECT ABS(a) AS a FROM t1 ORDER BY a;
```

In the following statement, the [ORDER BY](#) refers to a name that is not the name of a column in the select list. But there is a column in [t1](#) named [a](#), so the [ORDER BY](#) uses that, and the index can be used. (The resulting sort order may be completely different from the order for [ABS\(a\)](#), of course.)

```
SELECT ABS(a) AS b FROM t1 ORDER BY a;
```

By default, MySQL sorts all [GROUP BY *col1, col2, ...*](#) queries as if you specified [ORDER BY *col1, col2, ...*](#) in the query as well. If you include an [ORDER BY](#) clause explicitly that contains the same column list, MySQL optimizes it away without any speed penalty, although the sorting still occurs. If a query includes [GROUP BY](#) but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying [ORDER BY NULL](#). For example:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

With [EXPLAIN SELECT ... ORDER BY](#), you can check whether MySQL can use indexes to resolve the query. It cannot if you see [Using filesort](#) in the [Extra](#) column. See [Section 8.8.1, “Optimizing Queries with EXPLAIN”](#). [Filesort](#) uses a fixed-length row-storage format similar to that used by the [MEMORY](#) storage engine. Variable-length types such as [VARCHAR](#) are stored using a fixed length.

MySQL has two `filesort` algorithms for sorting and retrieving results. The original method uses only the `ORDER BY` columns. The modified method uses not just the `ORDER BY` columns, but all the columns used in the query.

The optimizer selects which `filesort` algorithm to use. It normally uses the modified algorithm except when `BLOB` or `TEXT` columns are involved, in which case it uses the original algorithm.

The original `filesort` algorithm works as follows:

1. Read all rows according to key or by table scanning. Rows that do not match the `WHERE` clause are skipped.
2. For each row, store a pair of values in a buffer (the sort key and the row pointer). The size of the buffer is the value of the `sort_buffer_size` [554] system variable.
3. When the buffer gets full, run a `qsort` (quicksort) on it and store the result in a temporary file. Save a pointer to the sorted block. (If all pairs fit into the sort buffer, no temporary file is created.)
4. Repeat the preceding steps until all rows have been read.
5. Do a multi-merge of up to `MERGEBUFF` (7) regions to one block in another temporary file. Repeat until all blocks from the first file are in the second file.
6. Repeat the following until there are fewer than `MERGEBUFF2` (15) blocks left.
7. On the last multi-merge, only the pointer to the row (the last part of the sort key) is written to a result file.
8. Read the rows in sorted order by using the row pointers in the result file. To optimize this, we read in a big block of row pointers, sort them, and use them to read the rows in sorted order into a row buffer. The size of the buffer is the value of the `read_rnd_buffer_size` [545] system variable. The code for this step is in the `sql/records.cc` source file.

One problem with this approach is that it reads rows twice: One time when evaluating the `WHERE` clause, and again after sorting the pair values. And even if the rows were accessed successively the first time (for example, if a table scan is done), the second time they are accessed randomly. (The sort keys are ordered, but the row positions are not.)

The modified `filesort` algorithm incorporates an optimization such that it records not only the sort key value and row position, but also the columns required for the query. This avoids reading the rows twice. The modified `filesort` algorithm works like this:

1. Read the rows that match the `WHERE` clause.
2. For each row, record a tuple of values consisting of the sort key value and row position, and also the columns required for the query.
3. Sort the tuples by sort key value
4. Retrieve the rows in sorted order, but read the required columns directly from the sorted tuples rather than by accessing the table a second time.

Using the modified `filesort` algorithm, the tuples are longer than the pairs used in the original method, and fewer of them fit in the sort buffer (the size of which is given by `sort_buffer_size` [554]). As a result, it is possible for the extra I/O to make the modified approach slower, not faster. To avoid a slowdown, the optimization is used only if the total size of the extra columns in the sort tuple does not

exceed the value of the `max_length_for_sort_data` [522] system variable. (A symptom of setting the value of this variable too high is a combination of high disk activity and low CPU activity.)

For slow queries for which `filesort` is not used, try lowering `max_length_for_sort_data` [522] to a value that is appropriate to trigger a `filesort`.

If you want to increase `ORDER BY` speed, check whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, you can try the following strategies:

- Increase the size of the `sort_buffer_size` [554] variable.
- Increase the size of the `read_rnd_buffer_size` [545] variable.
- Use less RAM per row by declaring columns only as large as they need to be to hold the values stored in them. For example, `CHAR(16)` is better than `CHAR(200)` if values never exceed 16 characters.
- Change `tmpdir` [568] to point to a dedicated file system with large amounts of free space. Also, this option accepts several paths that are used in round-robin fashion, so you can use this feature to spread the load across several directories. Paths should be separated by colon characters (":") on Unix and semicolon characters(";") on Windows. The paths should be for directories in file systems that are located on different *physical* disks, not different partitions on the same disk.

8.13.10. GROUP BY Optimization

The most general way to satisfy a `GROUP BY` clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and to avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for `GROUP BY` are that all `GROUP BY` columns reference attributes from the same index, and that the index stores its keys in order (for example, this is a `BTREE` index and not a `HASH` index). Whether use of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

There are two ways to execute a `GROUP BY` query through index access, as detailed in the following sections. In the first method, the grouping operation is applied together with all range predicates (if any). The second method first performs a range scan, and then groups the resulting tuples.

In MySQL, `GROUP BY` is used for sorting, so the server may also apply `ORDER BY` optimizations to grouping. See [Section 8.13.9, "ORDER BY Optimization"](#).

8.13.10.1. Loose Index Scan

The most efficient way to process `GROUP BY` is when an index is used to directly retrieve the grouping columns. With this access method, MySQL uses the property of some index types that the keys are ordered (for example, `BTREE`). This property enables use of lookup groups in an index without having to consider all keys in the index that satisfy all `WHERE` conditions. This access method considers only a fraction of the keys in an index, so it is called a *loose index scan*. When there is no `WHERE` clause, a loose index scan reads as many keys as the number of groups, which may be a much smaller number than that of all keys. If the `WHERE` clause contains range predicates (see the discussion of the `range` [838] join type in [Section 8.8.1, "Optimizing Queries with EXPLAIN"](#)), a loose index scan looks up the first key of each group that satisfies the range conditions, and again reads the least possible number of keys. This is possible under the following conditions:

- The query is over a single table.

- The `GROUP BY` names only columns that form a leftmost prefix of the index and no other columns. (If, instead of `GROUP BY`, the query has a `DISTINCT` clause, all distinct attributes refer to columns that form a leftmost prefix of the index.) For example, if a table `t1` has an index on `(c1,c2,c3)`, loose index scan is applicable if the query has `GROUP BY c1, c2,`. It is not applicable if the query has `GROUP BY c2, c3` (the columns are not a leftmost prefix) or `GROUP BY c1, c2, c4` (`c4` is not in the index).
- The only aggregate functions used in the select list (if any) are `MIN()` [1206] and `MAX()` [1205], and all of them refer to the same column. The column must be in the index and must follow the columns in the `GROUP BY`.
- Any other parts of the index than those from the `GROUP BY` referenced in the query must be constants (that is, they must be referenced in equalities with constants), except for the argument of `MIN()` [1206] or `MAX()` [1205] functions.
- For columns in the index, full column values must be indexed, not just a prefix. For example, with `c1 VARCHAR(20)`, `INDEX (c1(10))`, the index cannot be used for loose index scan.

If loose index scan is applicable to a query, the `EXPLAIN` output shows `Using index for group-by` in the `Extra` column.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The loose index scan access method can be used for the following queries:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

The following queries cannot be executed with this quick select method, for the reasons given:

- There are aggregate functions other than `MIN()` [1206] or `MAX()` [1205]:

```
SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
```

- The columns in the `GROUP BY` clause do not form a leftmost prefix of the index:

```
SELECT c1, c2 FROM t1 GROUP BY c2, c3;
```

- The query refers to a part of a key that comes after the `GROUP BY` part, and for which there is no equality with a constant:

```
SELECT c1, c3 FROM t1 GROUP BY c1, c2;
```

Were the query to include `WHERE c3 = const`, loose index scan could be used.

As of MySQL 5.5, the loose index scan access method can be applied to other forms of aggregate function references in the select list, in addition to the `MIN()` [1206] and `MAX()` [1205] references already supported:

- `AVG(DISTINCT)` [1203], `SUM(DISTINCT)` [1206], and `COUNT(DISTINCT)` [1204] are supported. `AVG(DISTINCT)` [1203] and `SUM(DISTINCT)` [1206] take a single argument. `COUNT(DISTINCT)` [1204] can have more than one column argument.
- There must be no `GROUP BY` or `DISTINCT` clause in the query.
- The loose scan limitations described earlier still apply.

Assume that there is an index `idx(c1, c2, c3)` on table `t1(c1, c2, c3, c4)`. The loose index scan access method can be used for the following queries:

```
SELECT COUNT(DISTINCT c1), SUM(DISTINCT c1) FROM t1;
SELECT COUNT(DISTINCT c1, c2), COUNT(DISTINCT c2, c1) FROM t1;
```

Loose index scan is not applicable for the following queries:

```
SELECT DISTINCT COUNT(DISTINCT c1) FROM t1;
SELECT COUNT(DISTINCT c1) FROM t1 GROUP BY c1;
```

8.13.10.2. Tight Index Scan

A tight index scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a loose index scan are not met, it still may be possible to avoid creation of temporary tables for `GROUP BY` queries. If there are range conditions in the `WHERE` clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Because this method reads all keys in each range defined by the `WHERE` clause, or scans the whole index if there are no range conditions, we term it a *tight index scan*. With a tight index scan, the grouping operation is performed only after all keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that there is a constant equality condition for all columns in a query referring to parts of the key coming before or in between parts of the `GROUP BY` key. The constants from the equality conditions fill in any “gaps” in the search keys so that it is possible to form complete prefixes of the index. These index prefixes then can be used for index lookups. If we require sorting of the `GROUP BY` result, and it is possible to form search keys that are prefixes of the index, MySQL also avoids extra sorting operations because searching with prefixes in an ordered index already retrieves all the keys in order.

Assume that there is an index `idx(c1, c2, c3)` on table `t1(c1, c2, c3, c4)`. The following queries do not work with the loose index scan access method described earlier, but still work with the tight index scan access method.

- There is a gap in the `GROUP BY`, but it is covered by the condition `c2 = 'a'`:

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- The `GROUP BY` does not begin with the first part of the key, but there is a condition that provides a constant for that part:

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

8.13.11. DISTINCT Optimization

`DISTINCT` combined with `ORDER BY` needs a temporary table in many cases.

Because `DISTINCT` may use `GROUP BY`, learn how MySQL works with columns in `ORDER BY` or `HAVING` clauses that are not part of the selected columns. See [Section 12.16.3, “MySQL Extensions to GROUP BY”](#).

In most cases, a `DISTINCT` clause can be considered as a special case of `GROUP BY`. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;
SELECT c1, c2, c3 FROM t1
```

```
WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to `GROUP BY` queries can be also applied to queries with a `DISTINCT` clause. Thus, for more details on the optimization possibilities for `DISTINCT` queries, see [Section 8.13.10, “GROUP BY Optimization”](#).

When combining `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds `row_count` unique rows.

If you do not use columns from all tables named in a query, MySQL stops scanning any unused tables as soon as it finds the first match. In the following case, assuming that `t1` is used before `t2` (which you can check with `EXPLAIN`), MySQL stops reading from `t2` (for any particular row in `t1`) when it finds the first row in `t2`:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

8.13.12. Optimizing Subqueries with EXISTS Strategy

Certain optimizations are applicable to comparisons that use the `IN` operator to test subquery results (or that use `=ANY`, which is equivalent). This section discusses these optimizations, particularly with regard to the challenges that `NULL` values present. The last part of the discussion includes suggestions on what you can do to help the optimizer.

Consider the following subquery comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

MySQL evaluates queries “from outside to inside.” That is, it first obtains the value of the outer expression `outer_expr`, and then runs the subquery and captures the rows that it produces.

A very useful optimization is to “inform” the subquery that the only rows of interest are those where the inner expression `inner_expr` is equal to `outer_expr`. This is done by pushing down an appropriate equality into the subquery’s `WHERE` clause. That is, the comparison is converted to this:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

After the conversion, MySQL can use the pushed-down equality to limit the number of rows that it must examine when evaluating the subquery.

More generally, a comparison of `N` values to a subquery that returns `N`-value rows is subject to the same conversion. If `oei` and `iei` represent corresponding outer and inner expression values, this subquery comparison:

```
(oe_1, ..., oe_N) IN  
(SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

Becomes:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where  
AND oe_1 = ie_1  
AND ...  
AND oe_N = ie_N)
```

The following discussion assumes a single pair of outer and inner expression values for simplicity.

The conversion just described has its limitations. It is valid only if we ignore possible `NULL` values. That is, the “pushdown” strategy works as long as both of these two conditions are true:

- `outer_expr` and `inner_expr` cannot be `NULL`.

- You do not need to distinguish `NULL` from `FALSE` subquery results. (If the subquery is a part of an `OR` [1095] or `AND` [1094] expression in the `WHERE` clause, MySQL assumes that you do not care.)

When either or both of those conditions do not hold, optimization is more complex.

Suppose that `outer_expr` is known to be a non-`NULL` value but the subquery does not produce a row such that `outer_expr = inner_expr`. Then `outer_expr IN (SELECT ...)` evaluates as follows:

- `NULL`, if the `SELECT` produces any row where `inner_expr` is `NULL`
- `FALSE`, if the `SELECT` produces only non-`NULL` values or produces nothing

In this situation, the approach of looking for rows with `outer_expr = inner_expr` is no longer valid. It is necessary to look for such rows, but if none are found, also look for rows where `inner_expr` is `NULL`. Roughly speaking, the subquery can be converted to:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND
        (outer_expr=inner_expr OR inner_expr IS NULL))
```

The need to evaluate the extra `IS NULL` [1091] condition is why MySQL has the `ref_or_null` [838] access method:

```
mysql> EXPLAIN
-> SELECT outer_expr IN (SELECT t2.maybe_null_key
->                        FROM t2, t3 WHERE ...)
-> FROM t1;
***** 1. row *****
      id: 1
  select_type: PRIMARY
      table: t1
  ...
***** 2. row *****
      id: 2
  select_type: DEPENDENT SUBQUERY
      table: t2
      type: ref_or_null
possible_keys: maybe_null_key
      key: maybe_null_key
      key_len: 5
      ref: func
      rows: 2
  Extra: Using where; Using index
  ...
```

The `unique_subquery` [838] and `index_subquery` [838] subquery-specific access methods also have “or `NULL`” variants. However, they are not visible in `EXPLAIN` output, so you must use `EXPLAIN EXTENDED` followed by `SHOW WARNINGS` (note the `checking NULL` in the warning message):

```
mysql> EXPLAIN EXTENDED
-> SELECT outer_expr IN (SELECT maybe_null_key FROM t2) FROM t1\G
***** 1. row *****
      id: 1
  select_type: PRIMARY
      table: t1
  ...
***** 2. row *****
      id: 2
  select_type: DEPENDENT SUBQUERY
      table: t2
      type: index_subquery
possible_keys: maybe_null_key
      key: maybe_null_key
      key_len: 5
```

```

    ref: func
    rows: 2
    Extra: Using index

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1003
Message: select (`test`.`t1`.`outer_expr`,
               ((`test`.`t1`.`outer_expr`) in t2 on
               maybe_null_key checking NULL)) AS `outer_expr` IN (SELECT
               maybe_null_key FROM t2)` from `test`.`t1`

```

The additional `OR ... IS NULL` condition makes query execution slightly more complicated (and some optimizations within the subquery become inapplicable), but generally this is tolerable.

The situation is much worse when `outer_expr` can be `NULL`. According to the SQL interpretation of `NULL` as “unknown value,” `NULL IN (SELECT inner_expr ...)` should evaluate to:

- `NULL`, if the `SELECT` produces any rows
- `FALSE`, if the `SELECT` produces no rows

For proper evaluation, it is necessary to be able to check whether the `SELECT` has produced any rows at all, so `outer_expr = inner_expr` cannot be pushed down into the subquery. This is a problem, because many real world subqueries become very slow unless the equality can be pushed down.

Essentially, there must be different ways to execute the subquery depending on the value of `outer_expr`.

The optimizer chooses SQL compliance over speed, so it accounts for the possibility that `outer_expr` might be `NULL`.

If `outer_expr` is `NULL`, to evaluate the following expression, it is necessary to run the `SELECT` to determine whether it produces any rows:

```
NULL IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

It is necessary to run the original `SELECT` here, without any pushed-down equalities of the kind mentioned earlier.

On the other hand, when `outer_expr` is not `NULL`, it is absolutely essential that this comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

be converted to this expression that uses a pushed-down condition:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

Without this conversion, subqueries will be slow. To solve the dilemma of whether to push down or not push down conditions into the subquery, the conditions are wrapped in “trigger” functions. Thus, an expression of the following form:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

is converted into:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
        AND trigcond(outer_expr=inner_expr))
```

More generally, if the subquery comparison is based on several pairs of outer and inner expressions, the conversion takes this comparison:

```
(oe_1, ..., oe_N) IN (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

and converts it to this expression:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
        AND trigcond(oe_1=ie_1)
        AND ...
        AND trigcond(oe_N=ie_N)
    )
```

Each `trigcond(X)` is a special function that evaluates to the following values:

- `X` when the “linked” outer expression `oe_i` is not `NULL`
- `TRUE` when the “linked” outer expression `oe_i` is `NULL`

Note that trigger functions are *not* triggers of the kind that you create with `CREATE TRIGGER`.

Equalities that are wrapped into `trigcond()` functions are not first class predicates for the query optimizer. Most optimizations cannot deal with predicates that may be turned on and off at query execution time, so they assume any `trigcond(X)` to be an unknown function and ignore it. At the moment, triggered equalities can be used by those optimizations:

- Reference optimizations: `trigcond(X=Y [OR Y IS NULL])` can be used to construct `ref` [837], `eq_ref` [837], or `ref_or_null` [838] table accesses.
- Index lookup-based subquery execution engines: `trigcond(X=Y)` can be used to construct `unique_subquery` [838] or `index_subquery` [838] accesses.
- Table-condition generator: If the subquery is a join of several tables, the triggered condition will be checked as soon as possible.

When the optimizer uses a triggered condition to create some kind of index lookup-based access (as for the first two items of the preceding list), it must have a fallback strategy for the case when the condition is turned off. This fallback strategy is always the same: Do a full table scan. In `EXPLAIN` output, the fallback shows up as `Full scan on NULL key` in the `Extra` column:

```
mysql> EXPLAIN SELECT t1.col1,
      -> t1.col1 IN (SELECT t2.key1 FROM t2 WHERE t2.col2=t1.col2) FROM t1\G
***** 1. row *****
      id: 1
      select_type: PRIMARY
      table: t1
      ...
***** 2. row *****
      id: 2
      select_type: DEPENDENT SUBQUERY
      table: t2
      type: index_subquery
      possible_keys: key1
      key: key1
      key_len: 5
      ref: func
      rows: 2
      Extra: Using where; Full scan on NULL key
```

If you run `EXPLAIN EXTENDED` followed by `SHOW WARNINGS`, you can see the triggered condition:

```
***** 1. row *****
      Level: Note
      Code: 1003
      Message: select `test`.`t1`.`col1` AS `col1`,
```

```
<in_optimizer>(`test`.`t1`.`col1`,
<exists>(<index_lookup>(<cache>(`test`.`t1`.`col1`) in t2
on key1 checking NULL
where (`test`.`t2`.`col2` = `test`.`t1`.`col2`) having
trigcond(<is_not_null_test>(`test`.`t2`.`key1`)))) AS
`t1.col1 IN (select t2.key1 from t2 where t2.col2=t1.col2)`
from `test`.`t1`
```

The use of triggered conditions has some performance implications. A `NULL IN (SELECT ...)` expression now may cause a full table scan (which is slow) when it previously did not. This is the price paid for correct results (the goal of the trigger-condition strategy was to improve compliance and not speed).

For multiple-table subqueries, execution of `NULL IN (SELECT ...)` will be particularly slow because the join optimizer does not optimize for the case where the outer expression is `NULL`. It assumes that subquery evaluations with `NULL` on the left side are very rare, even if there are statistics that indicate otherwise. On the other hand, if the outer expression might be `NULL` but never actually is, there is no performance penalty.

To help the query optimizer better execute your queries, use these tips:

- Declare a column as `NOT NULL` if it really is. (This also helps other aspects of the optimizer by simplifying condition testing for the column.)
- If you do not need to distinguish a `NULL` from `FALSE` subquery result, you can easily avoid the slow execution path. Replace a comparison that looks like this:

```
outer_expr IN (SELECT inner_expr FROM ...)
```

with this expression:

```
(outer_expr IS NOT NULL) AND (outer_expr IN (SELECT inner_expr FROM ...))
```

Then `NULL IN (SELECT ...)` will never be evaluated because MySQL stops evaluating `AND` [1094] parts as soon as the expression result is clear.

Chapter 9. Language Structure

Table of Contents

9.1. Literal Values	937
9.1.1. String Literals	937
9.1.2. Number Literals	940
9.1.3. Date and Time Literals	940
9.1.4. Hexadecimal Literals	942
9.1.5. Boolean Literals	942
9.1.6. Bit-Field Literals	943
9.1.7. <code>NULL</code> Values	943
9.2. Schema Object Names	943
9.2.1. Identifier Qualifiers	945
9.2.2. Identifier Case Sensitivity	946
9.2.3. Mapping of Identifiers to File Names	948
9.2.4. Function Name Parsing and Resolution	950
9.3. Reserved Words	953
9.4. User-Defined Variables	956
9.5. Expression Syntax	959
9.6. Comment Syntax	961

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as database, table, and column names
- Reserved words
- User-defined and system variables
- Comments

9.1. Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal values, boolean values, and `NULL`. The section also covers the various nuances and “gotchas” that you may run into when dealing with these basic types in MySQL.

9.1.1. String Literals

A string is a sequence of bytes or characters, enclosed within either single quote (“’”) or double quote (“””) characters. Examples:

```
'a string'  
"another string"
```

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

```
'a string'  
'a' ' ' 'string'
```

If the `ANSI_QUOTES` [612] SQL mode is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

A *binary string* is a string of bytes that has no character set or collation. A *nonbinary string* is a string of characters that has a character set and collation. For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte. For nonbinary strings the unit is the character and some character sets support multi-byte characters. Character value ordering is a function of the string collation.

String literals may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

You can use `N'literal'` (or `n'literal'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For more information about these forms of string syntax, see [Section 10.1.3.5, “Character String Literal Character Set and Collation”](#), and [Section 10.1.3.6, “National Character Set”](#).

Within a string, certain sequences have special meaning unless the `NO_BACKSLASH_ESCAPES [614]` SQL mode is enabled. Each of these sequences begins with a backslash (“\”), known as the *escape character*. MySQL recognizes the escape sequences shown in [Table 9.1, “Special Character Escape Sequences”](#). For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, “\x” is just “x”. These sequences are case sensitive. For example, “\b” is interpreted as a backspace, but “\B” is interpreted as “B”. Escape processing is done according to the character set indicated by the `character_set_connection [487]` system variable. This is true even for strings that are preceded by an introducer that indicates a different character set, as discussed in [Section 10.1.3.5, “Character String Literal Character Set and Collation”](#).

Table 9.1. Special Character Escape Sequences

Escape Sequence	Character Represented by Sequence
<code>\0</code>	An ASCII NUL (0x00) character.
<code>\'</code>	A single quote (“'”) character.
<code>\"</code>	A double quote (“””) character.
<code>\b</code>	A backspace character.
<code>\n</code>	A newline (linefeed) character.
<code>\r</code>	A carriage return character.
<code>\t</code>	A tab character.
<code>\z</code>	ASCII 26 (Control+Z). See note following the table.
<code>\\</code>	A backslash (“\”) character.
<code>\%</code>	A “%” character. See note following the table.
<code>_</code>	A “_” character. See note following the table.

The ASCII 26 character can be encoded as “\z” to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use `mysql db_name < file_name`.

The “\%” and “_” sequences are used to search for literal instances of “%” and “_” in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of the [LIKE \[1111\]](#) operator in [Section 12.5.1, “String Comparison Functions”](#). If you use “\%” or “_” outside of pattern-matching contexts, they evaluate to the strings “\%” and “_”, not to “%” and “_”.

There are several ways to include quote characters within a string:

- A “'” inside a string quoted with “'” may be written as “''”.
- A “” inside a string quoted with “” may be written as “””.
- Precede the quote character by an escape character (“\”).
- A “'” inside a string quoted with “” needs no special treatment and need not be doubled or escaped. In the same way, “” inside a string quoted with “'” needs no special treatment.

The following `SELECT` statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', "hello", ""hello"", 'hel'lo', '\hello';
+-----+-----+-----+-----+-----+
| hello | "hello" | ""hello"" | hel'lo | '\hello |
+-----+-----+-----+-----+-----+

mysql> SELECT "hello", "'hello'", ""'hello'"" , "hel""lo", "\"hello";
+-----+-----+-----+-----+-----+
| hello | 'hello' | ""'hello'"" | hel"lo | "hello |
+-----+-----+-----+-----+-----+

mysql> SELECT 'This\nIs\nFour\nLines';
+-----+
| This
Is
Four
Lines |
+-----+

mysql> SELECT 'disappearing\ backslash';
+-----+
| disappearing backslash |
+-----+
```

If you want to insert binary data into a string column (such as a `BLOB` column), you should represent certain characters by escape sequences. Backslash (“\”) and the quote character used to quote the string must be escaped. In certain client environments, it may also be necessary to escape `NUL` or Control+Z. The `mysql` client truncates quoted strings containing `NUL` characters if they are not escaped, and Control+Z may be taken for END-OF-FILE on Windows if not escaped. For the escape sequences that represent each of these characters, see [Table 9.1, “Special Character Escape Sequences”](#).

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

- Process the string with a function that escapes the special characters. In a C program, you can use the `mysql_real_escape_string()` C API function to escape characters. See [Section 22.8.3.53, “mysql_real_escape_string\(\)”](#). Within SQL statements that construct other SQL statements, you can use the `QUOTE()` [1107] function. The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See [Section 22.10, “MySQL Perl API”](#). Other language interfaces may provide a similar capability.
- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values

to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

9.1.2. Number Literals

Integers are represented as a sequence of digits. Floats use “.” as a decimal separator. Either type of number may be preceded by “-” or “+” to indicate a negative or positive value, respectively

Examples of valid integers:

```
1221
0
-32
```

Examples of valid floating-point numbers:

```
294.42
-32032.6809e+10
148.00
```

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

9.1.3. Date and Time Literals

Date and time values can be represented in several formats, such as quoted strings or as numbers, depending on the exact type of the value and other factors. For example, in contexts where MySQL expects a date, it interprets any of '2015-07-21', '20150721', and 20150721 as a date.

This section describes the acceptable formats for date and time literals. For more information about the temporal data types, such as the range of permitted values, consult these sections:

- [Section 11.1.2, “Date and Time Type Overview”](#)
- [Section 11.3, “Date and Time Types”](#)

Standard SQL and ODBC Date and Time Literals. Standard SQL permits temporal literals to be specified using a type keyword and a string. The space between the keyword and string is optional.

```
DATE 'str'
TIME 'str'
TIMESTAMP 'str'
```

MySQL recognizes those constructions and also the corresponding ODBC syntax:

```
{ d 'str' }
{ t 'str' }
{ ts 'str' }
```

However, MySQL ignores the type keyword and each of the preceding constructions produces the string value 'str', with a type of `VARCHAR`.

String and Numeric Literals in Date and Time Context. MySQL recognizes `DATE` values in these formats:

- As a string in either 'YYYY-MM-DD' or 'YY-MM-DD' format. A “relaxed” syntax is permitted: Any punctuation character may be used as the delimiter between date parts. For example, '2012-12-31', '2012/12/31', '2012^12^31', and '2012@12@31' are equivalent.
- As a string with no delimiters in either 'YYYYMMDD' or 'YYMMDD' format, provided that the string makes sense as a date. For example, '20070523' and '070523' are interpreted as '2007-05-23', but '071332' is illegal (it has nonsensical month and day parts) and becomes '0000-00-00'.

- As a number in either `YYYYMMDD` or `YYMMDD` format, provided that the number makes sense as a date. For example, `19830905` and `830905` are interpreted as `'1983-09-05'`.

MySQL recognizes `DATETIME` and `TIMESTAMP` values in these formats:

- As a string in either `'YYYY-MM-DD HH:MM:SS'` or `'YY-MM-DD HH:MM:SS'` format. A “relaxed” syntax is permitted here, too: Any punctuation character may be used as the delimiter between date parts or time parts. For example, `'2012-12-31 11:30:45'`, `'2012^12^31 11+30+45'`, `'2012/12/31 11*30*45'`, and `'2012@12@31 11^30^45'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDDHHMMSS'` or `'YYMMDDHHMMSS'` format, provided that the string makes sense as a date. For example, `'20070523091528'` and `'070523091528'` are interpreted as `'2007-05-23 09:15:28'`, but `'071122129015'` is illegal (it has a nonsensical minute part) and becomes `'0000-00-00 00:00:00'`.
- As a number in either `YYYYMMDDHHMMSS` or `YYMMDDHHMMSS` format, provided that the number makes sense as a date. For example, `19830905132800` and `830905132800` are interpreted as `'1983-09-05 13:28:00'`.

A `DATETIME` or `TIMESTAMP` value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `DATETIME` or `TIMESTAMP` columns. For information about fractional seconds support in MySQL, see [Section 11.3.6, “Fractional Seconds in Time Values”](#).

Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

- Year values in the range `70-99` are converted to `1970-1999`.
- Year values in the range `00-69` are converted to `2000-2069`.

See also [Section 11.3.8, “Two-Digit Years in Dates”](#).

For values specified as strings that include date part delimiters, it is unnecessary to specify two digits for month or day values that are less than 10. `'2015-6-9'` is the same as `'2015-06-09'`. Similarly, for values specified as strings that include time part delimiters, it is unnecessary to specify two digits for hour, minute, or second values that are less than 10. `'2015-10-30 1:2:3'` is the same as `'2015-10-30 01:02:03'`.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in `YYYYMMDD` or `YYYYMMDDHHMMSS` format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in `YYMMDD` or `YYMMDDHHMMSS` format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as nondelimited strings are interpreted according their length. For a string 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify `'9903'`, thinking that represents March, 1999, MySQL converts it to the “zero” date value. This occurs because the year and month values are `99` and `03`, but the day part is completely missing. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, to insert the value `'1999-03-00'`, use `'990300'`.

MySQL recognizes `TIME` values in these formats:

- As a string in `'D HH:MM:SS'` format. You can also use one of the following “relaxed” syntaxes: `'HH:MM:SS'`, `'HH:MM'`, `'D HH:MM'`, `'D HH'`, or `'SS'`. Here `D` represents days and can have a value from 0 to 34.
- As a string with no delimiters in `'HHMMSS'` format, provided that it makes sense as a time. For example, `'101112'` is understood as `'10:11:12'`, but `'109712'` is illegal (it has a nonsensical minute part) and becomes `'00:00:00'`.
- As a number in `HHMMSS` format, provided that it makes sense as a time. For example, `101112` is understood as `'10:11:12'`. The following alternative formats are also understood: `SS`, `MMSS`, or `HHMMSS`.

A trailing fractional seconds part is recognized in the `'D HH:MM:SS.fraction'`, `'HH:MM:SS.fraction'`, `'HHMMSS.fraction'`, and `HHMMSS.fraction` time formats, where `fraction` is the fractional part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `TIME` columns. For information about fractional seconds support in MySQL, see [Section 11.3.6, “Fractional Seconds in Time Values”](#).

For `TIME` values specified as strings that include a time part delimiter, it is unnecessary to specify two digits for hours, minutes, or seconds values that are less than 10. `'8:3:2'` is the same as `'08:03:02'`.

9.1.4. Hexadecimal Literals

MySQL supports hexadecimal values, written using `X'val'`, `x'val'`, or `0xval` format, where `val` contains hexadecimal digits (`0..9, A..F`). Lettercase of the digits does not matter. For values written using `X'val'` or `x'val'` format, `val` must contain an even number of digits. For values written using `0xval` syntax, values that contain an odd number of digits are treated as having an extra leading 0. For example, `0x0a` and `0xaaa` are interpreted as `0x0a` and `0x0aaa`.

In numeric contexts, hexadecimal values act like integers (64-bit precision). In string contexts, they act like binary strings, where each pair of hex digits is converted to a character:

```
mysql> SELECT X'4D7953514C';
-> 'MySQL'
mysql> SELECT 0x0a+0;
-> 10
mysql> SELECT 0x5061756c;
-> 'Paul'
```

The default type of a hexadecimal value is a string. If you want to ensure that the value is treated as a number, you can use `CAST(... AS UNSIGNED)` [1168]:

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
-> 'A', 65
```

The `X'hexstring'` syntax is based on standard SQL. The `0x` syntax is based on ODBC. Hexadecimal strings are often used by ODBC to supply values for `BLOB` columns.

You can convert a string or a number to a string in hexadecimal format with the `HEX()` [1104] function:

```
mysql> SELECT HEX('cat');
-> '636174'
mysql> SELECT 0x636174;
-> 'cat'
```

9.1.5. Boolean Literals

The constants `TRUE` and `FALSE` evaluate to 1 and 0, respectively. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
```

```
-> 1, 1, 0, 0
```

9.1.6. Bit-Field Literals

Bit-field values can be written using `b'value'` or `0bvalue` notation. `value` is a binary value written using zeros and ones.

Bit-field notation is convenient for specifying values to be assigned to `BIT` columns:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
mysql> INSERT INTO t SET b = b'0101';
```

Bit values are returned as binary values. To display them in printable form, add `0` or use a conversion function such as `BIN()` [1101]. High-order 0 bits are not displayed in the converted value.

```
mysql> SELECT b+0, BIN(b+0), OCT(b+0), HEX(b+0) FROM t;
+-----+-----+-----+-----+
| b+0 | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+-----+-----+-----+-----+
| 255 | 11111111 | 377      | FF       |
| 10  | 1010     | 12       | A        |
| 5   | 101      | 5        | 5        |
+-----+-----+-----+-----+
```

Bit values assigned to user variables are treated as binary strings. To assign a bit value as a number to a user variable, use `CAST()` [1168] or `+0`:

```
mysql> SET @v1 = 0b1000001;
mysql> SET @v2 = CAST(0b1000001 AS UNSIGNED), @v3 = 0b1000001+0;
mysql> SELECT @v1, @v2, @v3;
+-----+-----+-----+
| @v1 | @v2 | @v3 |
+-----+-----+-----+
| A   | 65  | 65  |
+-----+-----+-----+
```

9.1.7. NULL Values

The `NULL` value means “no data.” `NULL` can be written in any lettercase. A synonym is `\N` (case sensitive).

For text file import or export operations performed with `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, `NULL` is represented by the `\N` sequence. See Section 13.2.6, “`LOAD DATA INFILE Syntax`”.

Be aware that the `NULL` value is different from values such as `0` for numeric types or the empty string for string types. For more information, see Section C.5.5.3, “`Problems with NULL Values`”.

9.2. Schema Object Names

Certain objects within MySQL, including database, table, index, column, alias, view, stored procedure, partition, tablespace, and other object names are known as identifiers. This section describes the permissible syntax for identifiers in MySQL. Section 9.2.2, “`Identifier Case Sensitivity`”, describes which types of identifiers are case sensitive and under what conditions.

An identifier may be quoted or unquoted. If an identifier contains special characters or is a reserved word, you *must* quote it whenever you refer to it. (Exception: A reserved word that follows a period in a qualified name must be an identifier, so it need not be quoted.) Reserved words are listed at Section 9.3, “`Reserved Words`”.

Identifiers are converted to Unicode internally. They may contain these characters:

- Permitted characters in unquoted identifiers:
 - ASCII: [0-9,a-z,A-Z\$_] (basic Latin letters, digits 0-9, dollar, underscore)
 - Extended: U+0080 .. U+FFFF
- Permitted characters in quoted identifiers include the full Unicode Basic Multilingual Plane (BMP), except U+0000:
 - ASCII: U+0001 .. U+007F
 - Extended: U+0080 .. U+FFFF
- ASCII NUL (U+0000) and supplementary characters (U+10000 and higher) are not permitted in quoted or unquoted identifiers.
- Identifiers may begin with a digit but unless quoted may not consist solely of digits.
- Database, table, and column names cannot end with space characters.

The identifier quote character is the backtick (“`”):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the [ANSI_QUOTES \[612\]](#) SQL mode is enabled, it is also permissible to quote identifiers within double quotation marks:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax...
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

The [ANSI_QUOTES \[612\]](#) mode causes the server to interpret double-quoted strings as identifiers. Consequently, when this mode is enabled, string literals must be enclosed within single quotation marks. They cannot be enclosed within double quotation marks. The server SQL mode is controlled as described in [Section 5.1.7, “Server SQL Modes”](#).

Identifier quote characters can be included within an identifier if you quote the identifier. If the character to be included within the identifier is the same as that used to quote the identifier itself, then you need to double the character. The following statement creates a table named `a`b` that contains a column named `c"d`:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
mysql> SELECT 1 AS `one`, 2 AS 'two';
+-----+-----+
| one | two |
+-----+-----+
| 1 | 2 |
+-----+-----+
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal.

It is recommended that you do not use names that begin with *Me* or *MeN*, where *M* and *N* are integers. For example, avoid using `1e` as an identifier, because an expression such as `1e+3` is ambiguous. Depending on context, it might be interpreted as the expression `1e + 3` or as the number `1e+3`.

Be careful when using `MD5()` [1187] to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier. See [Section 9.4, “User-Defined Variables”](#), for more information and examples of workarounds.

Special characters in database and table names are encoded in the corresponding file system names as described in [Section 9.2.3, “Mapping of Identifiers to File Names”](#). If you have databases or tables from an older version of MySQL that contain special characters and for which the underlying directory names or file names have not been updated to use the new encoding, the server displays their names with a prefix of `#mysql150#`. For information about referring to such names or converting them to the newer encoding, see that section.

The following table describes the maximum length for each type of identifier.

Identifier	Maximum Length (characters)
Database	64
Table	64
Column	64
Index	64
Constraint	64
Stored Procedure or Function	64
Trigger	64
View	64
Event	64
Tablespace	64
Server	64
Log File Group	64
Alias	256 (see exception following table)
Compound Statement Label	16

Aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

Identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions that are stored in `.frm` files and to identifiers stored in the grant tables in the `mysql` database. The sizes of the identifier string columns in the grant tables are measured in characters. You can use multi-byte characters without reducing the number of characters permitted for values stored in these columns, something not true prior to MySQL 4.1. As indicated earlier, the permissible Unicode characters are those in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted.

9.2.1. Identifier Qualifiers

MySQL permits names that consist of a single identifier or multiple identifiers. The components of a multiple-part name must be separated by period (“.”) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which the final identifier is interpreted.

In MySQL, you can refer to a table column using any of the following forms.

Column Reference	Meaning
<i>col_name</i>	The column <i>col_name</i> from whichever table used in the statement contains a column of that name.
<i>tbl_name.col_name</i>	The column <i>col_name</i> from table <i>tbl_name</i> of the default database.
<i>db_name.tbl_name.col_name</i>	The column <i>col_name</i> from table <i>tbl_name</i> of the database <i>db_name</i> .

The qualifier character is a separate token and need not be contiguous with the associated identifiers. For example, *tbl_name.col_name* and *tbl_name . col_name* are equivalent.

If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write ``my-table`.`my-column``, not ``my-table.my-column``.

A reserved word that follows a period in a qualified name must be an identifier, so in that context it need not be quoted.

You need not specify a *tbl_name* or *db_name.tbl_name* prefix for a column reference in a statement unless the reference would be ambiguous. Suppose that tables *t1* and *t2* each contain a column *c*, and you retrieve *c* in a `SELECT` statement that uses both *t1* and *t2*. In this case, *c* is ambiguous because it is not unique among the tables used in the statement. You must qualify it with a table name as *t1.c* or *t2.c* to indicate which table you mean. Similarly, to retrieve from a table *t* in database *db1* and from a table *t* in database *db2* in the same statement, you must refer to columns in those tables as *db1.t.col_name* and *db2.t.col_name*.

The syntax `.tbl_name` means the table *tbl_name* in the default database. This syntax is accepted for ODBC compatibility because some ODBC programs prefix table names with a “.” character.

9.2.2. Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage engine). Triggers also correspond to files. Consequently, the case sensitivity of the underlying operating system plays a part in the case sensitivity of database, table, and trigger names. This means such names are not case sensitive in Windows, but are case sensitive in most varieties of Unix. One notable exception is Mac OS X, which is Unix-based but uses a default file system type (HFS+) that is not case sensitive. However, Mac OS X also supports UFS volumes, which are case sensitive just as on any Unix. See [Section 1.8.4, “MySQL Extensions to Standard SQL”](#). The `lower_case_table_names [518]` system variable also affects how the server handles identifier case sensitivity, as described later in this section.



Note

Although database, table, and trigger names are not case sensitive on some platforms, you should not refer to one of these using different cases within the same statement. The following statement would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column, index, stored routine, and event names are not case sensitive on any platform, nor are column aliases.

However, names of logfile groups are case sensitive. This differs from standard SQL.

By default, table aliases are case sensitive on Unix, but not so on Windows or Mac OS X. The following statement would not work on Unix, because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a
-> WHERE a.col_name = 1 OR A.col_name = 2;
```

However, this same statement is permitted on Windows. To avoid problems caused by such differences, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the `lower_case_table_names` [518] system variable, which you can set when starting `mysqld`. `lower_case_table_names` [518] can take the values shown in the following table. This variable does *not* affect case sensitivity of trigger identifiers. On Unix, the default value of `lower_case_table_names` [518] is 0. On Windows the default value is 1. On Mac OS X, the default value is 2.

Value	Meaning
0	Table and database names are stored on disk using the lettercase specified in the <code>CREATE TABLE</code> or <code>CREATE DATABASE</code> statement. Name comparisons are case sensitive. You should <i>not</i> set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or Mac OS X). If you force this variable to 0 with <code>--lower-case-table-names=0</code> [518] on a case-insensitive file system and access <code>MyISAM</code> table names using different lettercases, index corruption may result.
1	Table names are stored in lowercase on disk and name comparisons are not case sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names and table aliases.
2	Table and database names are stored on disk using the lettercase specified in the <code>CREATE TABLE</code> or <code>CREATE DATABASE</code> statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case sensitive. This works <i>only</i> on file systems that are not case sensitive! <code>InnoDB</code> table names are stored in lowercase, as for <code>lower_case_table_names=1</code> .

If you are using MySQL on only one platform, you do not normally have to change the `lower_case_table_names` [518] variable from its default value. However, you may encounter difficulties if you want to transfer tables between platforms that differ in file system case sensitivity. For example, on Unix, you can have two different tables named `my_table` and `MY_TABLE`, but on Windows these two names are considered identical. To avoid data transfer problems arising from lettercase of database or table names, you have two options:

- Use `lower_case_table_names=1` on all systems. The main disadvantage with this is that when you use `SHOW TABLES` or `SHOW DATABASES`, you do not see the names in their original lettercase.
- Use `lower_case_table_names=0` on Unix and `lower_case_table_names=2` on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

Exception: If you are using `InnoDB` tables and you are trying to avoid these data transfer problems, you should set `lower_case_table_names` [518] to 1 on all platforms to force names to be converted to lowercase.

If you plan to set the `lower_case_table_names` [518] system variable to 1 on Unix, you must first convert your old database and table names to lowercase before stopping `mysqld` and restarting it with the new variable setting. To do this for an individual table, use `RENAME TABLE`:

```
RENAME TABLE T1 TO t1;
```

To convert one or more entire databases, dump them before setting `lower_case_table_names` [518], then drop the databases, and reload them after setting `lower_case_table_names` [518]:

1. Use `mysqldump` to dump each database:

```
mysqldump --databases db1 > db1.sql
mysqldump --databases db2 > db2.sql
...
```

Do this for each database that must be recreated.

2. Use `DROP DATABASE` to drop each database.
3. Stop the server, set `lower_case_table_names` [518], and restart the server.
4. Reload the dump file for each database. Because `lower_case_table_names` [518] is set, each database and table name will be converted to lowercase as it is recreated:

```
mysql < db1.sql
mysql < db2.sql
...
```

Object names may be considered duplicates if their uppercase forms are equal according to a binary collation. That is true for names of cursors, conditions, procedures, functions, savepoints, stored routine parameters, stored program local variables, and plugins. It is not true for names of columns, constraints, databases, partitions, statements prepared with `PREPARE`, tables, triggers, users, and user-defined variables.

File system case sensitivity can affect searches in string columns of `INFORMATION_SCHEMA` tables. For more information, see [Section 10.1.7.9, “Collation and INFORMATION_SCHEMA Searches”](#).

9.2.3. Mapping of Identifiers to File Names

There is a correspondence between database and table identifiers and names in the file system. For the basic structure, MySQL represents each database as a directory in the data directory, and each table by one or more files in the appropriate database directory. For the table format files (`.FRM`), the data is always stored in this structure and location.

For the data and index files, the exact representation on disk is storage engine specific. These files may be stored in the same location as the `FRM` files, or the information may be stored in a separate file. `InnoDB` data is stored in the `InnoDB` data files. If you are using tablespaces with `InnoDB`, then the specific tablespace files you create are used instead.

Any character is legal in database or table identifiers except ASCII NUL (`0x00`). MySQL encodes any characters that are problematic in the corresponding file system objects when it creates database directories or table files:

- Basic Latin letters (`a..zA..Z`), digits (`0..9`) and underscore (`_`) are encoded as is. Consequently, their case sensitivity directly depends on file system features.
- All other national letters from alphabets that have uppercase/lowercase mapping are encoded as shown in the following table. Values in the Code Range column are UCS-2 values.

Code Range	Pattern	Number	Used	Unused	Blocks
00C0..017F	[@][0..4][g..z]	5*20= 100	97	3	Latin-1 Supplement + Latin Extended-A
0370..03FF	[@][5..9][g..z]	5*20= 100	88	12	Greek and Coptic
0400..052F	[@][g..z][0..6]	20*7= 140	137	3	Cyrillic + Cyrillic Supplement
0530..058F	[@][g..z][7..8]	20*2= 40	38	2	Armenian

Code Range	Pattern	Number	Used	Unused	Blocks
2160..217F	[@][g..z][9]	20*1= 20	16	4	Number Forms
0180..02AF	[@][g..z][a..k]	20*11=220	203	17	Latin Extended-B + IPA Extensions
1E00..1EFF	[@][g..z][l..r]	20*7= 140	136	4	Latin Extended Additional
1F00..1FFF	[@][g..z][s..z]	20*8= 160	144	16	Greek Extended
....	[@][a..f][g..z]	6*20= 120	0	120	RESERVED
24B6..24E9	[@][@][a..z]	26	26	0	Enclosed Alphanumerics
FF21..FF5A	[@][a..z][@]	26	26	0	Halfwidth and Fullwidth forms

One of the bytes in the sequence encodes lettercase. For example: `LATIN CAPITAL LETTER A WITH GRAVE` is encoded as `@0G`, whereas `LATIN SMALL LETTER A WITH GRAVE` is encoded as `@0g`. Here the third byte (`G` or `g`) indicates lettercase. (On a case-insensitive file system, both letters will be treated as the same.)

For some blocks, such as Cyrillic, the second byte determines lettercase. For other blocks, such as Latin1 Supplement, the third byte determines lettercase. If two bytes in the sequence are letters (as in Greek Extended), the leftmost letter character stands for lettercase. All other letter bytes must be in lowercase.

- All nonletter characters except underscore (`_`), as well as letters from alphabets that do not have uppercase/lowercase mapping (such as Hebrew) are encoded using hexadecimal representation using lowercase letters for hex digits `a..f`:

```
0x003F -> @003f
0xFFFF -> @ffff
```

The hexadecimal values correspond to character values in the `ucs2` double-byte character set.

On Windows, some names such as `nul`, `prn`, and `aux` are encoded by appending `@@@` to the name when the server creates the corresponding file or directory. This occurs on all platforms for portability of the corresponding database object between platforms.

If you have databases or tables from a version of MySQL older than 5.1.6 that contain special characters and for which the underlying directory names or file names have not been updated to use the new encoding, the server displays their names with a prefix of `#mysql50#` in the output from `INFORMATION_SCHEMA` tables or `SHOW` statements. For example, if you have a table named `a@b` and its name encoding has not been updated, `SHOW TABLES` displays it like this:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_test |
+-----+
| #mysql50#a@b   |
+-----+
```

To refer to such a name for which the encoding has not been updated, you must supply the `#mysql50#` prefix:

```
mysql> SHOW COLUMNS FROM `a@b`;
ERROR 1146 (42S02): Table 'test.a@b' doesn't exist

mysql> SHOW COLUMNS FROM `#mysql50#a@b`;
+-----+-----+-----+-----+-----+-----+
```

Field	Type	Null	Key	Default	Extra
i	int(11)	YES		NULL	

To update old names to eliminate the need to use the special prefix to refer to them, re-encode them with `mysqlcheck`. The following command updates all names to the new encoding:

```
shell> mysqlcheck --check-upgrade --fix-db-names --fix-table-names --all-databases
```

To check only specific databases or tables, omit `--all-databases` [303] and provide the appropriate database or table arguments. For information about `mysqlcheck` invocation syntax, see [Section 4.5.3](#), “`mysqlcheck` — A Table Maintenance Program”.



Note

The `#mysql50#` prefix is intended only to be used internally by the server. You should not create databases or tables with names that use this prefix.

Also, `mysqlcheck` cannot fix names that contain literal instances of the `@` character that is used for encoding special characters. If you have databases or tables that contain this character, use `mysqldump` to dump them before upgrading to MySQL 5.1.6 or later, and then reload the dump file after upgrading.

9.2.4. Function Name Parsing and Resolution

MySQL 5.5 supports built-in (native) functions, user-defined functions (UDFs), and stored functions. This section describes how the server recognizes whether the name of a built-in function is used as a function call or as an identifier, and how the server determines which function to use in cases when functions of different types exist with a given name.

Built-In Function Name Parsing

The parser uses default rules for parsing names of built-in functions. These rules can be changed by enabling the `IGNORE_SPACE` [613] SQL mode.

When the parser encounters a word that is the name of a built-in function, it must determine whether the name signifies a function call or is instead a nonexpression reference to an identifier such as a table or column name. For example, in the following statements, the first reference to `count` is a function call, whereas the second reference is a table name:

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

The parser should recognize the name of a built-in function as indicating a function call only when parsing what is expected to be an expression. That is, in nonexpression context, function names are permitted as identifiers.

However, some built-in functions have special parsing or implementation considerations, so the parser uses the following rules by default to distinguish whether their names are being used as function calls or as identifiers in nonexpression context:

- To use the name as a function call in an expression, there must be no whitespace between the name and the following “(” parenthesis character.
- Conversely, to use the function name as an identifier, it must not be followed immediately by a parenthesis.

The requirement that function calls be written with no whitespace between the name and the parenthesis applies only to the built-in functions that have special considerations. `COUNT` is one such name. The exact

list of function names for which following whitespace determines their interpretation are those listed in the `sql_functions[]` array of the `sql/lex.h` source file. Before MySQL 5.1, these are rather numerous (about 200), so you may find it easiest to treat the no-whitespace requirement as applying to all function calls. In MySQL 5.1 and later, parser improvements reduce to about 30 the number of affected function names.

For functions not listed in the `sql_functions[]` array, whitespace does not matter. They are interpreted as function calls only when used in expression context and may be used freely as identifiers otherwise. `ASCII` is one such name. However, for these nonaffected function names, interpretation may vary in expression context: `func_name ()` is interpreted as a built-in function if there is one with the given name; if not, `func_name ()` is interpreted as a user-defined function or stored function if one exists with that name.

The `IGNORE_SPACE` [613] SQL mode can be used to modify how the parser treats function names that are whitespace-sensitive:

- With `IGNORE_SPACE` [613] disabled, the parser interprets the name as a function call when there is no whitespace between the name and the following parenthesis. This occurs even when the function name is used in nonexpression context:

```
mysql> CREATE TABLE count(i INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'count(i INT)'
```

To eliminate the error and cause the name to be treated as an identifier, either use whitespace following the name or write it as a quoted identifier (or both):

```
CREATE TABLE count (i INT);
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

- With `IGNORE_SPACE` [613] enabled, the parser loosens the requirement that there be no whitespace between the function name and the following parenthesis. This provides more flexibility in writing function calls. For example, either of the following function calls are legal:

```
SELECT COUNT(*) FROM mytable;
SELECT COUNT (*) FROM mytable;
```

However, enabling `IGNORE_SPACE` [613] also has the side effect that the parser treats the affected function names as reserved words (see [Section 9.3, “Reserved Words”](#)). This means that a space following the name no longer signifies its use as an identifier. The name can be used in function calls with or without following whitespace, but causes a syntax error in nonexpression context unless it is quoted. For example, with `IGNORE_SPACE` [613] enabled, both of the following statements fail with a syntax error because the parser interprets `count` as a reserved word:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

To use the function name in nonexpression context, write it as a quoted identifier:

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

To enable the `IGNORE_SPACE` [613] SQL mode, use this statement:

```
SET sql_mode = 'IGNORE_SPACE';
```

`IGNORE_SPACE` [613] is also enabled by certain other composite modes such as `ANSI` [617] that include it in their value:

```
SET sql_mode = 'ANSI';
```

Check [Section 5.1.7, “Server SQL Modes”](#), to see which composite modes enable `IGNORE_SPACE` [613].

To minimize the dependency of SQL code on the `IGNORE_SPACE` [613] setting, use these guidelines:

- Avoid creating UDFs or stored functions that have the same name as a built-in function.
- Avoid using function names in nonexpression context. For example, these statements use `count` (one of the affected function names affected by `IGNORE_SPACE` [613]), so they fail with or without whitespace following the name if `IGNORE_SPACE` [613] is enabled:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

If you must use a function name in nonexpression context, write it as a quoted identifier:

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

The number of function names affected by `IGNORE_SPACE` [613] was reduced significantly in MySQL 5.1.13, from about 200 to about 30. As of MySQL 5.1.13, only the following functions are still affected by the `IGNORE_SPACE` [613] setting.

ADDDATE	BIT_AND	BIT_OR	BIT_XOR
CAST	COUNT	CURDATE	CURTIME
DATE_ADD	DATE_SUB	EXTRACT	GROUP_CONCAT
MAX	MID	MIN	NOW
POSITION	SESSION_USER	STD	STDDEV
STDDEV_POP	STDDEV_SAMP	SUBDATE	SUBSTR
SUBSTRING	SUM	SYSDATE	SYSTEM_USER
TRIM	VARIANCE	VAR_POP	VAR_SAMP

For earlier versions of MySQL, check the contents of the `sql_functions[]` array in the `sql/lex.h` source file to see which functions are affected by `IGNORE_SPACE` [613].

Incompatibility warning: The change in MySQL 5.1.13 that reduces the number of function names affected by `IGNORE_SPACE` [613] improves the consistency of parser operation. However, it also introduces the possibility of incompatibility for old SQL code that relies on the following conditions:

- `IGNORE_SPACE` [613] is disabled.
- The presence or absence of whitespace following a function name is used to distinguish between a built-in function and stored function that have the same name, such as `PI()` [1127] versus `PI ()`.

For functions that are no longer affected by `IGNORE_SPACE` [613] as of MySQL 5.1.13, that strategy no longer works. Either of the following approaches can be used if you have code that is subject to the preceding incompatibility:

- If a stored function has a name that conflicts with a built-in function, refer to the stored function with a schema name qualifier, regardless of whether whitespace is present. For example, write `schema_name.PI()` or `schema_name.PI ()`.
- Alternatively, rename the stored function to use a nonconflicting name and change invocations of the function to use the new name.

Function Name Resolution

The following rules describe how the server resolves references to function names for function creation and invocation:

- Built-in functions and user-defined functions

An error occurs if you try to create a UDF with the same name as a built-in function.

- Built-in functions and stored functions

It is possible to create a stored function with the same name as a built-in function, but to invoke the stored function it is necessary to qualify it with a schema name. For example, if you create a stored function named `PI` in the `test` schema, you invoke it as `test.PI()` because the server resolves `PI()` [1127] as a reference to the built-in function. The server creates a warning if the stored function name collides with a built-in function name. The warning can be displayed with `SHOW WARNINGS`.

- User-defined functions and stored functions

User-defined functions and stored functions share the same namespace, so you cannot create a UDF and a stored function with the same name.

The preceding function name resolution rules have implications for upgrading to versions of MySQL that implement new built-in functions:

- If you have already created a user-defined function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name.
- If a new version of MySQL implements a built-in function with the same name as an existing stored function, you have two choices: Rename the stored function to use a nonconflicting name, or change calls to the function so that they use a schema qualifier (that is, use `schema_name.func_name()` syntax).

9.3. Reserved Words

Certain words such as `SELECT`, `DELETE`, or `BIGINT` are reserved and require special treatment for use as identifiers such as table and column names. This may also be true for the names of built-in functions.

Reserved words are permitted as identifiers if you quote them as described in [Section 9.2, “Schema Object Names”](#):

```
mysql> CREATE TABLE interval (begin INT, end INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'
```

```
mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved:

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Names of built-in functions are permitted as identifiers but may require care to be used as such. For example, `COUNT` is acceptable as a column name. However, by default, no whitespace is permitted in function invocations between the function name and the following “(” character. This requirement enables the parser to distinguish whether the name is used in a function call or in nonfunction context. For further detail on recognition of function names, see [Section 9.2.4, “Function Name Parsing and Resolution”](#).

The words in the following table are explicitly reserved in MySQL 5.5. In addition, `_FILENAME` is reserved. At some point, you might upgrade to a higher version, so it is a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the words in the table are forbidden by standard SQL as column or table names (for example, `GROUP`). A few are reserved because MySQL needs them and uses a `yacc` parser. A reserved word can be used as an identifier if you quote it.

For a more detailed list of reserved words, including differences between versions, see [Reserved Words in MySQL 5.5](#).

Table 9.2. Reserved Words in MySQL 5.5.29

ACCESSIBLE	ADD	ALL
ALTER	ANALYZE	AND
AS	ASC	ASENSITIVE
BEFORE	BETWEEN	BIGINT
BINARY	BLOB	BOTH
BY	CALL	CASCADE
CASE	CHANGE	CHAR
CHARACTER	CHECK	COLLATE
COLUMN	CONDITION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FLOAT4
FLOAT8	FOR	FORCE
FOREIGN	FROM	FULLTEXT
GRANT	GROUP	HAVING
HIGH_PRIORITY	HOURL_MICROSECOND	HOURL_MINUTE
HOURL_SECOND	IF	IGNORE
IN	INDEX	INFILE
INNER	INOUT	INSENSITIVE
INSERT	INT	INT1

Reserved Words

INT2	INT3	INT4
INT8	INTEGER	INTERVAL
INTO	IS	ITERATE
JOIN	KEY	KEYS
KILL	LEADING	LEAVE
LEFT	LIKE	LIMIT
LINEAR	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOOP	LOW_PRIORITY	MASTER_SSL_VERIFY_SERVER_CERT
MATCH	MAXVALUE	MEDIUMBLOB
MEDIUMINT	MEDIUMTEXT	MIDDLEINT
MINUTE_MICROSECOND	MINUTE_SECOND	MOD
MODIFIES	NATURAL	NOT
NO_WRITE_TO_BINLOG	NULL	NUMERIC
ON	OPTIMIZE	OPTION
OPTIONALLY	OR	ORDER
OUT	OUTER	OUTFILE
PRECISION	PRIMARY	PROCEDURE
PURGE	RANGE	READ
READS	READ_WRITE	REAL
REFERENCES	REGEXP	RELEASE
RENAME	REPEAT	REPLACE
REQUIRE	RESIGNAL	RESTRICT
RETURN	REVOKE	RIGHT
RLIKE	SCHEMA	SCHEMAS
SECOND_MICROSECOND	SELECT	SENSITIVE
SEPARATOR	SET	SHOW
SIGNAL	SMALLINT	SPATIAL
SPECIFIC	SQL	SQLEXCEPTION
SQLSTATE	SQLWARNING	SQL_BIG_RESULT
SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT	SSL
STARTING	STRAIGHT_JOIN	TABLE
TERMINATED	THEN	TINYBLOB
TINYINT	TINYTEXT	TO
TRAILING	TRIGGER	TRUE
UNDO	UNION	UNIQUE
UNLOCK	UNSIGNED	UPDATE
USAGE	USE	USING

UTC_DATE	UTC_TIME	UTC_TIMESTAMP
VALUES	VARBINARY	VARCHAR
VARCHARACTER	VARYING	WHEN
WHERE	WHILE	WITH
WRITE	XOR	YEAR_MONTH
ZEROFILL		

Table 9.3. New Reserved Words in MySQL 5.5

GENERAL	IGNORE_SERVER_IDS	MASTER_HEARTBEAT_PERIOD
MAXVALUE	RESIGNAL	SIGNAL
SLOW		

MySQL permits some keywords to be used as unquoted identifiers because many people previously used them. Examples are those in the following list:

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

9.4. User-Defined Variables

You can store a value in a user-defined variable in one statement and then refer to it later in another statement. This enables you to pass values from one statement to another. *User-defined variables are connection-specific.* That is, a user variable defined by one client cannot be seen or used by other clients. All variables for a given client connection are automatically freed when that client exits.

User variables are written as `@var_name`, where the variable name `var_name` consists of alphanumeric characters, “.”, “_”, and “\$”. A user variable name can contain other characters if you quote it as a string or identifier (for example, `@'my-var'`, `@"my-var"`, or `@`my-var``).

User variable names are not case sensitive in MySQL 5.0 and up.

One way to set a user-defined variable is by issuing a `SET` statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For `SET`, either `=` [1096] or `:=` [1095] can be used as the assignment operator.

You can also assign a value to a user variable in statements other than `SET`. In this case, the assignment operator must be `:=` [1095] and not `=` [1096] because the latter is treated as the comparison operator `=` [1089] in non-`SET` statements:

```
mysql> SET @t1=1, @t2=2, @t3:=4;
mysql> SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;
+-----+-----+-----+-----+
| @t1 | @t2 | @t3 | @t4 := @t1+@t2+@t3 |
+-----+-----+-----+
| 1 | 2 | 4 | 7 |
+-----+-----+-----+

```

User variables can be assigned a value from a limited set of data types: integer, decimal, floating-point, binary or nonbinary string, or `NULL` value. Assignment of decimal and real values does not preserve the precision or scale of the value. A value of a type other than one of the permissible types is converted to a permissible type. For example, a value having a temporal or spatial data type is converted to a binary string.

If a user variable is assigned a nonbinary (character) string value, it has the same character set and collation as the string. The coercibility of user variables is implicit. (This is the same coercibility as for table column values.)

Bit values assigned to user variables are treated as binary strings. To assign a bit value as a number to a user variable, use `CAST() [1168]` or `+0`:

```
mysql> SET @v1 = b'1000001';
mysql> SET @v2 = CAST(b'1000001' AS UNSIGNED), @v3 = b'1000001'+0;
mysql> SELECT @v1, @v2, @v3;
+-----+-----+-----+
| @v1 | @v2 | @v3 |
+-----+-----+-----+
| A | 65 | 65 |
+-----+-----+-----+

```

If the value of a user variable is selected in a result set, it is returned to the client as a string.

If you refer to a variable that has not been initialized, it has a value of `NULL` and a type of string.

User variables may be used in most contexts where expressions are permitted. This does not currently include contexts that explicitly require a literal value, such as in the `LIMIT` clause of a `SELECT` statement, or the `IGNORE N LINES` clause of a `LOAD DATA` statement.

As a general rule, you should never assign a value to a user variable and read the value within the same statement. You might get the results you expect, but this is not guaranteed. The order of evaluation for expressions involving user variables is undefined and may change based on the elements contained within a given statement; in addition, this order is not guaranteed to be the same between releases of the MySQL Server. In `SELECT @a, @a:=@a+1, ...`, you might think that MySQL will evaluate `@a` first and then do an assignment second. However, changing the statement (for example, by adding a `GROUP BY`, `HAVING`, or `ORDER BY` clause) may cause MySQL to select an execution plan with a different order of evaluation.

Another issue with assigning a value to a variable and reading the value within the same statement is that the default result type of a variable is based on its type at the start of the statement. The following example illustrates this:

```
mysql> SET @a='test';
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

For this `SELECT` statement, MySQL reports to the client that column one is a string and converts all accesses of `@a` to strings, even though `@a` is set to a number for the second row. After the `SELECT` statement executes, `@a` is regarded as a number for the next statement.

To avoid problems with this behavior, either do not assign a value to and read the value of the same variable within a single statement, or else set the variable to `0`, `0.0`, or `'` to define its type before you use it.

In a `SELECT` statement, each select expression is evaluated only when sent to the client. This means that in a `HAVING`, `GROUP BY`, or `ORDER BY` clause, referring to a variable that is assigned a value in the select expression list does *not* work as expected:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

The reference to `b` in the `HAVING` clause refers to an alias for an expression in the select list that uses `@aa`. This does not work as expected: `@aa` contains the value of `id` from the previous selected row, not from the current row.

User variables are intended to provide data values. They cannot be used directly in an SQL statement as an identifier or as part of an identifier, such as in contexts where a table or database name is expected, or as a reserved word such as `SELECT`. This is true even if the variable is quoted, as shown in the following example:

```
mysql> SELECT c1 FROM t;
+-----+
| c1 |
+-----+
| 0 |
+-----+
| 1 |
+-----+
2 rows in set (0.00 sec)

mysql> SET @col = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+-----+
| @col |
+-----+
| c1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT `@col` FROM t;
ERROR 1054 (42S22): Unknown column '@col' in 'field list'

mysql> SET @col = "`c1`";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+-----+
| @col |
+-----+
| `c1` |
+-----+
1 row in set (0.00 sec)
```

An exception to this principle that user variables cannot be used to provide identifiers is that if you are constructing a string for use as a prepared statement to be executed later. In this case, user variables can be used to provide any part of the statement. The following example illustrates how this can be done:

```
mysql> SET @c = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SET @s = CONCAT("SELECT ", @c, " FROM t");
Query OK, 0 rows affected (0.00 sec)

mysql> PREPARE stmt FROM @s;
Query OK, 0 rows affected (0.04 sec)
Statement prepared
```

```
mysql> EXECUTE stmt;
+-----+
| c1 |
+-----+
| 0 |
+-----+
| 1 |
+-----+
2 rows in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)
```

See [Section 13.5, “SQL Syntax for Prepared Statements”](#), for more information.

A similar technique can be used in application programs to construct SQL statements using program variables, as shown here using PHP 5:

```
<?php
    $mysqli = new mysqli("localhost", "user", "pass", "test");

    if( mysqli_connect_errno() )
        die("Connection failed: %s\n", mysqli_connect_error());

    $col = "c1";

    $query = "SELECT $col FROM t";

    $result = $mysqli->query($query);

    while($row = $result->fetch_assoc())
    {
        echo "<p>" . $row["$col"] . "</p>\n";
    }

    $result->close();

    $mysqli->close();
?>
```

Assembling an SQL statement in this fashion is sometimes known as “Dynamic SQL”.

9.5. Expression Syntax

The following rules define expression syntax in MySQL. The grammar shown here is based on that given in the `sql/sql_yacc.yy` file of MySQL source distributions. See the notes after the grammar for additional information about some of the terms. Operator precedence is given in [Section 12.3.1, “Operator Precedence”](#).

```
expr:
    expr OR expr
  | expr || expr
  | expr XOR expr
  | expr AND expr
  | expr && expr
  | NOT expr
  | ! expr
  | boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
  | boolean_primary

boolean_primary:
    boolean_primary IS [NOT] NULL
  | boolean_primary <=> predicate
  | boolean_primary comparison_operator predicate
```

```

| boolean_primary comparison_operator {ALL | ANY} (subquery)
| predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:
| bit_expr [NOT] IN (subquery)
| bit_expr [NOT] IN (expr [, expr] ...)
| bit_expr [NOT] BETWEEN bit_expr AND predicate
| bit_expr SOUNDS LIKE bit_expr
| bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]
| bit_expr [NOT] REGEXP bit_expr
| bit_expr

bit_expr:
| bit_expr | bit_expr
| bit_expr & bit_expr
| bit_expr << bit_expr
| bit_expr >> bit_expr
| bit_expr + bit_expr
| bit_expr - bit_expr
| bit_expr * bit_expr
| bit_expr / bit_expr
| bit_expr DIV bit_expr
| bit_expr MOD bit_expr
| bit_expr % bit_expr
| bit_expr ^ bit_expr
| bit_expr + interval_expr
| bit_expr - interval_expr
| simple_expr

simple_expr:
| literal
| identifier
| function_call
| simple_expr COLLATE collation_name
| param_marker
| variable
| simple_expr || simple_expr
| + simple_expr
| - simple_expr
| ~ simple_expr
| ! simple_expr
| BINARY simple_expr
| (expr [, expr] ...)
| ROW (expr, expr [, expr] ...)
| (subquery)
| EXISTS (subquery)
| {identifier expr}
| match_expr
| case_expr
| interval_expr

```

Notes:

For literal value syntax, see [Section 9.1, “Literal Values”](#).

For identifier syntax, see [Section 9.2, “Schema Object Names”](#).

Variables can be user variables, system variables, or stored program local variables or parameters:

- User variables: [Section 9.4, “User-Defined Variables”](#)
- System variables: [Section 5.1.5, “Using System Variables”](#)
- Local variables: [Section 13.6.4.1, “Local Variable DECLARE Syntax”](#)

- Parameters: [Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#)

param_marker is '?' as used in prepared statements for placeholders. See [Section 13.5.1, “PREPARE Syntax”](#).

(*subquery*) indicates a subquery that returns a single value; that is, a scalar subquery. See [Section 13.2.10.1, “The Subquery as Scalar Operand”](#).

{*identifier expr*} is ODBC escape syntax and is accepted for ODBC compatibility. The value is *expr*. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

match_expr indicates a `MATCH` [1153] expression. See [Section 12.9, “Full-Text Search Functions”](#).

case_expr indicates a `CASE` [1097] expression. See [Section 12.4, “Control Flow Functions”](#).

interval_expr represents a time interval. The syntax is `INTERVAL expr unit`, where *unit* is a specifier such as `hour`, `day`, or `week`. For the full list of *unit* specifiers, see the description of the `DATE_ADD()` [1135] function in [Section 12.7, “Date and Time Functions”](#).

The meaning of some operators depends on the SQL mode:

- By default, `||` [1095] is a logical `OR` [1095] operator. With `PIPES_AS_CONCAT` [616] enabled, `||` [1095] is string concatenation, with a precedence between `^` [1182] and the unary operators.
- By default, `!` [1094] has a higher precedence than `NOT`. With `HIGH_NOT_PRECEDENCE` [613] enabled, `!` [1094] and `NOT` have the same precedence.

See [Section 5.1.7, “Server SQL Modes”](#).

9.6. Comment Syntax

MySQL Server supports three comment styles:

- From a “#” character to the end of the line.
- From a “-- ” sequence to the end of the line. In MySQL, the “-- ” (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in [Section 1.8.5.5, “--' as the Start of a Comment”](#).
- From a `/*` sequence to the following `*/` sequence, as in the C programming language. This syntax enables a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;      # This comment continues to the end of line
mysql> SELECT 1+1;      -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

Nested comments are not supported. (Under some conditions, nested comments might be permitted, but usually are not, and users should avoid them.)

MySQL Server supports some variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the “!” character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The comment syntax just described applies to how the `mysqld` server parses SQL statements. The `mysql` client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.)

Comments in this format, `/*!12345 ... */`, are not stored on the server. If this format is used to comment stored routines, the comments will not be retained on the server.

The use of short-form `mysql` commands such as `\C` within multi-line `/* ... */` comments is not supported.

Chapter 10. Globalization

Table of Contents

10.1. Character Set Support	963
10.1.1. Character Sets and Collations in General	964
10.1.2. Character Sets and Collations in MySQL	965
10.1.3. Specifying Character Sets and Collations	966
10.1.4. Connection Character Sets and Collations	973
10.1.5. Configuring the Character Set and Collation for Applications	976
10.1.6. Character Set for Error Messages	977
10.1.7. Collation Issues	979
10.1.8. String Repertoire	987
10.1.9. Operations Affected by Character Set Support	989
10.1.10. Unicode Support	992
10.1.11. Upgrading from Previous to Current Unicode Support	996
10.1.12. UTF-8 for Metadata	998
10.1.13. Column Character Set Conversion	1000
10.1.14. Character Sets and Collations That MySQL Supports	1001
10.2. Setting the Error Message Language	1013
10.3. Adding a Character Set	1014
10.3.1. Character Definition Arrays	1016
10.3.2. String Collating Support for Complex Character Sets	1017
10.3.3. Multi-Byte Character Support for Complex Character Sets	1017
10.4. Adding a Collation to a Character Set	1017
10.4.1. Collation Implementation Types	1018
10.4.2. Choosing a Collation ID	1020
10.4.3. Adding a Simple Collation to an 8-Bit Character Set	1021
10.4.4. Adding a UCA Collation to a Unicode Character Set	1022
10.5. Character Set Configuration	1025
10.6. MySQL Server Time Zone Support	1026
10.6.1. Staying Current with Time Zone Changes	1028
10.6.2. Time Zone Leap Second Support	1030
10.7. MySQL Server Locale Support	1031

This chapter covers issues of globalization, which includes internationalization (MySQL's capabilities for adapting to local use) and localization (selecting particular local conventions):

- MySQL support for character sets in SQL statements.
- How to configure the server to support different character sets.
- Selecting the language for error messages.
- How to set the server's time zone and enable per-connection time zone support.
- Selecting the locale for day and month names.

10.1. Character Set Support

MySQL includes character set support that enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. You can specify character sets at the server,

database, table, and column level. MySQL supports the use of character sets for the [MyISAM](#), [MEMORY](#), [NDBCLUSTER](#), and [InnoDB](#) storage engines.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment.
- Syntax for specifying character sets and collations.
- Affected functions and operations.
- Unicode support.
- The character sets and collations that are available, with notes.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the [utf8](#) Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about configuring character sets for application use and character set-related issues in client/server communication, see [Section 10.1.5, “Configuring the Character Set and Collation for Applications”](#), and [Section 10.1.4, “Connection Character Sets and Collations”](#).

10.1.1. Character Sets and Collations in General

A *character set* is a set of symbols and encodings. A *collation* is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: “A”, “B”, “a”, “b”. We give each letter a number: “A” = 0, “B” = 1, “a” = 2, “b” = 3. The letter “A” is a symbol, the number 0 is the **encoding** for “A”, and the combination of all four letters and their encodings is a **character set**.

Suppose that we want to compare two string values, “A” and “B”. The simplest way to do this is to look at the encodings: 0 for “A” and 1 for “B”. Because 0 is less than 1, we say “A” is less than “B”. What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): “compare the encodings.” We call this simplest of all possible collations a *binary* collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters “a” and “b” as equivalent to “A” and “B”; (2) then compare the encodings. We call this a *case-insensitive* collation. It is a little more complex than a binary collation.

In real life, most character sets have many characters: not just “A” and “B” but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an “accent” is a mark attached to a character as in German “ö”), and for multiple-character mappings (such as the rule that “ö” = “oe” in one of the two German collations).

MySQL can do these things for you:

- Store strings using a variety of character sets.

- Compare strings using a variety of collations.
- Mix strings with different character sets or collations in the same server, the same database, or even the same table.
- Enable specification of character set and collation at any level.

In these respects, MySQL is far ahead of most other database management systems. However, to use these features effectively, you need to know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

10.1.2. Character Sets and Collations in MySQL

The MySQL server can support multiple character sets. To list the available character sets, use the `SHOW CHARACTER SET` statement. A partial listing follows. For more complete information, see [Section 10.1.14, “Character Sets and Collations That MySQL Supports”](#).

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1
euckr	EUC-KR Korean	euckr_korean_ci	2
koi8u	KOI8-U Ukrainian	koi8u_general_ci	1
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
greek	ISO 8859-7 Greek	greek_general_ci	1
cp1250	Windows Central European	cp1250_general_ci	1
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
...			

Any given character set always has at least one collation. It may have several collations. To list the collations for a character set, use the `SHOW COLLATION` statement. For example, to see the collations for the `latin1` (cp1252 West European) character set, use this statement to find those collation names that begin with `latin1`:

```
mysql> SHOW COLLATION LIKE 'latin1%';
```

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5			0
latin1_swedish_ci	latin1	8	Yes	Yes	1
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	1
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0

The `latin1` collations have the following meanings.

Collation	Meaning
<code>latin1_german1_ci</code>	German DIN-1
<code>latin1_swedish_ci</code>	Swedish/Finnish
<code>latin1_danish_ci</code>	Danish/Norwegian
<code>latin1_german2_ci</code>	German DIN-2
<code>latin1_bin</code>	Binary according to <code>latin1</code> encoding
<code>latin1_general_ci</code>	Multilingual (Western European)
<code>latin1_general_cs</code>	Multilingual (ISO Western European), case sensitive
<code>latin1_spanish_ci</code>	Modern Spanish

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has one collation that is the *default collation*. For example, the default collation for `latin1` is `latin1_swedish_ci`. The output for `SHOW CHARACTER SET` indicates which collation is the default for each displayed character set.
- There is a convention for collation names: They start with the name of the character set with which they are associated, they usually include a language name, and they end with `_ci` (case insensitive), `_cs` (case sensitive), or `_bin` (binary).

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

Collation-Charts.Org is a useful site for information that shows how one collation compares to another.

10.1.3. Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The description in the following sections may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

`CHARACTER SET` is used in clauses that specify a character set. `CHARSET` can be used as a synonym for `CHARACTER SET`.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about character set-related issues in client/server communication, see [Section 10.1.4, “Connection Character Sets and Collations”](#).

10.1.3.1. Server Character Set and Collation

MySQL Server has a server character set and a server collation. These can be set at server startup on the command line or in an option file and changed at runtime.

Initially, the server character set and collation depend on the options that you use when you start `mysqld`. You can use `--character-set-server` [438] for the character set. Along with it, you can add `--collation-server` [438] for the collation. If you don't specify a character set, that is the same as saying `--character-set-server=latin1` [438]. If you specify only a character set (for example, `latin1`) but not a collation, that is the same as saying `--character-set-server=latin1` [438] `--collation-server=latin1_swedish_ci` [438] because `latin1_swedish_ci` is the default collation for `latin1`. Therefore, the following three commands all have the same effect:

```
shell> mysqld
shell> mysqld --character-set-server=latin1
shell> mysqld --character-set-server=latin1 \
           --collation-server=latin1_swedish_ci
```

One way to change the settings is by recompiling. To change the default server character set and collation when building from sources, use the `DEFAULT_CHARSET` [154] and `DEFAULT_COLLATION` [154] options for `CMake`. For example:

```
shell> cmake . -DDEFAULT_CHARSET=latin1
```

Or:

```
shell> cmake . -DDEFAULT_CHARSET=latin1 \
           -DDEFAULT_COLLATION=latin1_german1_ci
```

Both `mysqld` and `CMake` verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The server character set and collation are used as default values if the database character set and collation are not specified in `CREATE DATABASE` statements. They have no other purpose.

The current server character set and collation can be determined from the values of the `character_set_server` [488] and `collation_server` [489] system variables. These variables can be changed at runtime.

10.1.3.2. Database Character Set and Collation

Every database has a database character set and a database collation. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
  [[DEFAULT] CHARACTER SET charset_name]
  [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
  [[DEFAULT] CHARACTER SET charset_name]
  [[DEFAULT] COLLATE collation_name]
```

The keyword `SCHEMA` can be used instead of `DATABASE`.

All database options are stored in a text file named `db.opt` that can be found in the database directory.

The `CHARACTER SET` and `COLLATE` clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Example:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set `X` and collation `Y` are used.
- If `CHARACTER SET X` is specified without `COLLATE`, character set `X` and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with `Y` and collation `Y` are used.
- Otherwise, the server character set and server collation are used.

The database character set and collation are used as default values for table definitions if the table character set and collation are not specified in `CREATE TABLE` statements. The database character set also is used by `LOAD DATA INFILE`. The character set and collation have no other purposes.

The character set and collation for the default database can be determined from the values of the `character_set_database` [487] and `collation_database` [489] system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level system variables, `character_set_server` [488] and `collation_server` [489].

10.1.3.3. Table Character Set and Collation

Every table has a table character set and a table collation. The `CREATE TABLE` and `ALTER TABLE` statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)
  [[DEFAULT] CHARACTER SET charset_name]
  [COLLATE collation_name]]

ALTER TABLE tbl_name
  [[DEFAULT] CHARACTER SET charset_name]
  [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... )
CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set `X` and collation `Y` are used.
- If `CHARACTER SET X` is specified without `COLLATE`, character set `X` and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with `Y` and collation `Y` are used.
- Otherwise, the database character set and collation are used.

The table character set and collation are used as default values for column definitions if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

10.1.3.4. Column Character Set and Collation

Every “character” column (that is, a column of type `CHAR`, `VARCHAR`, or `TEXT`) has a column character set and a column collation. Column definition syntax for `CREATE TABLE` and `ALTER TABLE` has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
  [CHARACTER SET charset_name]
  [COLLATE collation_name]
```

These clauses can also be used for [ENUM](#) and [SET](#) columns:

```
col_name {ENUM | SET} (val_list)
  [CHARACTER SET charset_name]
  [COLLATE collation_name]
```

Examples:

```
CREATE TABLE t1
(
  col1 VARCHAR(5)
    CHARACTER SET latin1
    COLLATE latin1_german1_ci
);

ALTER TABLE t1 MODIFY
  col1 VARCHAR(5)
    CHARACTER SET latin1
    COLLATE latin1_swedish_ci;
```

MySQL chooses the column character set and collation in the following manner:

- If both [CHARACTER SET X](#) and [COLLATE Y](#) are specified, character set [X](#) and collation [Y](#) are used.

```
CREATE TABLE t1
(
  col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The character set and collation are specified for the column, so they are used. The column has character set [utf8](#) and collation [utf8_unicode_ci](#).

- If [CHARACTER SET X](#) is specified without [COLLATE](#), character set [X](#) and its default collation are used.

```
CREATE TABLE t1
(
  col1 CHAR(10) CHARACTER SET utf8
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The character set is specified for the column, but the collation is not. The column has character set [utf8](#) and the default collation for [utf8](#), which is [utf8_general_ci](#). To see the default collation for each character set, use the [SHOW COLLATION](#) statement.

- If [COLLATE Y](#) is specified without [CHARACTER SET](#), the character set associated with [Y](#) and collation [Y](#) are used.

```
CREATE TABLE t1
(
  col1 CHAR(10) COLLATE utf8_polish_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The collation is specified for the column, but the character set is not. The column has collation [utf8_polish_ci](#) and the character set is the one associated with the collation, which is [utf8](#).

- Otherwise, the table character set and collation are used.

```
CREATE TABLE t1
(
  col1 CHAR(10)
```

```
) CHARACTER SET latin1 COLLATE latin1_bin;
```

Neither the character set nor collation are specified for the column, so the table defaults are used. The column has character set `latin1` and collation `latin1_bin`.

The `CHARACTER SET` and `COLLATE` clauses are standard SQL.

If you use `ALTER TABLE` to convert a column from one character set to another, MySQL attempts to map the data values, but if the character sets are incompatible, there may be data loss.

10.1.3.5. Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

A character string literal may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT 'string';
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For the simple statement `SELECT 'string'`, the string has the character set and collation defined by the `character_set_connection` [487] and `collation_connection` [489] system variables.

The `_charset_name` expression is formally called an *introducer*. It tells the parser, “the string that is about to follow uses character set *X*.” Because this has confused people in the past, we emphasize that an introducer does not change the string to the introducer character set like `CONVERT()` [1168] would do. It does not change the string's value, although padding may occur. The introducer is just a signal. An introducer is also legal before standard hex literal and numeric hex literal notation (`x'literal'` and `0xnnnn`), or before bit-field literal notation (`b'literal'` and `0bnnnn`).

Examples:

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
SELECT _latin1 b'1100011';
SELECT _latin1 0b1100011;
```

MySQL determines a literal's character set and collation in the following manner:

- If both `_X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.
- If `_X` is specified but `COLLATE` is not specified, character set *X* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- Otherwise, the character set and collation given by the `character_set_connection` [487] and `collation_connection` [489] system variables are used.

Examples:

- A string with `latin1` character set and `latin1_german1_ci` collation:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- A string with `latin1` character set and its default collation (that is, `latin1_swedish_ci`):

```
SELECT _latin1'Müller';
```

- A string with the connection default character set and collation:

```
SELECT 'Müller';
```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

An introducer indicates the character set for the following string, but does not change now how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection` [487].

The following examples show that escape processing occurs using `character_set_connection` [487] even in the presence of an introducer. The examples use `SET NAMES` (which changes `character_set_connection` [487], as discussed in Section 10.1.4, “Connection Character Sets and Collations”), and display the resulting strings using the `HEX()` [1104] function so that the exact string contents can be seen.

Example 1:

```
mysql> SET NAMES latin1;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+-----+-----+
| HEX('à\n') | HEX(_sjis'à\n') |
+-----+-----+
| E00A      | E00A              |
+-----+-----+
1 row in set (0.00 sec)
```

Here, “à” (hex value `E0`) is followed by “\n”, the escape sequence for newline. The escape sequence is interpreted using the `character_set_connection` [487] value of `latin1` to produce a literal newline (hex value `0A`). This happens even for the second string. That is, the introducer of `_sjis` does not affect the parser's escape processing.

Example 2:

```
mysql> SET NAMES sjis;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+-----+-----+
| HEX('à\n') | HEX(_latin1'à\n') |
+-----+-----+
| E05C6E     | E05C6E             |
+-----+-----+
1 row in set (0.04 sec)
```

Here, `character_set_connection` [487] is `sjis`, a character set in which the sequence of “à” followed by “\n” (hex values `05` and `5C`) is a valid multi-byte character. Hence, the first two bytes of the string are interpreted as a single `sjis` character, and the “\n” is not interpreted as an escape character. The following “n” (hex value `6E`) is not interpreted as part of an escape sequence. This is true even for the second string; the introducer of `_latin1` does not affect escape processing.

10.1.3.6. National Character Set

Standard SQL defines `NCHAR` or `NATIONAL CHAR` as a way to indicate that a `CHAR` column should use some predefined character set. MySQL 5.5 uses `utf8` as this predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
```

```
NATIONAL CHARACTER(10)
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

You can use `N'literal'` (or `n'literal'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For information on upgrading character sets to MySQL 5.5 from versions prior to 4.1, see the *MySQL 3.23, 4.0, 4.1 Reference Manual*.

10.1.3.7. Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table and Column Definition

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that is straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table and Column Definition

```
CREATE TABLE t1
(
  c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1
  DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
```

```
USE d1;
CREATE TABLE t1
(
  c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

10.1.3.8. Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

10.1.4. Connection Character Sets and Collations

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The server character set and collation are the values of the `character_set_server` [488] and `collation_server` [489] system variables.
- The character set and collation of the default database are the values of the `character_set_database` [487] and `collation_database` [489] system variables.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has connection-related character set and collation system variables.

A “connection” is what you make when you connect to the server. The client sends SQL statements, such as queries, over the connection to the server. The server sends responses, such as result sets or error messages, over the connection back to the client. This leads to several questions about character set and collation handling for client connections, each of which can be answered in terms of system variables:

- What character set is the statement in when it leaves the client?

The server takes the `character_set_client` [486] system variable to be the character set in which statements are sent by the client.

- What character set should the server translate a statement to after receiving it?

For this, the server uses the `character_set_connection` [487] and `collation_connection` [489] system variables. It converts statements sent by the client from `character_set_client` [486] to `character_set_connection` [487] (except for string literals that have an introducer such as `_latin1` or `_utf8`). `collation_connection` [489] is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` [489] does not matter because columns have their own collation, which has a higher collation precedence.

- What character set should the server translate to before shipping result sets or error messages back to the client?

The `character_set_results` [488] system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, and result metadata such as column names and error messages.

Clients can fine-tune the settings for these variables, or depend on the defaults (in which case, you can skip the rest of this section). If you do not use the defaults, you must change the character settings *for each connection to the server*.

Two statements affect the connection-related character set variables as a group:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

`SET NAMES` indicates what character set the client will use to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server, “future incoming messages from this client are in character set `cp1251`.” It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement.)

A `SET NAMES 'x'` statement is equivalent to these three statements:

```
SET character_set_client = x;
SET character_set_results = x;
SET character_set_connection = x;
```

Setting `character_set_connection` [487] to `x` also implicitly sets `collation_connection` [489] to the default collation for `x`. It is unnecessary to set that collation explicitly. To specify a particular collation, use the optional `COLLATE` clause:

```
SET NAMES 'charset_name' COLLATE 'collation_name'
```

- `SET CHARACTER SET charset_name`

`SET CHARACTER SET` is similar to `SET NAMES` but sets `character_set_connection` [487] and `collation_connection` [489] to `character_set_database` [487] and `collation_database` [489]. A `SET CHARACTER SET x` statement is equivalent to these three statements:

```
SET character_set_client = x;
SET character_set_results = x;
SET collation_connection = @@collation_database;
```

Setting `collation_connection` [489] also implicitly sets `character_set_connection` [487] to the character set associated with the collation (equivalent to executing `SET character_set_connection = @@character_set_database`). It is unnecessary to set `character_set_connection` [487] explicitly.



Note

`ucs2`, `utf16`, and `utf32` cannot be used as a client character set, which means that they do not work for `SET NAMES` or `SET CHARACTER SET`.

The MySQL client programs `mysql`, `mysqladmin`, `mysqlcheck`, `mysqlimport`, and `mysqlshow` determine the default character set to use as follows:

- In the absence of other information, the programs use the compiled-in default character set, usually `latin1`.
- The programs can autodetect which character set to use based on the operating system setting, such as the value of the `LANG` or `LC_ALL` locale environment variable on Unix systems or the code page setting on Windows systems. For systems on which the locale is available from the OS, the client uses it to set the default character set rather than using the compiled-in default. For example, setting `LANG` to `ru_RU.KOI8-R` causes the `koi8r` character set to be used. Thus, users can configure the locale in their environment for use by MySQL clients.

The OS character set is mapped to the closest MySQL character set if there is no exact match. If the client does not support the matching character set, it uses the compiled-in default. For example, `ucs2` is not supported as a connection character set.

C applications that wish to use character set autodetection based on the OS setting can invoke the following `mysql_options()` call before connecting to the server:

```
mysql_options(mysql,
              MYSQL_SET_CHARSET_NAME,
              MYSQL_AUTODETECT_CHARSET_NAME);
```

- The programs support a `--default-character-set` [277] option, which enables users to specify the character set explicitly to override whatever default the client otherwise determines.



Note

Before MySQL 5.5, in the absence of other information, the MySQL client programs used the compiled-in default character set, usually `latin1`. An implication of this difference is that if your environment is configured to use a non-`latin1` locale, MySQL client programs will use a different connection character set than previously, as though you had issued an implicit `SET NAMES` statement. If the previous behavior is required, start the client with the `--default-character-set=latin1` option.

When a client connects to the server, it sends the name of the character set that it wants to use. The server uses the name to set the `character_set_client` [486], `character_set_results` [488], and `character_set_connection` [487] system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

With the `mysql` client, if you want to use a character set different from the default, you could explicitly execute `SET NAMES` every time you start up. However, to accomplish the same result more easily, you can add the `--default-character-set` [277] option setting to your `mysql` command line or in your option file. For example, the following option file setting changes the three connection-related character set variables set to `koi8r` each time you invoke `mysql`:

```
[mysql]
default-character-set=koi8r
```

If you are using the `mysql` client with auto-reconnect enabled (which is not recommended), it is preferable to use the `charset` command rather than `SET NAMES`. For example:

```
mysql> charset utf8
Charset changed
```

The `charset` command issues a `SET NAMES` statement, and also changes the default character set that `mysql` uses when it reconnects after the connection has dropped.

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say `SET NAMES` or `SET CHARACTER SET`, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say `SET NAMES 'latin1'` or `SET CHARACTER SET latin1` before issuing the `SELECT` statement, the server converts the `latin2` values to `latin1` just before sending results back. Conversion may be lossy if there are characters that are not in both character sets.

If you do not want the server to perform any conversion of result sets or error messages, set `character_set_results` [488] to `NULL` or `binary`:

```
SET character_set_results = NULL;
```

To see the values of the character set and collation system variables that apply to your connection, use these statements:

```
SHOW VARIABLES LIKE 'character_set%';  
SHOW VARIABLES LIKE 'collation%';
```

You must also consider the environment within which your MySQL applications execute. See [Section 10.1.5, “Configuring the Character Set and Collation for Applications”](#).

For more information about character sets and error messages, see [Section 10.1.6, “Character Set for Error Messages”](#).

10.1.5. Configuring the Character Set and Collation for Applications

For applications that store data using the default MySQL character set and collation ([latin1](#), [latin1_swedish_ci](#)), no special configuration should be needed. If applications require data storage using a different character set or collation, you can configure character set information several ways:

- Specify character settings per database. For example, applications that use one database might require [utf8](#), whereas applications that use another database might require [sjis](#).
- Specify character settings at server startup. This causes the server to use the given settings for all applications that do not make other arrangements.
- Specify character settings at configuration time, if you build MySQL from source. This causes the server to use the given settings for all applications, without having to specify them at server startup.

When different applications require different character settings, the per-database technique provides a good deal of flexibility. If most or all applications use the same character set, specifying character settings at server startup or configuration time may be most convenient.

For the per-database or server-startup techniques, the settings control the character set for data storage. Applications must also tell the server which character set to use for client/server communications, as described in the following instructions.

The examples shown here assume use of the [utf8](#) character set and [utf8_general_ci](#) collation.

Specify character settings per database. To create a database such that its tables will use a given default character set and collation for data storage, use a `CREATE DATABASE` statement like this:

```
CREATE DATABASE mydb  
  DEFAULT CHARACTER SET utf8  
  DEFAULT COLLATE utf8_general_ci;
```

Tables created in the database will use [utf8](#) and [utf8_general_ci](#) by default for any character columns.

Applications that use the database should also configure their connection to the server each time they connect. This can be done by executing a `SET NAMES 'utf8'` statement after connecting. The statement can be used regardless of connection method: The `mysql` client, PHP scripts, and so forth.

In some cases, it may be possible to configure the connection to use the desired character set some other way. For example, for connections made using `mysql`, you can specify the `--default-character-set=utf8 [277]` command-line option to achieve the same effect as `SET NAMES 'utf8'`.

For more information about configuring client connections, see [Section 10.1.4, “Connection Character Sets and Collations”](#).

If you change the default character set or collation for a database, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See [Section 13.1.15](#), “[CREATE PROCEDURE](#) and [CREATE FUNCTION Syntax](#)”.)

Specify character settings at server startup. To select a character set and collation at server startup, use the `--character-set-server` [\[438\]](#) and `--collation-server` [\[438\]](#) options. For example, to specify the options in an option file, include these lines:

```
[mysqld]
character-set-server=utf8
collation-server=utf8_general_ci
```

These settings apply server-wide and apply as the defaults for databases created by any application, and for tables created in those databases.

It is still necessary for applications to configure their connection using `SET NAMES` or equivalent after they connect, as described previously. You might be tempted to start the server with the `--init_connect="SET NAMES 'utf8'"` [\[505\]](#) option to cause `SET NAMES` to be executed automatically for each client that connects. However, this will yield inconsistent results because the `init_connect` [\[505\]](#) value is not executed for users who have the `SUPER` [\[697\]](#) privilege.

Specify character settings at MySQL configuration time. To select a character set and collation when you configure and build MySQL from source, use the `DEFAULT_CHARSET` [\[154\]](#) and `DEFAULT_COLLATION` [\[154\]](#) options for `CMake`:

```
shell> cmake . -DDEFAULT_CHARSET=utf8 \
-DDEFAULT_COLLATION=utf8_general_ci
```

The resulting server uses `utf8` and `utf8_general_ci` as the default for databases and tables and for client connections. It is unnecessary to use `--character-set-server` [\[438\]](#) and `--collation-server` [\[438\]](#) to specify those defaults at server startup. It is also unnecessary for applications to configure their connection using `SET NAMES` or equivalent after they connect to the server.

Regardless of how you configure the MySQL character set for application use, you must also consider the environment within which those applications execute. If you will send statements using UTF-8 text taken from a file that you create in an editor, you should edit the file with the locale of your environment set to UTF-8 so that the file encoding is correct and so that the operating system handles it correctly. If you use the `mysql` client from within a terminal window, the window must be configured to use UTF-8 or characters may not display properly. For a script that executes in a Web environment, the script must handle character encoding properly for its interaction with the MySQL server, and it must generate pages that correctly indicate the encoding so that browsers know how to display the content of the pages. For example, you can include this `<meta>` tag within your `<head>` element:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

10.1.6. Character Set for Error Messages

This section describes how the server uses character sets for constructing error messages and returning them to clients. For information about the language of error messages (rather than the character set), see [Section 10.2](#), “[Setting the Error Message Language](#)”.

As of MySQL 5.5, the server constructs error messages using UTF-8 and returns them to clients in the character set specified by the `character_set_results` [\[488\]](#) system variable.

The server constructs error messages as follows:

- The message template uses UTF-8.
- Parameters in the message template are replaced with values that apply to a specific error occurrence:
 - Identifiers such as table or column names use UTF-8 internally so they are copied as is.
 - Character (nonbinary) string values are converted from their character set to UTF-8.
 - Binary string values are copied as is for bytes in the range `0x20` to `0x7E`, and using `\x` hex encoding for bytes outside that range. For example, if a duplicate-key error occurs for an attempt to insert `0x41CF9F` into a `VARBINARY` unique column, the resulting error message uses UTF-8 with some bytes hex encoded:

```
Duplicate entry 'A\xC3\x9F' for key 1
```

To return a message to the client after it has been constructed, the server converts it from UTF-8 to the character set specified by the `character_set_results` [488] system variable. If `character_set_results` [488] has a value of `NULL` or `binary`, no conversion occurs. No conversion occurs if the variable value is `utf8`, either, because that matches the original error message character set.

For characters that cannot be represented in `character_set_results` [488], some encoding may occur during the conversion. The encoding uses Unicode code point values:

- Characters in the Basic Multilingual Plane (BMP) range (`0x0000` to `0xFFFF`) are written using `\nnnn` notation.
- Characters outside the BMP range (`0x01000` to `0x10FFFF`) are written using `\+nnnnnnn` notation.

Clients can set `character_set_results` [488] to control the character set in which they receive error messages. The variable can be set directly, or indirectly by means such as `SET NAMES`. For more information about `character_set_results` [488], see [Section 10.1.4, “Connection Character Sets and Collations”](#).

Prior to MySQL 5.5, the server constructs error messages and returns them to clients as follows:

- The message template has the character set associated with the error message language. For example, English, Korean, and Russian messages use `latin1`, `euckr`, and `koi8r`, respectively.
- Parameters in the message template are replaced with values that apply to a specific error occurrence. These parameters use their own character set. Identifiers such as table or column names use UTF-8. Data values retain their character set. For example, in the following duplicate-key message, `'xxx'` has the character set of the table column associated with key 1:

```
Duplicate entry 'xxx' for key1
```

The preceding method of error-message construction can result in messages that contain a mix of character sets unless all items involved contain only ASCII characters.

For MySQL 5.5 and higher, the encoding that occurs during the conversion to `character_set_results` [488] before returning error messages to clients can result in different message content compared to earlier versions. For example, if an error occurs for an attempt to drop a table named `ペ` (KATAKANA LETTER PE) and `character_set_results` [488] is a character set such as `latin1` that does not contain that character, the resulting message sent to the client has an encoded table name:

```
ERROR 1051 (42S02): Unknown table '\30DA'
```

Before MySQL 5.5, the name is not encoded:

```
ERROR 1051 (42S02): Unknown table '^\'
```

10.1.7. Collation Issues

The following sections discuss various aspects of character set collations.

10.1.7.1. Collation Names

MySQL collation names follow these rules:

- A name ending in `_ci` indicates a case-insensitive collation.
- A name ending in `_cs` indicates a case-sensitive collation.
- A name ending in `_bin` indicates a binary collation. Character comparisons are based on character binary code values.

10.1.7.2. Using `COLLATE` in SQL Statements

With the `COLLATE` clause, you can override whatever the default collation is for a comparison. `COLLATE` may be used in various parts of SQL statements. Here are some examples:

- With `ORDER BY`:

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- With `AS`:

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- With `GROUP BY`:

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- With aggregate functions:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- With `DISTINCT`:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- With `WHERE`:

```
SELECT *
FROM t1
WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
SELECT *
FROM t1
WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- With `HAVING`:

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

10.1.7.3. COLLATE Clause Precedence

The `COLLATE` clause has high precedence (higher than `||` [1095]), so the following two expressions are equivalent:

```
x || y COLLATE z
x || (y COLLATE z)
```

10.1.7.4. Collations Must Be for the Right Character Set

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the `latin2_bin` collation is not legal with the `latin1` character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

10.1.7.5. Collation of Expressions

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column `x`:

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, with multiple operands, there can be ambiguity. For example:

```
SELECT x FROM T WHERE x = 'Y';
```

Should the comparison use the collation of the column `x`, or of the string literal `'Y'`? Both `x` and `'Y'` have collations, so which collation takes precedence?

Standard SQL resolves such questions using what used to be called “coercibility” rules. MySQL assigns coercibility values as follows:

- An explicit `COLLATE` clause has a coercibility of 0. (Not coercible at all.)
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column or a stored routine parameter or local variable has a coercibility of 2.
- A “system constant” (the string returned by functions such as `USER()` [1197] or `VERSION()` [1198]) has a coercibility of 3.
- The collation of a literal has a coercibility of 4.
- `NULL` or an expression that is derived from `NULL` has a coercibility of 5.

MySQL uses coercibility values with the following rules to resolve ambiguities:

- Use the collation with the lowest coercibility value.

- If both sides have the same coercibility, then:
 - If both sides are Unicode, or both sides are not Unicode, it is an error.
 - If one of the sides has a Unicode character set, and another side has a non-Unicode character set, the side with Unicode character set wins, and automatic character set conversion is applied to the non-Unicode side. For example, the following statement does not return an error:

```
SELECT CONCAT(utf8_column, latin1_column) FROM t1;
```

It returns a result that has a character set of `utf8` and the same collation as `utf8_column`. Values of `latin1_column` are automatically converted to `utf8` before concatenating.

- For an operation with operands from the same character set but that mix a `_bin` collation and a `_ci` or `_cs` collation, the `_bin` collation is used. This is similar to how operations that mix nonbinary and binary strings evaluate the operands as binary strings, except that it is for collations rather than data types.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings.

Examples:

Comparison	Collation Used
<code>column1 = 'A'</code>	Use collation of <code>column1</code>
<code>column1 = 'A' COLLATE x</code>	Use collation of <code>'A' COLLATE x</code>
<code>column1 COLLATE x = 'A' COLLATE y</code>	Error

The `COERCIBILITY()` [1191] function can be used to determine the coercibility of a string expression:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
-> 0
mysql> SELECT COERCIBILITY(VERSION());
-> 3
mysql> SELECT COERCIBILITY('A');
-> 4
```

See [Section 12.14, “Information Functions”](#).

For implicit conversion of a numeric or temporal value to a string, such as occurs for the argument `1` in the expression `CONCAT(1, 'abc')` [1102], the result is a character (nonbinary) string that has a character set and collation determined by the `character_set_connection` [487] and `collation_connection` [489] system variables. (Before MySQL 5.5.3, the result is a binary string for which the character set and collation are `binary`). See [Section 12.2, “Type Conversion in Expression Evaluation”](#).

10.1.7.6. The `_bin` and `binary` Collations

This section describes how `_bin` collations for nonbinary strings differ from the `binary` “collation” for binary strings.

Nonbinary strings (as stored in the `CHAR`, `VARCHAR`, and `TEXT` data types) have a character set and collation. A given character set can have several collations, each of which defines a particular sorting and comparison order for the characters in the set. One of these is the binary collation for the character set,

indicated by a `_bin` suffix in the collation name. For example, `latin1` and `utf8` have binary collations named `latin1_bin` and `utf8_bin`.

Binary strings (as stored in the `BINARY`, `VARBINARY`, and `BLOB` data types) have no character set or collation in the sense that nonbinary strings do. (Applied to a binary string, the `CHARSET()` and `COLLATION()` functions both return a value of `binary`.) Binary strings are sequences of bytes and the numeric values of those bytes determine sort order.

The `_bin` collations differ from the `binary` collation in several respects.

The unit for sorting and comparison. Binary strings are sequences of bytes. Sorting and comparison is always based on numeric byte values. Nonbinary strings are sequences of characters, which might be multi-byte. Collations for nonbinary strings define an ordering of the character values for sorting and comparison. For the `_bin` collation, this ordering is based solely on binary code values of the characters (which is similar to ordering for binary strings except that a `_bin` collation must take into account that a character might contain multiple bytes). For other collations, character ordering might take additional factors such as lettercase into account.

Character set conversion. A nonbinary string has a character set and is converted to another character set in many cases, even when the string has a `_bin` collation:

- When assigning column values from another column that has a different character set:

```
UPDATE t1 SET utf8_bin_column=latin1_column;
INSERT INTO t1 (latin1_column) SELECT utf8_bin_column FROM t2;
```

- When assigning column values for `INSERT` or `UPDATE` using a string literal:

```
SET NAMES latin1;
INSERT INTO t1 (utf8_bin_column) VALUES ('string-in-latin1');
```

- When sending results from the server to a client:

```
SET NAMES latin1;
SELECT utf8_bin_column FROM t2;
```

For binary string columns, no conversion occurs. For the preceding cases, the string value is copied byte-wise.

Lettercase conversion. Collations provide information about lettercase of characters, so characters in a nonbinary string can be converted from one lettercase to another, even for `_bin` collations that ignore lettercase for ordering:

```
mysql> SET NAMES latin1 COLLATE latin1_bin;
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT LOWER('aA'), UPPER('zZ');
+-----+-----+
| LOWER('aA') | UPPER('zZ') |
+-----+-----+
| aa          | ZZ          |
+-----+-----+
1 row in set (0.13 sec)
```

The concept of lettercase does not apply to bytes in a binary string. To perform lettercase conversion, the string must be converted to a nonbinary string:

```
mysql> SET NAMES binary;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT LOWER('aA'), LOWER(CONVERT('aA' USING latin1));
```

```

+-----+
| LOWER('aA') | LOWER(CONVERT('aA' USING latin1)) |
+-----+
| aA          | aa                                |
+-----+
1 row in set (0.00 sec)

```

Trailing space handling in comparisons. Nonbinary strings have `PADSPACE` behavior for all collations, including `_bin` collations. Trailing spaces are insignificant in comparisons:

```

mysql> SET NAMES utf8 COLLATE utf8_bin;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a ' = 'a';
+-----+
| 'a ' = 'a' |
+-----+
|           1 |
+-----+
1 row in set (0.00 sec)

```

For binary strings, all characters are significant in comparisons, including trailing spaces:

```

mysql> SET NAMES binary;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a ' = 'a';
+-----+
| 'a ' = 'a' |
+-----+
|           0 |
+-----+
1 row in set (0.00 sec)

```

Trailing space handling for inserts and retrievals. `CHAR(N)` columns store nonbinary strings. Values shorter than `N` characters are extended with spaces on insertion. For retrieval, trailing spaces are removed.

`BINARY(N)` columns store binary strings. Values shorter than `N` bytes are extended with `0x00` bytes on insertion. For retrieval, nothing is removed; a value of the declared length is always returned.

```

mysql> CREATE TABLE t1 (
->   a CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin,
->   b BINARY(10)
-> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t1 VALUES ('a','a');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(a), HEX(b) FROM t1;
+-----+
| HEX(a) | HEX(b) |
+-----+
| 61     | 61000000000000000000 |
+-----+
1 row in set (0.04 sec)

```

10.1.7.7. The `BINARY` Operator

The `BINARY` [1168] operator casts the string following it to a binary string. This is an easy way to force a comparison to be done byte by byte rather than character by character. `BINARY` [1168] also causes trailing spaces to be significant.

```

mysql> SELECT 'a ' = 'A';

```

```

-> 1
mysql> SELECT BINARY 'a' = 'A';
-> 0
mysql> SELECT 'a' = 'a ';
-> 1
mysql> SELECT BINARY 'a' = 'a ';
-> 0

```

`BINARY str` is shorthand for `CAST(str AS BINARY)` [1168].

The `BINARY` attribute in character column definitions has a different effect. A character column defined with the `BINARY` attribute is assigned the binary collation of the column character set. Every character set has a binary collation. For example, the binary collation for the `latin1` character set is `latin1_bin`, so if the table default character set is `latin1`, these two column definitions are equivalent:

```

CHAR(10) BINARY
CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin

```

The effect of `BINARY` as a column attribute differs from its effect prior to MySQL 4.1. Formerly, `BINARY` resulted in a column that was treated as a binary string. A binary string is a string of bytes that has no character set or collation, which differs from a nonbinary character string that has a binary collation. For both types of strings, comparisons are based on the numeric values of the string unit, but for nonbinary strings the unit is the character and some character sets support multi-byte characters. [Section 11.4.2, “The BINARY and VARBINARY Types”](#).

The use of `CHARACTER SET binary` in the definition of a `CHAR`, `VARCHAR`, or `TEXT` column causes the column to be treated as a binary data type. For example, the following pairs of definitions are equivalent:

```

CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB

```

10.1.7.8. Examples of the Effect of Collation

Example 1: Sorting German Umlauts

Suppose that column `X` in table `T` has these `latin1` column values:

```

Muffler
Müller
MX Systems
MySQL

```

Suppose also that the column values are retrieved using the following statement:

```

SELECT X FROM T ORDER BY X COLLATE collation_name;

```

The following table shows the resulting order of the values if we use `ORDER BY` with different collations.

<code>latin1_swedish_ci</code>	<code>latin1_german1_ci</code>	<code>latin1_german2_ci</code>
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems

<code>latin1_swedish_ci</code>	<code>latin1_german1_ci</code>	<code>latin1_german2_ci</code>
MySQL	MySQL	MySQL

The character that causes the different sort orders in this example is the U with two dots over it (ü), which the Germans call “U-umlaut.”

- The first column shows the result of the `SELECT` using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.
- The second column shows the result of the `SELECT` using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the `SELECT` using the German DIN-2 rule, which says that U-umlaut sorts with UE.

Example 2: Searching for German Umlauts

Suppose that you have three tables that differ only by the character set and collation used:

```
mysql> SET NAMES utf8;
mysql> CREATE TABLE german1 (
  ->   c CHAR(10)
  -> ) CHARACTER SET latin1 COLLATE latin1_german1_ci;
mysql> CREATE TABLE german2 (
  ->   c CHAR(10)
  -> ) CHARACTER SET latin1 COLLATE latin1_german2_ci;
mysql> CREATE TABLE germanutf8 (
  ->   c CHAR(10)
  -> ) CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Each table contains two records:

```
mysql> INSERT INTO german1 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO german2 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');
```

Two of the above collations have an `A = Ä` equality, and one has no such equality (`latin1_german2_ci`). For that reason, you'll get these results in comparisons:

```
mysql> SELECT * FROM german1 WHERE c = 'Bär';
+-----+
| c      |
+-----+
| Bar    |
| Bär    |
+-----+
mysql> SELECT * FROM german2 WHERE c = 'Bär';
+-----+
| c      |
+-----+
| Bär    |
+-----+
mysql> SELECT * FROM germanutf8 WHERE c = 'Bär';
+-----+
| c      |
+-----+
| Bar    |
| Bär    |
+-----+
```

This is not a bug but rather a consequence of the sorting properties of `latin1_german1_ci` and `utf8_unicode_ci` (the sorting shown is done according to the German DIN 5007 standard).

10.1.7.9. Collation and `INFORMATION_SCHEMA` Searches

String columns in `INFORMATION_SCHEMA` tables have a collation of `utf8_general_ci`, which is case insensitive. However, searches in `INFORMATION_SCHEMA` string columns are also affected by file system case sensitivity. For values that correspond to objects that are represented in the file system, such as names of databases and tables, searches may be case sensitive if the file system is case sensitive. This section describes how to work around this issue if necessary; see also Bug #34921.

Suppose that a query searches the `SCHEMATA.SCHEMA_NAME` column for the `test` database. On Linux, file systems are case sensitive, so comparisons of `SCHEMATA.SCHEMA_NAME` with `'test'` match, but comparisons with `'TEST'` do not:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
-> WHERE SCHEMA_NAME = 'test';
+-----+
| SCHEMA_NAME |
+-----+
| test        |
+-----+
1 row in set (0.01 sec)

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
-> WHERE SCHEMA_NAME = 'TEST';
Empty set (0.00 sec)
```

On Windows or Mac OS X where file systems are not case sensitive, comparisons match both `'test'` and `'TEST'`:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
-> WHERE SCHEMA_NAME = 'test';
+-----+
| SCHEMA_NAME |
+-----+
| test        |
+-----+
1 row in set (0.00 sec)

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
-> WHERE SCHEMA_NAME = 'TEST';
+-----+
| SCHEMA_NAME |
+-----+
| TEST        |
+-----+
1 row in set (0.00 sec)
```

The value of the `lower_case_table_names` [518] system variable makes no difference in this context.

This behavior occurs because the `utf8_general_ci` collation is not used for `INFORMATION_SCHEMA` queries when searching the file system for database objects. It is a result of optimizations implemented for `INFORMATION_SCHEMA` searches in MySQL. For information about these optimizations, see [Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”](#).

Searches in `INFORMATION_SCHEMA` string columns for values that refer to `INFORMATION_SCHEMA` itself do use the `utf8_general_ci` collation because `INFORMATION_SCHEMA` is a “virtual” database and is not represented in the file system. For example, comparisons with `SCHEMATA.SCHEMA_NAME` match `'information_schema'` or `'INFORMATION_SCHEMA'` regardless of platform:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
-> WHERE SCHEMA_NAME = 'information_schema';
+-----+
```

```

| SCHEMA_NAME |
+-----+
| information_schema |
+-----+
1 row in set (0.00 sec)

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
      -> WHERE SCHEMA_NAME = 'INFORMATION_SCHEMA';
+-----+
| SCHEMA_NAME |
+-----+
| information_schema |
+-----+
1 row in set (0.00 sec)

```

If the result of a string operation on an `INFORMATION_SCHEMA` column differs from expectations, a workaround is to use an explicit `COLLATE` clause to force a suitable collation ([Section 10.1.7.2, “Using COLLATE in SQL Statements”](#)). For example, to perform a case-insensitive search, use `COLLATE` with the `INFORMATION_SCHEMA` column name:

```

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
      -> WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'test';
+-----+
| SCHEMA_NAME |
+-----+
| test        |
+-----+
1 row in set (0.00 sec)

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
      -> WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'TEST';
+-----+
| SCHEMA_NAME |
+-----+
| test        |
+-----+
1 row in set (0.00 sec)

```

You can also use the `UPPER()` [\[1110\]](#) or `LOWER()` [\[1105\]](#) function:

```

WHERE UPPER(SCHEMA_NAME) = 'TEST'
WHERE LOWER(SCHEMA_NAME) = 'test'

```

Although a case-insensitive comparison can be performed even on platforms with case-sensitive file systems, as just shown, it is not necessarily always the right thing to do. On such platforms, it is possible to have multiple objects with names that differ only in lettercase. For example, tables named `city`, `CITY`, and `City` can all exist simultaneously. Consider whether a search should match all such names or just one and write queries accordingly:

```

WHERE TABLE_NAME COLLATE utf8_bin = 'City'
WHERE TABLE_NAME COLLATE utf8_general_ci = 'city'
WHERE UPPER(TABLE_NAME) = 'CITY'
WHERE LOWER(TABLE_NAME) = 'city'

```

The first of those comparisons (with `utf8_bin`) is case sensitive; the others are not.

10.1.8. String Repertoire

The *repertoire* of a character set is the collection of characters in the set.

String expressions have a repertoire attribute, which can have two values:

- `ASCII`: The expression can contain only characters in the Unicode range `U+0000` to `U+007F`.

- **UNICODE**: The expression can contain characters in the Unicode range `U+0000` to `U+FFFF`.

The **ASCII** range is a subset of **UNICODE** range, so a string with **ASCII** repertoire can be converted safely without loss of information to the character set of any string with **UNICODE** repertoire or to a character set that is a superset of **ASCII**. (All MySQL character sets are supersets of **ASCII** with the exception of `swe7`, which reuses some punctuation characters for Swedish accented characters.) The use of repertoire enables character set conversion in expressions for many cases where MySQL would otherwise return an “illegal mix of collations” error.

The following discussion provides examples of expressions and their repertoires, and describes how the use of repertoire changes string expression evaluation:

- The repertoire for string constants depends on string content:

```
SET NAMES utf8; SELECT 'abc';
SELECT _utf8'def';
SELECT N'MySQL';
```

Although the character set is `utf8` in each of the preceding cases, the strings do not actually contain any characters outside the ASCII range, so their repertoire is **ASCII** rather than **UNICODE**.

- Columns having the `ascii` character set have **ASCII** repertoire because of their character set. In the following table, `c1` has **ASCII** repertoire:

```
CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET ascii);
```

The following example illustrates how repertoire enables a result to be determined in a case where an error occurs without repertoire:

```
CREATE TABLE t1 (
  c1 CHAR(1) CHARACTER SET latin1,
  c2 CHAR(1) CHARACTER SET ascii
);
INSERT INTO t1 VALUES ('a','b');
SELECT CONCAT(c1,c2) FROM t1;
```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
and (ascii_general_ci,IMPLICIT) for operation 'concat'
```

Using repertoire, subset to superset (`ascii` to `latin1`) conversion can occur and a result is returned:

```
+-----+
| CONCAT(c1,c2) |
+-----+
| ab            |
+-----+
```

- Functions with one string argument inherit the repertoire of their argument. The result of `UPPER(_utf8'abc')` [1110] has **ASCII** repertoire because its argument has **ASCII** repertoire.
- For functions that return a string but do not have string arguments and use `character_set_connection` [487] as the result character set, the result repertoire is **ASCII** if `character_set_connection` [487] is `ascii`, and **UNICODE** otherwise:

```
FORMAT(numeric_column, 4);
```

Use of repertoire changes how MySQL evaluates the following example:

```
SET NAMES ascii;
```

```
CREATE TABLE t1 (a INT, b VARCHAR(10) CHARACTER SET latin1);
INSERT INTO t1 VALUES (1, 'b');
SELECT CONCAT(FORMAT(a, 4), b) FROM t1;
```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (ascii_general_ci,COERCIBLE)
and (latin1_swedish_ci,IMPLICIT) for operation 'concat'
```

With repertoire, a result is returned:

```
+-----+
| CONCAT(FORMAT(a, 4), b) |
+-----+
| 1.0000b                |
+-----+
```

- Functions with two or more string arguments use the “widest” argument repertoire for the result repertoire (**UNICODE** is wider than **ASCII**). Consider the following `CONCAT()` [1102] calls:

```
CONCAT(_ucs2 0x0041, _ucs2 0x0042)
CONCAT(_ucs2 0x0041, _ucs2 0x00C2)
```

For the first call, the repertoire is **ASCII** because both arguments are within the range of the `ascii` character set. For the second call, the repertoire is **UNICODE** because the second argument is outside the `ascii` character set range.

- The repertoire for function return values is determined based only on the repertoire of the arguments that affect the result's character set and collation.

```
IF(column1 < column2, 'smaller', 'greater')
```

The result repertoire is **ASCII** because the two string arguments (the second argument and the third argument) both have **ASCII** repertoire. The first argument does not matter for the result repertoire, even if the expression uses string values.

10.1.9. Operations Affected by Character Set Support

This section describes operations that take character set information into account.

10.1.9.1. Result Strings

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, `UPPER(X)` [1110] returns a string whose character string and collation are the same as that of `X`. The same applies for `INSTR()` [1104], `LCASE()` [1104], `LOWER()` [1105], `LTRIM()` [1106], `MID()` [1106], `REPEAT()` [1107], `REPLACE()` [1107], `REVERSE()` [1107], `RIGHT()` [1107], `RPAD()` [1107], `RTRIM()` [1108], `SOUNDEX()` [1108], `SUBSTRING()` [1109], `TRIM()` [1109], `UCASE()` [1110], and `UPPER()` [1110].

Note: The `REPLACE()` [1107] function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has no character set or collation. This can be checked by using the `CHARSET()` [1191] and `COLLATION()` [1191] functions, both of which return `binary` to indicate that their argument is a binary string:

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+-----+-----+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+-----+-----+
| binary              | binary                  |
+-----+-----+
```

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

- If an explicit `COLLATE X` occurs, use `X`.
- If explicit `COLLATE X` and `COLLATE Y` occur, raise an error.
- Otherwise, if all collations are `X`, use `X`.
- Otherwise, the result has no collation.

For example, with `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, the resulting collation is `X`. The same applies for `UNION`, `||` [1095], `CONCAT()` [1102], `ELT()` [1102], `GREATEST()` [1092], `IF()` [1097], and `LEAST()` [1093].

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the `character_set_connection` [487] and `collation_connection` [489] system variables. This applies only to `CAST()` [1168], `CONV()` [1125], `FORMAT()` [1103], `HEX()` [1104], and `SPACE()` [1108].

If you are uncertain about the character set or collation of the result returned by a string function, you can use the `CHARSET()` [1191] or `COLLATION()` [1191] function to find out:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+-----+-----+-----+
| USER()          | CHARSET(USER()) | COLLATION(USER()) |
+-----+-----+-----+
| test@localhost | utf8            | utf8_general_ci  |
+-----+-----+-----+
```

10.1.9.2. CONVERT() and CAST()

`CONVERT()` [1168] provides a way to convert data between different character sets. The syntax is:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
  SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` [1168] is implemented according to the standard SQL specification.

You may also use `CAST()` [1168] to convert a string to a different character set. The syntax is:

```
CAST(character_string AS character_data_type CHARACTER SET charset_name)
```

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

If you use `CAST()` [1168] without specifying `CHARACTER SET`, the resulting character set and collation are defined by the `character_set_connection` [487] and `collation_connection` [489] system variables. If you use `CAST()` [1168] with `CHARACTER SET X`, the resulting character set and collation are `X` and the default collation of `X`.

You may not use a `COLLATE` clause inside a `CONVERT()` [1168] or `CAST()` [1168] call, but you may use it outside. For example, `CAST(... COLLATE ...)` [1168] is illegal, but `CAST(...)
COLLATE ...` [1168] is legal:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

10.1.9.3. `SHOW` Statements and `INFORMATION_SCHEMA`

Several `SHOW` statements provide additional character set information. These include `SHOW CHARACTER SET`, `SHOW COLLATION`, `SHOW CREATE DATABASE`, `SHOW CREATE TABLE` and `SHOW COLUMNS`. These statements are described here briefly. For more information, see [Section 13.7.5, “SHOW Syntax”](#).

`INFORMATION_SCHEMA` has several tables that contain information similar to that displayed by the `SHOW` statements. For example, the `CHARACTER_SETS` and `COLLATIONS` tables contain the information displayed by `SHOW CHARACTER SET` and `SHOW COLLATION`. See [Chapter 20, `INFORMATION_SCHEMA` Tables](#).

The `SHOW CHARACTER SET` statement shows all available character sets. It takes an optional `LIKE` [1111] clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description                | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European      | latin1_swedish_ci | 1      |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1      |
| latin5  | ISO 8859-9 Turkish        | latin5_turkish_ci  | 1      |
| latin7  | ISO 8859-13 Baltic        | latin7_general_ci  | 1      |
+-----+-----+-----+-----+
```

The output from `SHOW COLLATION` includes all available character sets. It takes an optional `LIKE` [1111] clause that indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+-----+
| Collation      | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_german1_ci | latin1  | 5  |         |         | 0       |
| latin1_swedish_ci | latin1  | 8  | Yes     | Yes     | 0       |
| latin1_danish_ci  | latin1  | 15 |         |         | 0       |
| latin1_german2_ci | latin1  | 31 |         | Yes     | 2       |
| latin1_bin        | latin1  | 47 |         | Yes     | 0       |
| latin1_general_ci | latin1  | 48 |         |         | 0       |
| latin1_general_cs | latin1  | 49 |         |         | 0       |
| latin1_spanish_ci | latin1  | 94 |         |         | 0       |
+-----+-----+-----+-----+-----+-----+
```

`SHOW CREATE DATABASE` displays the `CREATE DATABASE` statement that creates a given database:

```
mysql> SHOW CREATE DATABASE test;
+-----+-----+
| Database | Create Database                                     |
+-----+-----+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+-----+-----+
```

If no `COLLATE` clause is shown, the default collation for the character set applies.

`SHOW CREATE TABLE` is similar, but displays the `CREATE TABLE` statement to create a given table. The column definitions indicate any character set specifications, and the table options include character set information.

The `SHOW COLUMNS` statement displays the collations of a table's columns when invoked as `SHOW FULL COLUMNS`. Columns with `CHAR`, `VARCHAR`, or `TEXT` data types have collations. Numeric and other noncharacter types have no collation (indicated by `NULL` as the `Collation` value). For example:

```
mysql> SHOW FULL COLUMNS FROM person\G
***** 1. row *****
  Field: id
  Type: smallint(5) unsigned
  Collation: NULL
  Null: NO
  Key: PRI
  Default: NULL
  Extra: auto_increment
  Privileges: select,insert,update,references
  Comment:
***** 2. row *****
  Field: name
  Type: char(60)
  Collation: latin1_swedish_ci
  Null: NO
  Key:
  Default:
  Extra:
  Privileges: select,insert,update,references
  Comment:
```

The character set is not part of the display but is implied by the collation name.

10.1.10. Unicode Support

The initial implementation of Unicode support (in MySQL 4.1) included two character sets for storing Unicode data:

- `ucs2`, the UCS-2 encoding of the Unicode character set using 16 bits per character.
- `utf8`, a UTF-8 encoding of the Unicode character set using one to three bytes per character.

These two character sets support the characters from the Basic Multilingual Plane (BMP) of Unicode Version 3.0. BMP characters have these characteristics:

- Their code values are between 0 and 65535 (or `U+0000 .. U+FFFF`).
- They can be encoded with a fixed 16-bit word, as in `ucs2`.
- They can be encoded with 8, 16, or 24 bits, as in `utf8`.
- They are sufficient for almost all characters in major languages.

Characters not supported by the aforementioned character sets include supplementary characters that lie outside the BMP. Characters outside the BMP compare as `REPLACEMENT CHARACTER` and convert to `'?'` when converted to a Unicode character set.

As of MySQL 5.5.3, Unicode support includes supplementary characters, which requires new character sets that have a broader range and therefore take more space. The following table shows a brief feature comparison of previous and current Unicode support.

Before MySQL 5.5	MySQL 5.5 and up
All Unicode 3.0 characters	All Unicode 5.0 characters
No supplementary characters	With supplementary characters
<code>ucs2</code> character set, BMP only	No change
<code>utf8</code> character set for up to three bytes, BMP only	No change
	New <code>utf8mb4</code> character set for up to four bytes, BMP or supplemental
	New <code>utf16</code> character set, BMP or supplemental
	New <code>utf32</code> character set, BMP or supplemental

These changes are upward compatible. If you want to use the new character sets, there are potential incompatibility issues for your applications; see [Section 10.1.11, “Upgrading from Previous to Current Unicode Support”](#). That section also describes how to convert tables from `utf8` to the (4-byte) `utf8mb4` character set, and what constraints may apply in doing so.

MySQL 5.5 supports these Unicode character sets:

- `ucs2`, the UCS-2 encoding of the Unicode character set using 16 bits per character.
- `utf16`, the UTF-16 encoding for the Unicode character set; like `ucs2` but with an extension for supplementary characters.
- `utf32`, the UTF-32 encoding for the Unicode character set using 32 bits per character.
- `utf8`, a UTF-8 encoding of the Unicode character set using one to three bytes per character.
- `utf8mb4`, a UTF-8 encoding of the Unicode character set using one to four bytes per character.

`ucs2` and `utf8` support BMP characters. `utf8mb4`, `utf16`, and `utf32` support BMP and supplementary characters.

A similar set of collations is available for each Unicode character set. For example, each has a Danish collation, the names of which are `ucs2_danish_ci`, `utf16_danish_ci`, `utf32_danish_ci`, `utf8_danish_ci`, and `utf8mb4_danish_ci`. All Unicode collations are listed at [Section 10.1.14.1, “Unicode Character Sets”](#), which also describes collation properties for supplementary characters.

Note that although many of the supplementary characters come from East Asian languages, what MySQL 5.5 adds is support for more Japanese and Chinese characters in Unicode character sets, not support for new Japanese and Chinese character sets.

The MySQL implementation of UCS-2, UTF-16, and UTF-32 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of values. Other database systems might use little-endian byte order or a BOM. In such cases, conversion of values will need to be performed when transferring data between those systems and MySQL.

MySQL uses no BOM for UTF-8 values.

Client applications that need to communicate with the server using Unicode should set the client character set accordingly; for example, by issuing a `SET NAMES 'utf8'` statement. `ucs2`, `utf16`, and `utf32` cannot be used as a client character set, which means that they do not work for `SET NAMES` or `SET CHARACTER SET`. (See [Section 10.1.4, “Connection Character Sets and Collations”](#).)

The following sections provide additional detail on the Unicode character sets in MySQL.

10.1.10.1. The `ucs2` Character Set (UCS-2 Unicode Encoding)

In UCS-2, every character is represented by a 2-byte Unicode code with the most significant byte first. For example: `LATIN CAPITAL LETTER A` has the code `0x0041` and it is stored as a 2-byte sequence: `0x00 0x41`. `CYRILLIC SMALL LETTER YERU` (Unicode `0x044B`) is stored as a 2-byte sequence: `0x04 0x4B`. For Unicode characters and their codes, please refer to the [Unicode Home Page](#).

In MySQL, the `ucs2` character set is a fixed-length 16-bit encoding for Unicode BMP characters.

10.1.10.2. The `utf16` Character Set (UTF-16 Unicode Encoding)

The `utf16` character set is the `ucs2` character set with an extension that enables encoding of supplementary characters:

- For a BMP character, `utf16` and `ucs2` have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, `utf16` has a special sequence for representing the character using 32 bits. This is called the “surrogate” mechanism: For a number greater than `0xffff`, take 10 bits and add them to `0xd800` and put them in the first 16-bit word, take 10 more bits and add them to `0xdc00` and put them in the next 16-bit word. Consequently, all supplementary characters require 32 bits, where the first 16 bits are a number between `0xd800` and `0xdbff`, and the last 16 bits are a number between `0xdc00` and `0xdfff`. Examples are in Section [15.5 Surrogates Area](#) of the Unicode 4.0 document.

Because `utf16` supports surrogates and `ucs2` does not, there is a validity check that applies only in `utf16`: You cannot insert a top surrogate without a bottom surrogate, or vice versa. For example:

```
INSERT INTO t (ucs2_column) VALUES (0xd800); /* legal */
INSERT INTO t (utf16_column)VALUES (0xd800); /* illegal */
```

There is no validity check for characters that are technically valid but are not true Unicode (that is, characters that Unicode considers to be “unassigned code points” or “private use” characters or even “illegals” like `0xffff`). For example, since `U+F8FF` is the Apple Logo, this is legal:

```
INSERT INTO t (utf16_column)VALUES (0xf8ff); /* legal */
```

Such characters cannot be expected to mean the same thing to everyone.

Because MySQL must allow for the worst case (that one character requires four bytes) the maximum length of a `utf16` column or index is only half of the maximum length for a `ucs2` column or index. For example, in MySQL 5.5, the maximum length of a `MEMORY` table index key is 3072 bytes, so these statements create tables with the longest permitted indexes for `ucs2` and `utf16` columns:

```
CREATE TABLE tf (s1 VARCHAR(1536) CHARACTER SET ucs2) ENGINE=MEMORY;
CREATE INDEX i ON tf (s1);
CREATE TABLE tg (s1 VARCHAR(768) CHARACTER SET utf16) ENGINE=MEMORY;
CREATE INDEX i ON tg (s1);
```

10.1.10.3. The `utf32` Character Set (UTF-32 Unicode Encoding)

The `utf32` character set is fixed length (like `ucs2` and unlike `utf16`). `utf32` uses 32 bits for every character, unlike `ucs2` (which uses 16 bits for every character), and unlike `utf16` (which uses 16 bits for some characters and 32 bits for others).

`utf32` takes twice as much space as `ucs2` and more space than `utf16`, but `utf32` has the same advantage as `ucs2` that it is predictable for storage: The required number of bytes for `utf32` equals the

number of characters times 4. Also, unlike `utf16`, there are no tricks for encoding in `utf32`, so the stored value equals the code value.

To demonstrate how the latter advantage is useful, here is an example that shows how to determine a `utf8mb4` value given the `utf32` code value:

```
/* Assume code value = 100cc LINEAR B WHEELED CHARIOT */
CREATE TABLE tmp (utf32_col CHAR(1) CHARACTER SET utf32,
                  utf8mb4_col CHAR(1) CHARACTER SET utf8mb4);
INSERT INTO tmp VALUES (0x000100cc,NULL);
UPDATE tmp SET utf8mb4_col = utf32_col;
SELECT HEX(utf32_col),HEX(utf8mb4_col) FROM tmp;
```

MySQL is very forgiving about additions of unassigned Unicode characters or private-use-area characters. There is in fact only one validity check for `utf32`: No code value may be greater than `0x10ffff`. For example, this is illegal:

```
INSERT INTO t (utf32_column) VALUES (0x110000); /* illegal */
```

10.1.10.4. The `utf8` Character Set (3-Byte UTF-8 Unicode Encoding)

UTF-8 (Unicode Transformation Format with 8-bit units) is an alternative way to store Unicode data. It is implemented according to RFC 3629, which describes encoding sequences that take from one to four bytes. (An older standard for UTF-8 encoding, RFC 2279, describes UTF-8 sequences that take from one to six bytes. RFC 3629 renders RFC 2279 obsolete; for this reason, sequences with five and six bytes are no longer used.)

The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a 2-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use 3-byte or 4-byte sequences.

The `utf8` character set is the same in MySQL 5.5 as before 5.5 and has exactly the same characteristics:

- No support for supplementary characters (BMP characters only).
- A maximum of three bytes per multi-byte character.

Exactly the same set of characters is available in `utf8` as in `ucs2`. That is, they have the same repertoire.

10.1.10.5. The `utf8mb3` “Character Set” (Alias for `utf8`)

In a future version of MySQL, it is possible that `utf8` will become the 4-byte `utf8`, and that users who want to indicate 3-byte `utf8` will have to say `utf8mb3`. To avoid some future problems which might occur with replication when master and slave servers have different MySQL versions, it is possible as of MySQL 5.5.3 for users to specify `utf8mb3` in `CHARACTER SET` clauses, and `utf8mb3_collation_substring` in `COLLATE` clauses, where `collation_substring` is `bin`, `czech_ci`, `danish_ci`, `esperanto_ci`, `estonian_ci`, and so forth. For example:

```
CREATE TABLE t (s1 CHAR(1) CHARACTER SET utf8mb3;
SELECT * FROM t WHERE s1 COLLATE utf8mb3_general_ci = 'x';
DECLARE x VARCHAR(5) CHARACTER SET utf8mb3 COLLATE utf8mb3_danish_ci;
SELECT CAST('a' AS CHAR CHARACTER SET utf8) COLLATE utf8_czech_ci;
```

MySQL immediately converts instances of `utf8mb3` in an alias to `utf8`, so in statements such as `SHOW CREATE TABLE` or `SELECT CHARACTER_SET_NAME FROM INFORMATION_SCHEMA.COLUMNS` or `SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLUMNS`, users will see the true name, `utf8` or `utf8_collation_substring`.

The `utf8mb3` alias is valid only in `CHARACTER SET` clauses, and in certain other places. For example, these are legal:

```
mysql> --character-set-server=utf8mb3
SET NAMES 'utf8mb3'; /* and other SET statements that have similar effect */
SELECT _utf8mb3 'a';
```

10.1.10.6. The `utf8mb4` Character Set (4-Byte UTF-8 Unicode Encoding)

The character set named `utf8` uses a maximum of three bytes per character and contains only BMP characters. As of MySQL 5.5.3, the `utf8mb4` character set uses a maximum of four bytes per character supports supplemental characters:

- For a BMP character, `utf8` and `utf8mb4` have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, `utf8` cannot store the character at all, while `utf8mb4` requires four bytes to store it. Since `utf8` cannot store the character at all, you do not have any supplementary characters in `utf8` columns and you need not worry about converting characters or losing data when upgrading `utf8` data from older versions of MySQL.

`utf8mb4` is a superset of `utf8`, so for an operation such as the following concatenation, the result has character set `utf8mb4` and the collation of `utf8mb4_col`:

```
SELECT CONCAT(utf8_col, utf8mb4_col);
```

Similarly, the following comparison in the `WHERE` clause works according to the collation of `utf8mb_col`:

```
SELECT * FROM utf8_tbl, utf8mb4_tbl
WHERE utf8_tbl.utf8_col = utf8mb4_tbl.utf8mb4_col;
```



Tip

To save space with UTF-8, use `VARCHAR` instead of `CHAR`. Otherwise, MySQL must reserve three (or four) bytes for each character in a `CHAR CHARACTER SET utf8` (or `utf8mb4`) column because that is the maximum possible length. For example, MySQL must reserve 40 bytes for a `CHAR(10) CHARACTER SET utf8mb4` column.

10.1.11. Upgrading from Previous to Current Unicode Support

This section describes issues pertaining to Unicode support that you may face when upgrading to MySQL 5.5 from an older MySQL release. It also provides guidelines for downgrading from MySQL 5.5 to an older release.

In most respects, upgrading to MySQL 5.5 should present few problems with regard to Unicode usage, although there are some potential areas of incompatibility. These are the primary areas of concern:

- For the variable-length character data types (`VARCHAR` and the `TEXT` types), the maximum length in characters is less for `utf8mb4` columns than for `utf8` columns.
- For all character data types (`CHAR`, `VARCHAR`, and the `TEXT` types), the maximum number of characters that can be indexed is less for `utf8mb4` columns than for `utf8` columns.

Consequently, if you want to upgrade tables from `utf8` to `utf8mb4` to take advantage of supplementary-character support, it may be necessary to change some column or index definitions.

Tables can be converted from `utf8` to `utf8mb4` by using `ALTER TABLE`. Suppose that a table was originally defined as follows:

```
CREATE TABLE t1 (
  col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  col2 CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL
) CHARACTER SET utf8;
```

The following statement converts `t1` to use `utf8mb4`:

```
ALTER TABLE t1
  DEFAULT CHARACTER SET utf8mb4,
  MODIFY col1 CHAR(10)
    CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  MODIFY col2 CHAR(10)
    CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL;
```

In terms of table content, conversion from `utf8` to `utf8mb4` presents no problems:

- For a BMP character, `utf8` and `utf8mb4` have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, `utf8` cannot store the character at all, while `utf8mb4` requires four bytes to store it. Since `utf8` cannot store the character at all, you do not have any supplementary characters in `utf8` columns and you need not worry about converting characters or losing data when upgrading `utf8` data from older versions of MySQL.

In terms of table structure, the catch when converting from `utf8` to `utf8mb4` is that the maximum length of a column or index key is unchanged in terms of *bytes*. Therefore, it is smaller in terms of *characters* because the maximum length of a character is four bytes instead of three. For the `CHAR`, `VARCHAR`, and `TEXT` data types, watch for these things when converting your MySQL tables:

- Check all definitions of `utf8` columns and make sure they will not exceed the maximum length for the storage engine.
- Check all indexes on `utf8` columns and make sure they will not exceed the maximum length for the storage engine. Sometimes the maximum can change due to storage engine enhancements.

If the preceding conditions apply, you must either reduce the defined length of columns or indexes, or continue to use `utf8` rather than `utf8mb4`.

Here are some examples where structural changes may be needed:

- A `TINYTEXT` column can hold up to 255 bytes, so it can hold up to 85 3-byte or 63 4-byte characters. Suppose that you have a `TINYTEXT` column that uses `utf8` but must be able to contain more than 63 characters. You cannot convert it to `utf8mb4` unless you also change the data type to a longer type such as `TEXT`.

Similarly, a very long `VARCHAR` column may need to be changed to one of the longer `TEXT` types if you want to convert it from `utf8` to `utf8mb4`.

- `InnoDB` has a maximum index length of 767 bytes, so for `utf8` or `utf8mb4` columns, you can index a maximum of 255 or 191 characters, respectively. If you currently have `utf8` columns with indexes longer than 191 characters, you will need to index a smaller number of characters. In an `InnoDB` table, these column and index definitions are legal:

```
coll VARCHAR(500) CHARACTER SET utf8, INDEX (coll(255))
```

To use `utf8mb4` instead, the index must be smaller:

```
coll VARCHAR(500) CHARACTER SET utf8mb4, INDEX (coll(191))
```

The preceding types of changes are most likely to be required only if you have very long columns or indexes. Otherwise, you should be able to convert your tables from `utf8` to `utf8mb4` without problems. You can do this by using `ALTER TABLE` as described earlier in this section after upgrading in place to 5.5.

The following items summarize other potential areas of incompatibility:

- Performance of 4-byte UTF-8 (`utf8mb4`) is slower than for 3-byte UTF-8 (`utf8`). If you do not want to incur this penalty, continue to use `utf8`.
- `SET NAMES 'utf8mb4'` causes use of the 4-byte character set for connection character sets. As long as no 4-byte characters are sent from the server, there should be no problems. Otherwise, applications that expect to receive a maximum of three bytes per character may have problems. Conversely, applications that expect to send 4-byte characters must ensure that the server understands them.
- Applications cannot send `utf16` or `utf32` character data to an older server that does not understand them.
- For replication, if the character sets that support supplementary characters are going to be used on the master, all slaves must understand them as well. If you attempt to replicate from a MySQL 5.5 master to an older slave, `utf8` data will be seen as `utf8` by the slave and should replicate correctly. But you cannot send `utf8mb4`, `utf16`, or `utf32` data.

Also, keep in mind the general principle that if a table has different definitions on the master and slave, this can lead to unexpected results. For example, the differences in limitations on index key length makes it risky to use `utf8` on the master and `utf8mb4` on the slave.

If you have upgraded to MySQL 5.5, and then decide to downgrade back to an older release, these considerations apply:

- `ucs2` and `utf8` data should present no problems.
- Any definitions that refer to the `utf8mb4`, `utf16`, or `utf32` character sets will not be recognized by the older server.
- For object definitions that refer to the `utf8mb4` character set, you can dump them with `mysqldump` in MySQL 5.5, edit the dump file to change instances of `utf8mb4` to `utf8`, and reload the file in the older server, as long as there are no 4-byte characters in the data. The older server will see `utf8` in the dump file object definitions and create new objects that use the (3-byte) `utf8` character set.

10.1.12. UTF-8 for Metadata

Metadata is “the data about the data.” Anything that *describes* the database—as opposed to being the *contents* of the database—is metadata. Thus column names, database names, user names, version names, and most of the string results from `SHOW` are metadata. This is also true of the contents of tables in `INFORMATION_SCHEMA` because those tables by definition contain information about database objects.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, neither the `SHOW` statements nor `SELECT` statements for tables in `INFORMATION_SCHEMA` would work properly because different rows in the same column of the results of these operations would be in different character sets.

- Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the `USER()` [1197], `CURRENT_USER()` [1192], `SESSION_USER()` [1197], `SYSTEM_USER()` [1197], `DATABASE()` [1193], and `VERSION()` [1198] functions have the UTF-8 character set by default.

The server sets the `character_set_system` [488] system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_system | utf8 |
+-----+-----+
```

Storage of metadata using Unicode does *not* mean that the server returns headers of columns and the results of `DESCRIBE` functions in the `character_set_system` [488] character set by default. When you use `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set determined by the value of the `character_set_results` [488] system variable, which has a default value of `latin1`. If you want the server to pass metadata results back in a different character set, use the `SET NAMES` statement to force the server to perform character set conversion. `SET NAMES` sets the `character_set_results` [488] and other related system variables. (See [Section 10.1.4, “Connection Character Sets and Collations”](#).) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client to perform the conversion, but this option is not always available for all clients.

If `character_set_results` [488] is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system` [488]).

Error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the `USER()` [1197] function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM t1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF-8 before the comparison.

```
INSERT INTO t1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` [1197] are automatically converted to `latin1` before the assignment.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. For more information about coercion of strings, see [Section 10.1.7.5, “Collation of Expressions”](#).

10.1.13. Column Character Set Conversion

To convert a binary or nonbinary string column to use a particular character set, use `ALTER TABLE`. For successful conversion to occur, one of the following conditions must apply:

- If the column has a binary data type (`BINARY`, `VARBINARY`, `BLOB`), all the values that it contains must be encoded using a single character set (the character set you're converting the column to). If you use a binary column to store information in multiple character sets, MySQL has no way to know which values use which character set and cannot convert the data properly.
- If the column has a nonbinary data type (`CHAR`, `VARCHAR`, `TEXT`), its contents should be encoded in the column character set, not some other character set. If the contents are encoded in a different character set, you can convert the column to use a binary data type first, and then to a nonbinary column with the desired character set.

Suppose that a table `t` has a binary column named `coll` defined as `VARBINARY(50)`. Assuming that the information in the column is encoded using a single character set, you can convert it to a nonbinary column that has that character set. For example, if `coll` contains binary data representing characters in the `greek` character set, you can convert it as follows:

```
ALTER TABLE t MODIFY coll VARCHAR(50) CHARACTER SET greek;
```

If your original column has a type of `BINARY(50)`, you could convert it to `CHAR(50)`, but the resulting values will be padded with `0x00` bytes at the end, which may be undesirable. To remove these bytes, use the `TRIM()` [1109] function:

```
UPDATE t SET coll = TRIM(TRAILING 0x00 FROM coll);
```

Suppose that table `t` has a nonbinary column named `coll` defined as `CHAR(50) CHARACTER SET latin1` but you want to convert it to use `utf8` so that you can store values from many languages. The following statement accomplishes this:

```
ALTER TABLE t MODIFY coll CHAR(50) CHARACTER SET utf8;
```

Conversion may be lossy if the column contains characters that are not in both character sets.

A special case occurs if you have old tables from before MySQL 4.1 where a nonbinary column contains values that actually are encoded in a character set different from the server's default character set. For example, an application might have stored `sjis` values in a column, even though MySQL's default character set was `latin1`. It is possible to convert the column to use the proper character set but an additional step is required. Suppose that the server's default character set was `latin1` and `coll` is defined as `CHAR(50)` but its contents are `sjis` values. The first step is to convert the column to a binary data type, which removes the existing character set information without performing any character conversion:

```
ALTER TABLE t MODIFY coll BLOB;
```

The next step is to convert the column to a nonbinary data type with the proper character set:

```
ALTER TABLE t MODIFY coll CHAR(50) CHARACTER SET sjis;
```

This procedure requires that the table not have been modified already with statements such as `INSERT` or `UPDATE` after an upgrade to MySQL 4.1 or later. In that case, MySQL would store new values in the column using `latin1`, and the column will contain a mix of `sjis` and `latin1` values and cannot be converted properly.

If you specified attributes when creating a column initially, you should also specify them when altering the table with `ALTER TABLE`. For example, if you specified `NOT NULL` and an explicit `DEFAULT` value, you

should also provide them in the `ALTER TABLE` statement. Otherwise, the resulting column definition will not include those attributes.

10.1.14. Character Sets and Collations That MySQL Supports

MySQL supports 70+ collations for 30+ character sets. This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the permissible collations are listed.

You can always list the available character sets and their default collations with the `SHOW CHARACTER SET` statement:

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation
big5	Big5 Traditional Chinese	big5_chinese_ci
dec8	DEC West European	dec8_swedish_ci
cp850	DOS West European	cp850_general_ci
hp8	HP West European	hp8_english_ci
koi8r	KOI8-R Relcom Russian	koi8r_general_ci
latin1	cp1252 West European	latin1_swedish_ci
latin2	ISO 8859-2 Central European	latin2_general_ci
swe7	7bit Swedish	swe7_swedish_ci
ascii	US ASCII	ascii_general_ci
ujis	EUC-JP Japanese	ujis_japanese_ci
sjis	Shift-JIS Japanese	sjis_japanese_ci
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci
tis620	TIS620 Thai	tis620_thai_ci
euckr	EUC-KR Korean	euckr_korean_ci
koi8u	KOI8-U Ukrainian	koi8u_general_ci
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci
greek	ISO 8859-7 Greek	greek_general_ci
cp1250	Windows Central European	cp1250_general_ci
gbk	GBK Simplified Chinese	gbk_chinese_ci
latin5	ISO 8859-9 Turkish	latin5_turkish_ci
armscii8	ARMSCII-8 Armenian	armscii8_general_ci
utf8	UTF-8 Unicode	utf8_general_ci
ucs2	UCS-2 Unicode	ucs2_general_ci
cp866	DOS Russian	cp866_general_ci
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci
macce	Mac Central European	macce_general_ci
macroman	Mac West European	macroman_general_ci
cp852	DOS Central European	cp852_general_ci
latin7	ISO 8859-13 Baltic	latin7_general_ci
utf8mb4	UTF-8 Unicode	utf8mb4_general_ci
cp1251	Windows Cyrillic	cp1251_general_ci
utf16	UTF-16 Unicode	utf16_general_ci
cp1256	Windows Arabic	cp1256_general_ci
cp1257	Windows Baltic	cp1257_general_ci
utf32	UTF-32 Unicode	utf32_general_ci
binary	Binary pseudo charset	binary
geostd8	GEOSTD8 Georgian	geostd8_general_ci
cp932	SJIS for Windows Japanese	cp932_japanese_ci
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci

The `utf8mb4`, `utf16`, and `utf32` character sets were added in MySQL 5.5.3.

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

Collation-Charts.Org is a useful site for information that shows how one collation compares to another.

10.1.14.1. Unicode Character Sets

MySQL 5.5 supports these Unicode character sets:

- `ucs2`, the UCS-2 encoding of the Unicode character set using 16 bits per character.
- `utf16`, the UTF-16 encoding for the Unicode character set; like `ucs2` but with an extension for supplementary characters.
- `utf32`, the UTF-32 encoding for the Unicode character set using 32 bits per character.
- `utf8`, a UTF-8 encoding of the Unicode character set using one to three bytes per character.
- `utf8mb4`, a UTF-8 encoding of the Unicode character set using one to four bytes per character.

`ucs2` and `utf8` support Basic Multilingual Plane (BMP) characters. `utf8mb4`, `utf16`, and `utf32` support BMP and supplementary characters. The `utf8mb4`, `utf16`, and `utf32` character sets were added in MySQL 5.5.3.

You can store text in about 650 languages using these character sets. This section lists the collations available for each Unicode character set and describes their differentiating properties. For general information about the character sets, see [Section 10.1.10, “Unicode Support”](#).

A similar set of collations is available for each Unicode character set. These are shown in the following list, where `xxx` represents the character set name. For example, `xxx_danish_ci` represents the Danish collations, the specific names of which are `ucs2_danish_ci`, `utf16_danish_ci`, `utf32_danish_ci`, `utf8_danish_ci`, and `utf8mb4_danish_ci`.

- `xxx_bin`
- `xxx_czech_ci`
- `xxx_danish_ci`
- `xxx_esperanto_ci`
- `xxx_estonian_ci`
- `xxx_general_ci` (default)
- `xxx_general_mysql500_ci`
- `xxx_hungarian_ci`
- `xxx_icelandic_ci`
- `xxx_latvian_ci`
- `xxx_lithuanian_ci`
- `xxx_persian_ci`
- `xxx_polish_ci`
- `xxx_roman_ci`
- `xxx_romanian_ci`
- `xxx_sinhala_ci`

- `xxx_slovak_ci`
- `xxx_slovenian_ci`
- `xxx_spanish_ci`
- `xxx_spanish2_ci`
- `xxx_swedish_ci`
- `xxx_turkish_ci`
- `xxx_unicode_ci`

MySQL implements the `xxx_unicode_ci` collations according to the Unicode Collation Algorithm (UCA) described at <http://www.unicode.org/reports/tr10/>. The collation uses the version-4.0.0 UCA weight keys: <http://www.unicode.org/Public/UC4.0.0/allkeys-4.0.0.txt>. Currently, the `xxx_unicode_ci` collations have only partial support for the Unicode Collation Algorithm. Some characters are not supported yet. Also, combining marks are not fully supported. This affects primarily Vietnamese, Yoruba, and some smaller languages such as Navajo.

MySQL implements language-specific Unicode collations only if the ordering with `xxx_unicode_ci` does not work well for a language. Language-specific collations are UCA-based. They are derived from `xxx_unicode_ci` with additional language tailoring rules.

For any Unicode character set, operations performed using the `xxx_general_ci` collation are faster than those for the `xxx_unicode_ci` collation. For example, comparisons for the `utf8_general_ci` collation are faster, but slightly less correct, than comparisons for `utf8_unicode_ci`. The reason for this is that `utf8_unicode_ci` supports mappings such as expansions; that is, when one character compares as equal to combinations of other characters. For example, in German and some other languages “ß” is equal to “ss”. `utf8_unicode_ci` also supports contractions and ignorable characters. `utf8_general_ci` is a legacy collation that does not support expansions, contractions, or ignorable characters. It can make only one-to-one comparisons between characters.

To further illustrate, the following equalities hold in both `utf8_general_ci` and `utf8_unicode_ci` (for the effect this has in comparisons or when doing searches, see [Section 10.1.7.8, “Examples of the Effect of Collation”](#)):

```
Ä = A
Ö = O
Û = U
```

A difference between the collations is that this is true for `utf8_general_ci`:

```
ß = s
```

Whereas this is true for `utf8_unicode_ci`, which supports the German DIN-1 ordering (also known as dictionary order):

```
ß = ss
```

MySQL implements language-specific collations for the `utf8` character set only if the ordering with `utf8_unicode_ci` does not work well for a language. For example, `utf8_unicode_ci` works fine for German dictionary order and French, so there is no need to create special `utf8` collations.

`utf8_general_ci` also is satisfactory for both German and French, except that “ß” is equal to “s”, and not to “ss”. If this is acceptable for your application, you should use `utf8_general_ci` because it is faster. Otherwise, use `utf8_unicode_ci` because it is more accurate.

`xxx_swedish_ci` includes Swedish rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

```
Ü = Y < Ö
```

The `xxx_spanish_ci` and `xxx_spanish2_ci` collations correspond to modern Spanish and traditional Spanish, respectively. In both collations, “ñ” (n-tilde) is a separate letter between “n” and “o”. In addition, for traditional Spanish, “ch” is a separate letter between “c” and “d”, and “ll” is a separate letter between “l” and “m”

In the `xxx_roman_ci` collations, I and J compare as equal, and U and V compare as equal.

For all Unicode collations except the “binary” (`xxx_bin`) collations, MySQL performs a table lookup to find a character's collating weight. If a character is not in the table (for example, because it is a “new” character), collating weight determination becomes more complex:

- For BMP characters in general collations (`xxx_general_ci`), weight = code point.
- For BMP characters in UCA collations (for example, `xxx_unicode_ci` and language-specific collations), the following algorithm applies:

```
if (code >= 0x3400 && code <= 0x4DB5)
    base= 0xFB80; /* CJK Ideograph Extension */
else if (code >= 0x4E00 && code <= 0x9FA5)
    base= 0xFB40; /* CJK Ideograph */
else
    base= 0xFBC0; /* All other characters */
aaaa= base + (code >> 15);
bbbb= (code & 0x7FFF) | 0x8000;
```

The result is a sequence of two collating elements, `aaaa` followed by `bbbb`.

Thus, `U+04cf CYRILLIC SMALL LETTER PALOCHKA` currently is, with all UCA collations, greater than `U+04c0 CYRILLIC LETTER PALOCHKA`. Eventually, after further collation tuning, all palochkas will sort together.

- For supplementary characters in general collations, the weight is the weight for `0xffff REPLACEMENT CHARACTER`. For supplementary characters in UCA collations, their collating weight is `0xffff`. That is, to MySQL, all supplementary characters are equal to each other, and greater than almost all BMP characters.

An example with Deseret characters and `COUNT(DISTINCT)`:

```
CREATE TABLE t (s1 VARCHAR(5) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
INSERT INTO t VALUES (0xffff); /* REPLACEMENT CHARACTER */
INSERT INTO t VALUES (0x010412); /* DESERET CAPITAL LETTER BEE */
INSERT INTO t VALUES (0x010413); /* DESERET CAPITAL LETTER TEE */
SELECT COUNT(DISTINCT s1) FROM t;
```

The result is 2 because in the MySQL `xxx_unicode_ci` collations, the replacement character has a weight of `0x0dc6`, whereas Deseret Bee and Deseret Tee both have a weight of `0xffff`. (Were the `utf32_general_ci` collation used instead, the result would be 1 because all three characters have a weight of `0xffff` in that collation.)

The current rule that all supplementary characters are equal to each other is nonoptimal but is not expected to cause trouble. These characters are very rare, so it will be very rare that a multi-character string consists entirely of supplementary characters. In Japan, since the supplementary characters are obscure Kanji ideographs, the typical user does not care what order they are in, anyway. If you really want rows sorted by MySQL's rule and secondarily by code point value, it is easy:

```
ORDER BY s1 COLLATE utf32_unicode_ci, s1 COLLATE utf32_bin
```

The `utf16_bin` Collation

There is a difference between “ordering by the character's code value” and “ordering by the character's binary representation,” a difference that appears only with `utf16_bin`, because of surrogates.

Suppose that `utf16_bin` (the binary collation for `utf16`) was a binary comparison “byte by byte” rather than “character by character.” If that were so, the order of characters in `utf16_bin` would differ from the order in `utf8_bin`. For example, the following chart shows two rare characters. The first character is in the range `E000-FFFF`, so it is greater than a surrogate but less than a supplementary. The second character is a supplementary.

Code point	Character	utf8	utf16
0FF9D	HALFWIDTH KATAKANA LETTER N	EF BE 9D	FF 9D
10384	UGARITIC LETTER DELTA	F0 90 8E 84	D8 00 DF 84

The two characters in the chart are in order by code point value because `0xff9d < 0x10384`. And they are in order by `utf8` value because `0xef < 0xf0`. But they are not in order by `utf16` value, if we use byte-by-byte comparison, because `0xff > 0xd8`.

So MySQL's `utf16_bin` collation is not “byte by byte.” It is “by code point.” When MySQL sees a supplementary-character encoding in `utf16`, it converts to the character's code-point value, and then compares. Therefore, `utf8_bin` and `utf16_bin` are the same ordering. This is consistent with the SQL:2008 standard requirement for a UCS_BASIC collation: “UCS_BASIC is a collation in which the ordering is determined entirely by the Unicode scalar values of the characters in the strings being sorted. It is applicable to the UCS character repertoire. Since every character repertoire is a subset of the UCS repertoire, the UCS_BASIC collation is potentially applicable to every character set. NOTE 11: The Unicode scalar value of a character is its code point treated as an unsigned integer.”

If the character set is `ucs2`, comparison is byte-by-byte, but `ucs2` strings should not contain surrogates, anyway.

The `xxx_general_mysql500_ci` collations were added in MySQL 5.5.21. They preserve the pre-5.1.24 ordering of the original `xxx_general_ci` collations and permit upgrades for tables created before MySQL 5.1.24. For more information, see [Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”](#), and [Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”](#).

For additional information about Unicode collations in MySQL, see Collation-Charts.Org (`utf8`).

10.1.14.2. West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- `ascii` (US ASCII) collations:
 - `ascii_bin`
 - `ascii_general_ci` (default)
- `cp850` (DOS West European) collations:
 - `cp850_bin`
 - `cp850_general_ci` (default)

- `dec8` (DEC Western European) collations:
 - `dec8_bin`
 - `dec8_swedish_ci` (default)
- `hp8` (HP Western European) collations:
 - `hp8_bin`
 - `hp8_english_ci` (default)
- `latin1` (cp1252 West European) collations:
 - `latin1_bin`
 - `latin1_danish_ci`
 - `latin1_general_ci`
 - `latin1_general_cs`
 - `latin1_german1_ci`
 - `latin1_german2_ci`
 - `latin1_spanish_ci`
 - `latin1_swedish_ci` (default)

`latin1` is the default character set. MySQL's `latin1` is the same as the Windows `cp1252` character set. This means it is the same as the official [ISO 8859-1](#) or IANA (Internet Assigned Numbers Authority) `latin1`, except that IANA `latin1` treats the code points between `0x80` and `0x9f` as “undefined,” whereas `cp1252`, and therefore MySQL's `latin1`, assign characters for those positions. For example, `0x80` is the Euro sign. For the “undefined” entries in `cp1252`, MySQL translates `0x81` to Unicode `0x0081`, `0x8d` to `0x008d`, `0x8f` to `0x008f`, `0x90` to `0x0090`, and `0x9d` to `0x009d`.

The `latin1_swedish_ci` collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.

The `latin1_german1_ci` and `latin1_german2_ci` collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the “dictionary collation” and DIN-2 is called the “phone book collation.” For an example of the effect this has in comparisons or when doing searches, see [Section 10.1.7.8, “Examples of the Effect of Collation”](#).

- `latin1_german1_ci` (dictionary) rules:

```

Ä = A
Ö = O
Ü = U
ß = s
    
```

- `latin1_german2_ci` (phone-book) rules:

```

Ä = AE
Ö = OE
Ü = UE
    
```

ß = ss

In the `latin1_spanish_ci` collation, “ñ” (n-tilde) is a separate letter between “n” and “o”.

- `macroman` (Mac West European) collations:
 - `macroman_bin`
 - `macroman_general_ci` (default)
- `swe7` (7bit Swedish) collations:
 - `swe7_bin`
 - `swe7_swedish_ci` (default)

For additional information about Western European collations in MySQL, see Collation-Charts.Org ([ascii](#), [cp850](#), [dec8](#), [hp8](#), [latin1](#), [macroman](#), [swe7](#)).

10.1.14.3. Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, Poland, and Serbia (Latin).

- `cp1250` (Windows Central European) collations:
 - `cp1250_bin`
 - `cp1250_croatian_ci`
 - `cp1250_czech_cs`
 - `cp1250_general_ci` (default)
 - `cp1250_polish_ci`
- `cp852` (DOS Central European) collations:
 - `cp852_bin`
 - `cp852_general_ci` (default)
- `keybcs2` (DOS Kamenicky Czech-Slovak) collations:
 - `keybcs2_bin`
 - `keybcs2_general_ci` (default)
- `latin2` (ISO 8859-2 Central European) collations:
 - `latin2_bin`
 - `latin2_croatian_ci`
 - `latin2_czech_cs`
 - `latin2_general_ci` (default)
 - `latin2_hungarian_ci`

- `macce` (Mac Central European) collations:
 - `macce_bin`
 - `macce_general_ci` (default)

For additional information about Central European collations in MySQL, see Collation-Charts.Org ([cp1250](#), [cp852](#), [keybcs2](#), [latin2](#), [macce](#)).

10.1.14.4. South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

- `armscii8` (ARMScii8 Armenian) collations:
 - `armscii8_bin`
 - `armscii8_general_ci` (default)
- `cp1256` (Windows Arabic) collations:
 - `cp1256_bin`
 - `cp1256_general_ci` (default)
- `geostd8` (GEOSTD8 Georgian) collations:
 - `geostd8_bin`
 - `geostd8_general_ci` (default)
- `greek` (ISO 8859-7 Greek) collations:
 - `greek_bin`
 - `greek_general_ci` (default)
- `hebrew` (ISO 8859-8 Hebrew) collations:
 - `hebrew_bin`
 - `hebrew_general_ci` (default)
- `latin5` (ISO 8859-9 Turkish) collations:
 - `latin5_bin`
 - `latin5_turkish_ci` (default)

For additional information about South European and Middle Eastern collations in MySQL, see Collation-Charts.Org ([armscii8](#), [cp1256](#), [geostd8](#), [greek](#), [hebrew](#), [latin5](#)).

10.1.14.5. Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- `cp1257` (Windows Baltic) collations:
 - `cp1257_bin`

- `cp1257_general_ci` (default)
- `cp1257_lithuanian_ci`
- `latin7` (ISO 8859-13 Baltic) collations:
 - `latin7_bin`
 - `latin7_estonian_cs`
 - `latin7_general_ci` (default)
 - `latin7_general_cs`

For additional information about Baltic collations in MySQL, see Collation-Charts.Org ([cp1257](#), [latin7](#)).

10.1.14.6. Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, Ukrainian, and Serbian (Cyrillic) languages.

- `cp1251` (Windows Cyrillic) collations:
 - `cp1251_bin`
 - `cp1251_bulgarian_ci`
 - `cp1251_general_ci` (default)
 - `cp1251_general_cs`
 - `cp1251_ukrainian_ci`
- `cp866` (DOS Russian) collations:
 - `cp866_bin`
 - `cp866_general_ci` (default)
- `koi8r` (KOI8-R Relcom Russian) collations:
 - `koi8r_bin`
 - `koi8r_general_ci` (default)
- `koi8u` (KOI8-U Ukrainian) collations:
 - `koi8u_bin`
 - `koi8u_general_ci` (default)

For additional information about Cyrillic collations in MySQL, see Collation-Charts.Org ([cp1251](#), [cp866](#), [koi8r](#), [koi8u](#)).).

10.1.14.7. Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See

[Section 10.1.14.7.1, “The `cp932` Character Set”](#), for additional information about the `cp932` and `sjis` character sets.

For answers to some common questions and problems relating support for Asian character sets in MySQL, see [Section B.11, “MySQL 5.5 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”](#).

- `big5` (Big5 Traditional Chinese) collations:
 - `big5_bin`
 - `big5_chinese_ci` (default)
- `cp932` (SJIS for Windows Japanese) collations:
 - `cp932_bin`
 - `cp932_japanese_ci` (default)
- `ujis` (EUC-JP Japanese) collations:
 - `ujis_bin`
 - `ujis_japanese_ci` (default)
- `eucjpms` (UJIS for Windows Japanese) collations:
 - `eucjpms_bin`
 - `eucjpms_japanese_ci` (default)
- `euckr` (EUC-KR Korean) collations:
 - `euckr_bin`
 - `euckr_korean_ci` (default)
- `gb2312` (GB2312 Simplified Chinese) collations:
 - `gb2312_bin`
 - `gb2312_chinese_ci` (default)
- `gbk` (GBK Simplified Chinese) collations:
 - `gbk_bin`
 - `gbk_chinese_ci` (default)
- `sjis` (Shift-JIS Japanese) collations:
 - `sjis_bin`
 - `sjis_japanese_ci` (default)
- `tis620` (TIS620 Thai) collations:
 - `tis620_bin`
 - `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:
 - `ujis_bin`
 - `ujis_japanese_ci` (default)

The `big5_chinese_ci` collation sorts on number of strokes.

For additional information about Asian collations in MySQL, see Collation-Charts.Org ([big5](#), [cp932](#), [eucjms](#), [euckr](#), [gb2312](#), [gbk](#), [sjis](#), [tis620](#), [ujis](#)).

10.1.14.7.1. The `cp932` Character Set

Why is `cp932` needed?

In MySQL, the `sjis` character set corresponds to the `Shift_JIS` character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See <http://www.iana.org/assignments/character-sets>.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to `Shift_JIS` that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of `Shift_JIS` and its exact name is `Microsoft Windows Codepage : 932` or `cp932`. In addition to the characters supported by `Shift_JIS`, `cp932` supports extension characters such as NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.

Many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted using Unicode (`ucs2`).
- The `sjis` character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the conversion rule. MySQL supports only one of these rules (described later).

The MySQL `cp932` character set is designed to solve these problems.

Because MySQL supports character set conversion, it is important to separate IANA `Shift_JIS` and `cp932` into two different character sets because they provide different conversion rules.

How does `cp932` differ from `sjis`?

The `cp932` character set differs from `sjis` in the following ways:

- `cp932` supports NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.
- Some `cp932` characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to `cp932`, one of the code points must be selected. For this “round trip conversion,” the rule recommended by Microsoft is used. (See <http://support.microsoft.com/kb/170559/EN-US/>.)

The conversion rule works like this:

- If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
- If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.
- If the character is in both IBM selected characters and NEC selected—IBM extended characters, use the code point of IBM extended characters.

The table shown at <http://www.microsoft.com/globaldev/reference/dbcs/932.htm> provides information about the Unicode values of `cp932` characters. For `cp932` table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (`ucs2`) encoding. For table entries with an underlined two-digit value appears, there is a range of `cp932` character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the `cp932` characters that begin with those digits.

The following links are of special interest. They correspond to the encodings for the following sets of characters:

- NEC special characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_87.htm

- NEC selected—IBM extended characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_ED.htm

http://www.microsoft.com/globaldev/reference/dbcs/932/932_EE.htm

- IBM selected characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_FA.htm

http://www.microsoft.com/globaldev/reference/dbcs/932/932_FB.htm

http://www.microsoft.com/globaldev/reference/dbcs/932/932_FC.htm

- `cp932` supports conversion of user-defined characters in combination with `ucjpm`, and solves the problems with `sjis/ujis` conversion. For details, please refer to <http://www.opengroup.or.jp/jvc/cde/sjis-euc-e.html>.

For some characters, conversion to and from `ucs2` is different for `sjis` and `cp932`. The following tables illustrate these differences.

Conversion to `ucs2`:

<code>sjis/cp932</code> Value	<code>sjis</code> -> <code>ucs2</code> Conversion	<code>cp932</code> -> <code>ucs2</code> Conversion
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

Conversion from `ucs2`:

<code>ucs2</code> value	<code>ucs2</code> -> <code>sjis</code> Conversion	<code>ucs2</code> -> <code>cp932</code> Conversion
005C	815F	5C
007E	7E	7E

ucs2 value	ucs2 -> sjis Conversion	ucs2 -> cp932 Conversion
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

Users of any Japanese character sets should be aware that using `--character-set-client-handshake` [437] (or `--skip-character-set-client-handshake` [437]) has an important effect. See Section 5.1.3, “Server Command Options”.

10.2. Setting the Error Message Language

By default, `mysqld` produces error messages in English, but they can also be displayed in any of several other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish.

You can select which language the server uses for error messages using the instructions in this section.

As of MySQL 5.5, the server searches for the error message file in two locations:

- It tries to find the file in a directory constructed from two system variable values, `lc_messages_dir` [512] and `lc_messages` [512], with the latter converted to a language name. Suppose that you start the server using this command:

```
shell> mysqld --lc_messages_dir=/usr/share/mysql --lc_messages=fr_FR
```

In this case, `mysqld` maps the locale `fr_FR` to the language `french` and looks for the error file in the `/usr/share/mysql/french` directory.

- If the message file cannot be found in the directory constructed as just described, the server ignores the `lc_messages` [512] value and uses only the `lc_messages_dir` [512] value as the location in which to look.

The `lc_messages_dir` [512] system variable has only a global value and is read only. `lc_messages` [512] has global and session values and can be modified at runtime, so the error message language can be changed while the server is running, and individual clients each can have a different error message language by changing their session `lc_messages` [512] value to a different locale name. For example, if the server is using the `fr_FR` locale for error messages, a client can execute this statement to receive error messages in English:

```
mysql> SET lc_messages = 'en_US';
```

Before MySQL 5.5, the `lc_messages_dir` [512] and `lc_messages` [512] system variables were unavailable. To start `mysqld` with a particular language for error messages, the `--language` [446] or `-L` option were used. The option value can be a language name or the full path to the error message file. For example:

```
shell> mysqld --language=french
```

Or:

```
shell> mysqld --language=/usr/local/share/french
```

Specify the language name in lowercase.

MySQL 5.5 treats `--language` [446] as an alias for `lc_messages_dir` [512].

By default, the language files are located in the `share/mysql/LANGUAGE` directory under the MySQL base directory.

For information about changing the character set for error messages (rather than the language), see [Section 10.1.6, “Character Set for Error Messages”](#).

You can change the content of the error messages produced by the server using the instructions in the MySQL Internals manual, available at [MySQL Internals: Error Messages](#). If you do change the content of error messages, remember to repeat your changes after each upgrade to a newer version of MySQL.

10.3. Adding a Character Set

This section discusses the procedure for adding a character set to MySQL. The proper procedure depends on whether the character set is simple or complex:

- If the character set does not need special string collating routines for sorting and does not need multi-byte character support, it is simple.
- If the character set needs either of those features, it is complex.

For example, `greek` and `swe7` are simple character sets, whereas `big5` and `czech` are complex character sets.

To use the following instructions, you must have a MySQL source distribution. In the instructions, `MYSET` represents the name of the character set that you want to add.

1. Add a `<charset>` element for `MYSET` to the `sql/share/charsets/Index.xml` file. Use the existing contents in the file as a guide to adding new contents. A partial listing for the `latin1` `<charset>` element follows:

```
<charset name="latin1">
  <family>Western</family>
  <description>cp1252 West European</description>
  ...
  <collation name="latin1_swedish_ci" id="8" order="Finnish, Swedish">
    <flag>primary</flag>
    <flag>compiled</flag>
  </collation>
  <collation name="latin1_danish_ci" id="15" order="Danish"/>
  ...
  <collation name="latin1_bin" id="47" order="Binary">
```

```

    <flag>binary</flag>
    <flag>compiled</flag>
  </collation>
  ...
</charset>

```

The `<charset>` element must list all the collations for the character set. These must include at least a binary collation and a default (primary) collation. The default collation is often named using a suffix of `general_ci` (general, case insensitive). It is possible for the binary collation to be the default collation, but usually they are different. The default collation should have a `primary` flag. The binary collation should have a `binary` flag.

You must assign a unique ID number to each collation. The range of IDs from 1024 to 2047 is reserved for user-defined collations. Before MySQL 5.5, the ID must be chosen from the range 1 to 254. To find the maximum of the currently used collation IDs, use this query:

```
SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
```

2. This step depends on whether you are adding a simple or complex character set. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines collation functions, multi-byte functions, or both.

For a simple character set, create a configuration file, `MYSET.xml`, that describes the character set properties. Create this file in the `sql/share/charsets` directory. You can use a copy of `latin1.xml` as the basis for this file. The syntax for the file is very simple:

- Comments are written as ordinary XML comments (`<!-- text -->`).
- Words within `<map>` array elements are separated by arbitrary amounts of whitespace.
- Each word within `<map>` array elements must be a number in hexadecimal format.
- The `<map>` array element for the `<ctype>` element has 257 words. The other `<map>` array elements after that have 256 words. See [Section 10.3.1, “Character Definition Arrays”](#).
- For each collation listed in the `<charset>` element for the character set in `Index.xml`, `MYSET.xml` must contain a `<collation>` element that defines the character ordering.

For a complex character set, create a C source file that describes the character set properties and defines the support routines necessary to properly perform operations on the character set:

- Create the file `ctype-MYSET.c` in the `strings` directory. Look at one of the existing `ctype-*.c` files (such as `ctype-big5.c`) to see what needs to be defined. The arrays in your file must have names like `ctype_MYSET`, `to_lower_MYSET`, and so on. These correspond to the arrays for a simple character set. See [Section 10.3.1, “Character Definition Arrays”](#).
 - For each `<collation>` element listed in the `<charset>` element for the character set in `Index.xml`, the `ctype-MYSET.c` file must provide an implementation of the collation.
 - If the character set requires string collating functions, see [Section 10.3.2, “String Collating Support for Complex Character Sets”](#).
 - If the character set requires multi-byte character support, see [Section 10.3.3, “Multi-Byte Character Support for Complex Character Sets”](#).
3. Modify the configuration information. Use the existing configuration information as a guide to adding information for `MYSYS`. The example here assumes that the character set has default and binary collations, but more lines are needed if `MYSET` has additional collations.

- a. Edit `mysys/charset-def.c`, and “register” the collations for the new character set.

Add these lines to the “declaration” section:

```
#ifdef HAVE_CHARSET_MYSET
extern CHARSET_INFO my_charset_MYSET_general_ci;
extern CHARSET_INFO my_charset_MYSET_bin;
#endif
```

Add these lines to the “registration” section:

```
#ifdef HAVE_CHARSET_MYSET
    add_compiled_collation(&my_charset_MYSET_general_ci);
    add_compiled_collation(&my_charset_MYSET_bin);
#endif
```

- b. If the character set uses `ctype-MYSET.c`, edit `strings/CMakeLists.txt` and add `ctype-MYSET.c` to the definition of the `STRINGS_SOURCES` variable.
- c. Edit `cmake/character_sets.cmake`:
 - i. Add `MYSET` to the value of with `CHARSETS_AVAILABLE` in alphabetic order.
 - ii. Add `MYSET` to the value of `CHARSETS_COMPLEX` in alphabetic order. This is needed even for simple character sets, or CMake will not recognize `-DDEFAULT_CHARSET=MYSET` [154].

4. Reconfigure, recompile, and test.

10.3.1. Character Definition Arrays

Each simple character set has a configuration file located in the `sql/share/charsets` directory. For a character set named `MYSYS`, the file is named `MYSET.xml`. It uses `<map>` array elements to list character set properties. `<map>` elements appear within these elements:

- `<ctype>` defines attributes for each character.
- `<lower>` and `<upper>` list the lowercase and uppercase characters.
- `<unicode>` maps 8-bit character values to Unicode values.
- `<collation>` elements indicate character ordering for comparisons and sorts, one element per collation. Binary collations need no `<map>` element because the character codes themselves provide the ordering.

For a complex character set as implemented in a `ctype-MYSET.c` file in the `strings` directory, there are corresponding arrays: `ctype_MYSET[]`, `to_lower_MYSET[]`, and so forth. Not every complex character set has all of the arrays. See also the existing `ctype-*.c` files for examples. See the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

Most of the arrays are indexed by character value and have 256 elements. The `<ctype>` array is indexed by character value + 1 and has 257 elements. This is a legacy convention for handling EOF.

`<ctype>` array elements are bit values. Each element describes the attributes of a single character in the character set. Each attribute is associated with a bitmask, as defined in `include/m_ctype.h`:

```
#define _MY_U 01 /* Upper case */
#define _MY_L 02 /* Lower case */
#define _MY_NMR 04 /* Numeral (digit) */
```

```
#define _MY_SPC 010    /* Spacing character */
#define _MY_PNT 020    /* Punctuation */
#define _MY_CTR 040    /* Control character */
#define _MY_B 0100    /* Blank */
#define _MY_X 0200    /* hexadecimal digit */
```

The `<ctype>` value for a given character should be the union of the applicable bitmask values that describe the character. For example, 'A' is an uppercase character (`_MY_U`) as well as a hexadecimal digit (`_MY_X`), so its `ctype` value should be defined like this:

```
ctype['A'+1] = _MY_U | _MY_X = 01 | 0200 = 0201
```

The bitmask values in `m_ctype.h` are octal values, but the elements of the `<ctype>` array in `MYSET.xml` should be written as hexadecimal values.

The `<lower>` and `<upper>` arrays hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
lower['A'] should contain 'a'
upper['a'] should contain 'A'
```

Each `<collation>` array indicates how characters should be ordered for comparison and sorting purposes. MySQL sorts characters based on the values of this information. In some cases, this is the same as the `<upper>` array, which means that sorting is case-insensitive. For more complicated sorting rules (for complex character sets), see the discussion of string collating in [Section 10.3.2, “String Collating Support for Complex Character Sets”](#).

10.3.2. String Collating Support for Complex Character Sets

For a simple character set named `MYSET`, sorting rules are specified in the `MYSET.xml` configuration file using `<map>` array elements within `<collation>` elements. If the sorting rules for your language are too complex to be handled with simple arrays, you must define string collating functions in the `ctype-MYSET.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `big5`, `czech`, `gbk`, `sjis`, and `tis160` character sets. Take a look at the `MY_COLLATION_HANDLER` structures to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

10.3.3. Multi-Byte Character Support for Complex Character Sets

If you want to add support for a new character set named `MYSET` that includes multi-byte characters, you must use multi-byte character functions in the `ctype-MYSET.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `euc_kr`, `gb2312`, `gbk`, `sjis`, and `ujis` character sets. Take a look at the `MY_CHARSET_HANDLER` structures to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

10.4. Adding a Collation to a Character Set

A collation is a set of rules that defines how to compare and sort character strings. Each collation in MySQL belongs to a single character set. Every character set has at least one collation, and most have two or more collations.

A collation orders characters based on weights. Each character in a character set maps to a weight. Characters with equal weights compare as equal, and characters with unequal weights compare according to the relative magnitude of their weights.

MySQL supports several collation implementations, as discussed in [Section 10.4.1, “Collation Implementation Types”](#). Some of these can be added to MySQL without recompiling:

- Simple collations for 8-bit character sets.
- UCA-based collations for Unicode character sets.
- Binary (`xxx_bin`) collations.

The following sections describe how to add collations of the first two types to existing character sets. All existing character sets already have a binary collation, so there is no need here to describe how to add one.

Summary of the procedure for adding a new collation:

1. Choose a collation ID.
2. Add configuration information that names the collation and describes the character-ordering rules.
3. Restart the server.
4. Verify that the collation is present.

The instructions here cover only collations that can be added without recompiling MySQL. To add a collation that does require recompiling (as implemented by means of functions in a C source file), use the instructions in [Section 10.3, “Adding a Character Set”](#). However, instead of adding all the information required for a complete character set, just modify the appropriate files for an existing character set. That is, based on what is already present for the character set’s current collations, add data structures, functions, and configuration information for the new collation.

**Note**

If you modify an existing collation, that may affect the ordering of rows for indexes on columns that use the collation. In this case, rebuild any such indexes to avoid problems such as incorrect query results. For further information, see [Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”](#).

Additional Resources

- The Unicode Collation Algorithm (UCA) specification: <http://www.unicode.org/reports/tr10/>
- The Locale Data Markup Language (LDML) specification: <http://www.unicode.org/reports/tr35/>
- MySQL Blog article “Instructions for adding a new Unicode collation”: <http://blogs.mysql.com/peterg/2008/05/19/instructions-for-adding-a-new-unicode-collation/>

10.4.1. Collation Implementation Types

MySQL implements several types of collations:

Simple collations for 8-bit character sets

This kind of collation is implemented using an array of 256 weights that defines a one-to-one mapping from character codes to weights. `latin1_swedish_ci` is an example. It is a case-insensitive collation, so the uppercase and lowercase versions of a character have the same weights and they compare as equal.

```
mysql> SET NAMES 'latin1' COLLATE 'latin1_swedish_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a' = 'A';
+-----+
| 'a' = 'A' |
+-----+
|          1 |
+-----+
1 row in set (0.00 sec)
```

For implementation instructions, see [Section 10.4.3, “Adding a Simple Collation to an 8-Bit Character Set”](#).

Complex collations for 8-bit character sets

This kind of collation is implemented using functions in a C source file that define how to order characters, as described in [Section 10.3, “Adding a Character Set”](#).

Collations for non-Unicode multi-byte character sets

For this type of collation, 8-bit (single-byte) and multi-byte characters are handled differently. For 8-bit characters, character codes map to weights in case-insensitive fashion. (For example, the single-byte characters 'a' and 'A' both have a weight of 0x41.) For multi-byte characters, there are two types of relationship between character codes and weights:

- Weights equal character codes. `sjis_japanese_ci` is an example of this kind of collation. The multi-byte character 'ぢ' has a character code of 0x82C0, and the weight is also 0x82C0.
- Character codes map one-to-one to weights, but a code is not necessarily equal to the weight. `gbk_chinese_ci` is an example of this kind of collation. The multi-byte character '臙' has a character code of 0x81B0 but a weight of 0xC286.

For implementation instructions, see [Section 10.3, “Adding a Character Set”](#).

Collations for Unicode multi-byte character sets

Some of these collations are based on the Unicode Collation Algorithm (UCA), others are not.

Non-UCA collations have a one-to-one mapping from character code to weight. In MySQL, such collations are case insensitive and accent insensitive. `utf8_general_ci` is an example: 'a', 'A', 'À', and 'á' each have different character codes but all have a weight of 0x0041 and compare as equal.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_general_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a' = 'A', 'a' = 'À', 'a' = 'á';
+-----+-----+-----+
| 'a' = 'A' | 'a' = 'À' | 'a' = 'á' |
+-----+-----+-----+
|          1 |          1 |          1 |
+-----+-----+-----+
1 row in set (0.06 sec)
```

UCA-based collations in MySQL have these properties:

- If a character has weights, each weight uses 2 bytes (16 bits).
- A character may have zero weights (or an empty weight). In this case, the character is ignorable. Example: "U+0000 NULL" does not have a weight and is ignorable.
- A character may have one weight. Example: 'a' has a weight of 0x0E33.

- A character may have many weights. This is an expansion. Example: The German letter 'ß' (SZ ligature, or SHARP S) has a weight of `0x0FEA0FEA`.
- Many characters may have one weight. This is a contraction. Example: 'ch' is a single letter in Czech and has a weight of `0x0EE2`.

A many-characters-to-many-weights mapping is also possible (this is contraction with expansion), but is not supported by MySQL.

For implementation instructions, for a non-UCA collation, see [Section 10.3, “Adding a Character Set”](#). For a UCA collation, see [Section 10.4.4, “Adding a UCA Collation to a Unicode Character Set”](#).

Miscellaneous collations

There are also a few collations that do not fall into any of the previous categories.

10.4.2. Choosing a Collation ID

Each collation must have a unique ID. To add a collation, you must choose an ID value that is not currently used. The range of IDs from 1024 to 2047 is reserved for user-defined collations. Before MySQL 5.5, the ID must be chosen from the range 1 to 254. The collation ID that you choose will appear in these contexts:

- The `ID` column of the `INFORMATION_SCHEMA.COLLATIONS` table.
- The `Id` column of `SHOW COLLATION` output.
- The `charsetnr` member of the `MYSQL_FIELD` C API data structure.
- The `number` member of the `MY_CHARSET_INFO` data structure returned by the `mysql_get_character_set_info()` C API function.

To determine the largest currently used ID, issue the following statement:

```
mysql> SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
+-----+
| MAX(ID) |
+-----+
|      210 |
+-----+
```

To display a list of all currently used IDs, issue this statement:

```
mysql> SELECT ID FROM INFORMATION_SCHEMA.COLLATIONS ORDER BY ID;
+-----+
| ID |
+-----+
|  1 |
|  2 |
| ... |
| 52 |
| 53 |
| 57 |
| 58 |
| ... |
| 98 |
| 99 |
|128 |
|129 |
| ... |
|210 |
+-----+
```

**Warning**

Before MySQL 5.5, which provides for a range of user-defined collation IDs, you must choose an ID in the range from 1 to 254. In this case, if you upgrade MySQL, you may find that the collation ID you choose has been assigned to a collation included in the new MySQL distribution. In this case, you will need to choose a new value for your own collation.

In addition, before upgrading, you should save the configuration files that you change. If you upgrade in place, the process will replace the your modified files.

10.4.3. Adding a Simple Collation to an 8-Bit Character Set

This section describes how to add a simple collation for an 8-bit character set by writing the `<collation>` elements associated with a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. The example adds a collation named `latin1_test_ci` to the `latin1` character set.

1. Choose a collation ID, as shown in [Section 10.4.2, “Choosing a Collation ID”](#). The following steps use an ID of 1024.
2. Modify the `Index.xml` and `latin1.xml` configuration files. These files will be located in the directory named by the `character_sets_dir` [488] system variable. You can check the variable value as follows, although the path name might be different on your system:

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+-----+-----+
| Variable_name | Value                               |
+-----+-----+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+-----+-----+
```

3. Choose a name for the collation and list it in the `Index.xml` file. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. For example:

```
<charset name="latin1">
  ...
  <collation name="latin1_test_ci" id="1024"/>
  ...
</charset>
```

4. In the `latin1.xml` configuration file, add a `<collation>` element that names the collation and that contains a `<map>` element that defines a character code-to-weight mapping table for character codes 0 to 255. Each value within the `<map>` element must be a number in hexadecimal format.

```
<collation name="latin1_test_ci">
<map>
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
```

```

44 4E 4F 4F 4F 4F 5C D7 5C 55 55 55 59 59 DE DF
41 41 41 41 5B 5D 5B 43 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5C F7 5C 55 55 55 59 59 DE FF
</map>
</collation>

```

- Restart the server and use this statement to verify that the collation is present:

```

mysql> SHOW COLLATION LIKE 'latin1_test_ci';
+-----+-----+-----+-----+-----+-----+
| Collation      | Charset | Id    | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_test_ci | latin1  | 1024  |         |          | 1       |
+-----+-----+-----+-----+-----+-----+

```

10.4.4. Adding a UCA Collation to a Unicode Character Set

This section describes how to add a UCA collation for a Unicode character set by writing the `<collation>` element within a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. It uses a subset of the Locale Data Markup Language (LDML) specification, which is available at <http://www.unicode.org/reports/tr35/>. With this method, you need not define the entire collation. Instead, you begin with an existing “base” collation and describe the new collation in terms of how it differs from the base collation. The following table lists the base collations of the Unicode character sets for which UCA collations can be defined.

Table 10.1. MySQL Character Sets Available for User-Defined UCA Collations

Character Set	Base Collation
utf8	utf8_unicode_ci
ucs2	ucs2_unicode_ci
utf16	utf16_unicode_ci
utf32	utf32_unicode_ci

The following sections show how to add a collation that is defined using LDML syntax, and provide a summary of LDML rules supported in MySQL.

10.4.4.1. Defining a UCA Collation using LDML Syntax

To add a UCA collation for a Unicode character set without recompiling MySQL, use the following procedure. If you are unfamiliar with the LDML rules used to describe the collation’s sort characteristics, see [Section 10.4.4.2, “LDML Syntax Supported in MySQL”](#).

The example adds a collation named `utf8_phone_ci` to the `utf8` character set. The collation is designed for a scenario involving a Web application for which users post their names and phone numbers. Phone numbers can be given in very different formats:

```

+7-12345-67
+7-12-345-67
+7 12 345 67
+7 (12) 345 67
+71234567

```

The problem raised by dealing with these kinds of values is that the varying permissible formats make searching for a specific phone number very difficult. The solution is to define a new collation that reorders punctuation characters, making them ignorable.

- Choose a collation ID, as shown in [Section 10.4.2, “Choosing a Collation ID”](#). The following steps use an ID of 1029.

- To modify the `Index.xml` configuration file. This file will be located in the directory named by the `character_sets_dir` [488] system variable. You can check the variable value as follows, although the path name might be different on your system:

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+-----+-----+
| Variable_name | Value                                     |
+-----+-----+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+-----+-----+
```

- Choose a name for the collation and list it in the `Index.xml` file. In addition, you'll need to provide the collation ordering rules. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. Within the `<collation>` element, provide a `<rules>` element containing the ordering rules:

```
<charset name="utf8">
...
  <collation name="utf8_phone_ci" id="1029">
    <rules>
      <reset>\u0000</reset>
      <i>\u0020</i> <!-- space -->
      <i>\u0028</i> <!-- left parenthesis -->
      <i>\u0029</i> <!-- right parenthesis -->
      <i>\u002B</i> <!-- plus -->
      <i>\u002D</i> <!-- hyphen -->
    </rules>
  </collation>
...
</charset>
```

- If you want a similar collation for other Unicode character sets, add other `<collation>` elements. For example, to define `ucs2_phone_ci`, add a `<collation>` element to the `<charset name="ucs2">` element. Remember that each collation must have its own unique ID.
- Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION LIKE 'utf8_phone_ci';
+-----+-----+-----+-----+-----+-----+
| Collation      | Charset | Id   | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| utf8_phone_ci | utf8    | 1029 |         |          | 8       |
+-----+-----+-----+-----+-----+-----+
```

Now test the collation to make sure that it has the desired properties.

Create a table containing some sample phone numbers using the new collation:

```
mysql> CREATE TABLE phonebook (
  ->   name VARCHAR(64),
  ->   phone VARCHAR(64) CHARACTER SET utf8 COLLATE utf8_phone_ci
  -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO phonebook VALUES ('Svoj','+7 912 800 80 02');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Hf','+7 (912) 800 80 04');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Bar','+7-912-800-80-01');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO phonebook VALUES ('Ramil','(7912) 800 80 03');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Sanja','+380 (912) 8008005');
Query OK, 1 row affected (0.00 sec)
```

Run some queries to see whether the ignored punctuation characters are in fact ignored for sorting and comparisons:

```
mysql> SELECT * FROM phonebook ORDER BY phone;
+-----+-----+
| name | phone |
+-----+-----+
| Sanja | +380 (912) 8008005 |
| Bar | +7-912-800-80-01 |
| Svoj | +7 912 800 80 02 |
| Ramil | (7912) 800 80 03 |
| Hf | +7 (912) 800 80 04 |
+-----+-----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='+7(912)800-80-01';
+-----+-----+
| name | phone |
+-----+-----+
| Bar | +7-912-800-80-01 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='79128008001';
+-----+-----+
| name | phone |
+-----+-----+
| Bar | +7-912-800-80-01 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='7 9 1 2 8 0 0 8 0 0 1';
+-----+-----+
| name | phone |
+-----+-----+
| Bar | +7-912-800-80-01 |
+-----+-----+
1 row in set (0.00 sec)
```

10.4.4.2. LDML Syntax Supported in MySQL

This section describes the LDML syntax that MySQL recognizes. This is a subset of the syntax described in the LDML specification available at <http://www.unicode.org/reports/tr35/>, which should be consulted for further information. The rules described here are all supported except that character sorting occurs only at the primary level. Rules that specify differences at secondary or higher sort levels are recognized (and thus can be included in collation definitions) but are treated as equality at the primary level.

Character Representation

Characters named in LDML rules can be written in `\unnnn` format, where `nnnn` is the hexadecimal Unicode code point value. Within hexadecimal values, the digits `A` through `F` are not case sensitive; `\u00E1` and `\u00e1` are equivalent. Basic Latin letters `A-Z` and `a-z` can also be written literally (this is a MySQL limitation; the LDML specification permits literal non-Latin1 characters in the rules). Only characters in the Basic Multilingual Plane can be specified. This notation does not apply to characters outside the BMP range of `0000` to `FFFF`.

The `Index.xml` file itself should be written using ASCII encoding.

Syntax Rules

LDML has reset rules and shift rules to specify character ordering. Orderings are given as a set of rules that begin with a reset rule that establishes an anchor point, followed by shift rules that indicate how characters sort relative to the anchor point.

- A `<reset>` rule does not specify any ordering in and of itself. Instead, it “resets” the ordering for subsequent shift rules to cause them to be taken in relation to a given character. Either of the following rules resets subsequent shift rules to be taken in relation to the letter 'A':

```
<reset>A</reset>
<reset>\u0041</reset>
```

- The `<p>`, `<s>`, and `<t>` shift rules define primary, secondary, and tertiary differences of a character from another character:
 - Use primary differences to distinguish separate letters.
 - Use secondary differences to distinguish accent variations.
 - Use tertiary differences to distinguish lettercase variations.

Either of these rules specifies a primary shift rule for the 'G' character:

```
<p>G</p>
<p>\u0047</p>
```

- The `<i>` shift rule indicates that one character sorts identically to another. The following rules cause 'b' to sort the same as 'a':

```
<reset>a</reset>
<i>b</i>
```

The `<i>` shift rule is supported as of MySQL 5.5.3. Prior to 5.5.3, use `<s> ... </s>` instead.

10.5. Character Set Configuration

You can change the default server character set and collation with the `--character-set-server` [438] and `--collation-server` [438] options when you start the server. The collation must be a legal collation for the default character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.) See [Section 5.1.3, “Server Command Options”](#).

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- Your program uses an incorrect path to determine where the character sets are stored (which is typically the `share/mysql/charsets` or `share/charsets` directory under the MySQL installation directory). This can be fixed by using the `--character-sets-dir` option when you run the program in question. For example, to specify a directory to be used by MySQL client programs, list it in the `[client]` group of your option file. The examples given here show what the setting might look like for Unix or Windows, respectively:

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets

[client]
character-sets-dir="C:/Program Files/MySQL/MySQL Server 5.5/share/charsets"
```

- The character set is a complex character set that cannot be loaded dynamically. In this case, you must recompile the program with support for the character set.

For Unicode character sets, you can define collations without recompiling by using LDML notation. See [Section 10.4.4, “Adding a UCA Collation to a Unicode Character Set”](#).

- The character set is a dynamic character set, but you do not have a configuration file for it. In this case, you should install the configuration file for the character set from a new MySQL distribution.
- If your character set index file does not contain the name for the character set, your program displays an error message. The file is named `Index.xml` and the message is:

```
Character set 'charset_name' is not a compiled character set and is not
specified in the '/usr/share/mysql/charsets/Index.xml' file
```

To solve this problem, you should either get a new index file or manually add the name of any missing character sets to the current file.

You can force client programs to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This is normally unnecessary. However, when `character_set_system` [488] differs from `character_set_server` [488] or `character_set_client` [486], and you input characters manually (as database object identifiers, column values, or both), these may be displayed incorrectly in output from the client or the output itself may be formatted incorrectly. In such cases, starting the mysql client with `--default-character-set=system_character_set` [277]—that is, setting the client character set to match the system character set—should fix the problem.

For `MyISAM` tables, you can check the character set name and number for a table with `myisamchk -dvv tbl_name`.

10.6. MySQL Server Time Zone Support

The MySQL server maintains several time zone settings:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` [560] system variable. The value does not change thereafter.

You can set the system time zone for MySQL Server at startup with the `--timezone=timezone_name` [258] option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`. The permissible values for `--timezone` [258] or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

- The server's current time zone. The global `time_zone` [567] system variable indicates the time zone the server currently is operating in. The initial value for `time_zone` [567] is 'SYSTEM', which indicates that the server time zone is the same as the system time zone.

The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone=timezone` [441] option on the command line, or you can use the following line in an option file:

```
default-time-zone='timezone'
```

If you have the `SUPER` [697] privilege, you can set the global server time zone value at runtime with this statement:

```
mysql> SET GLOBAL time_zone = timezone;
```

- Per-connection time zones. Each client that connects has its own time zone setting, given by the session `time_zone` [567] variable. Initially, the session variable takes its value from the global `time_zone` [567] variable, but the client can change its own time zone with this statement:

```
mysql> SET time_zone = timezone;
```

The current session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` [1143] or `CURTIME()` [1135], and values stored in and retrieved from `TIMESTAMP` columns. Values for `TIMESTAMP` columns are converted from the current time zone to UTC for storage, and from UTC to the current time zone for retrieval.

The current time zone setting does not affect values displayed by functions such as `UTC_TIMESTAMP()` [1151] or values in `DATE`, `TIME`, or `DATETIME` columns. Nor are values in those data types stored in UTC; the time zone applies for them only when converting from `TIMESTAMP` values. If you want locale-specific arithmetic for `DATE`, `TIME`, or `DATETIME` values, convert them to UTC, perform the arithmetic, and then convert back.

The current values of the global and client-specific time zones can be retrieved like this:

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

`timezone` values can be given in several formats, none of which are case sensitive:

- The value `'SYSTEM'` indicates that the time zone should be the same as the system time zone.
- The value can be given as a string indicating an offset from UTC, such as `'+10:00'` or `'-6:00'`.
- The value can be given as a named time zone, such as `'Europe/Helsinki'`, `'US/Eastern'`, or `'MET'`. Named time zones can be used only if the time zone information tables in the `mysql` database have been created and populated.

The MySQL installation procedure creates the time zone tables in the `mysql` database, but does not load them. You must do so manually using the following instructions. (If you are upgrading to MySQL 4.1.3 or later from an earlier version, you can create the tables by upgrading your `mysql` database. Use the instructions in [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#). After creating the tables, you can load them.)



Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. For example, the rules for Daylight Saving Time in the United States, Mexico, and parts of Canada changed in 2007. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See the notes at the end of this section.

If your system has its own `zoneinfo` database (the set of files describing time zones), you should use the `mysql_tzinfo_to_sql` program for filling the time zone tables. Examples of such systems are Linux, FreeBSD, Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a `zoneinfo` database, you can use the downloadable package described later in this section.

The `mysql_tzinfo_to_sql` program is used to load the time zone tables. On the command line, pass the `zoneinfo` directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or to generate leap second information:

- To load a single time zone file `tz_file` that corresponds to a time zone name `tz_name`, invoke `mysql_tzinfo_to_sql` like this:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone needs to account for leap seconds, initialize the leap second information like this, where `tz_file` is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

- After running `mysql_tzinfo_to_sql`, it is best to restart the server so that it does not continue to use any previously cached time zone data.

If your system is one that has no zoneinfo database (for example, Windows or HP-UX), you can use the package of pre-built time zone tables that is available for download at the MySQL Developer Zone:

```
http://dev.mysql.com/downloads/timezones.html
```

This time zone package contains `.frm`, `.MYD`, and `.MYI` files for the `MyISAM` time zone tables. These tables should be part of the `mysql` database, so you should place the files in the `mysql` subdirectory of your MySQL server's data directory. The server should be stopped while you do this and restarted afterward.



Warning

Do not use the downloadable package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

For information about time zone settings in replication setup, please see [Section 16.4.1, “Replication Features and Issues”](#).

10.6.1. Staying Current with Time Zone Changes

As mentioned earlier, when the time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are two factors to consider in staying current:

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time changes. Check the Web site for your operating system vendor for an update that addresses the time changes.
- If you replace the system's `/etc/localtime` timezone file with a version that uses rules differing from those in effect at `mysqld` startup, you should restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.

- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date. If your system has its own zoneinfo database, you should reload the MySQL time zone tables whenever the zoneinfo database is updated, using the instructions given earlier in this section. For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace your current time zone tables. `mysqld` caches time zone information that it looks up, so after replacing the time zone tables, you should restart `mysqld` to make sure that it does not continue to serve outdated time zone data.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+-----+
| COUNT(*) |
+-----+
|          0 |
+-----+
```

A count of zero indicates that the table is empty. In this case, no one can be using named time zones, and you don't need to update the tables. A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, you should be sure to reload your time zone tables so that anyone who uses named time zones will get correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses these two queries:

```
SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you would see an incorrect result like this:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+-----+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+-----+
| 2007-03-11 01:00:00 |
+-----+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+-----+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+-----+
| 2007-03-11 02:00:00 |
+-----+
```

After updating the tables, you should see the correct result:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+-----+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+-----+
| 2007-03-11 01:00:00 |
+-----+
```

```
mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+-----+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+-----+
| 2007-03-11 01:00:00 |
+-----+
```

10.6.2. Time Zone Leap Second Support

Leap second values are returned with a time part that ends with `:59:59`. This means that a function such as `NOW()` [1143] can return the same value for two or three consecutive seconds during the leap second. It remains true that literal temporal values having a time part that ends with `:59:60` or `:59:61` are considered invalid.

If it is necessary to search for `TIMESTAMP` values one second before the leap second, anomalous results may be obtained if you use a comparison with `'YYYY-MM-DD hh:mm:ss'` values. The following example demonstrates this. It changes the local time zone to UTC so there is no difference between internal values (which are in UTC) and displayed values (which have time zone correction applied).

```
mysql> CREATE TABLE t1 (
->   a INT,
->   ts TIMESTAMP DEFAULT NOW(),
->   PRIMARY KEY (ts)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> -- change to UTC
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:59'
mysql> SET timestamp = 1230767999;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:60'
mysql> SET timestamp = 1230768000;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> -- values differ internally but display the same
mysql> SELECT a, ts, UNIX_TIMESTAMP(ts) FROM t1;
+-----+-----+-----+
| a | ts | UNIX_TIMESTAMP(ts) |
+-----+-----+-----+
| 1 | 2008-12-31 23:59:59 | 1230767999 |
| 2 | 2008-12-31 23:59:59 | 1230768000 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> -- only the non-leap value matches
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:59';
+-----+-----+
| a | ts |
+-----+-----+
| 1 | 2008-12-31 23:59:59 |
+-----+-----+
1 row in set (0.00 sec)

mysql> -- the leap value with seconds=60 is invalid
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:60';
```

```
Empty set, 2 warnings (0.00 sec)
```

To work around this, you can use a comparison based on the UTC value actually stored in column, which has the leap second correction applied:

```
mysql> -- selecting using UNIX_TIMESTAMP value return leap value
mysql> SELECT * FROM t1 WHERE UNIX_TIMESTAMP(ts) = 1230768000;
+-----+-----+
| a     | ts                               |
+-----+-----+
| 2     | 2008-12-31 23:59:59             |
+-----+-----+
1 row in set (0.00 sec)
```

10.7. MySQL Server Locale Support

The locale indicated by the `lc_time_names` [512] system variable controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()` [1138], `DAYNAME()` [1140], and `MONTHNAME()` [1143] functions.

The `lc_time_names` [512] value does not affect the result from `FORMAT()` [1103], but this function takes an optional third parameter that enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` [512] system variable.

Locale names have language and region subtags listed by IANA (<http://www.iana.org/assignments/language-subtag-registry>) such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting, but you can set the value at server startup or set the `GLOBAL` value if you have the `SUPER` [697] privilege. Any client can examine the value of `lc_time_names` [512] or set its `SESSION` value to affect the locale for its own connection.

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----+
| @@lc_time_names |
+-----+
| en_US           |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| Friday                | January                  |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| Friday Fri January Jan                   |
+-----+
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
+-----+
```

```

| @@lc_time_names |
+-----+
| es_MX           |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| viernes                | enero                    |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| viernes vie enero ene                    |
+-----+
1 row in set (0.00 sec)

```

The day or month name for each of the affected functions is converted from `utf8` to the character set indicated by the `character_set_connection` [487] system variable.

`lc_time_names` [512] may be set to any of the following locale values. The set of locales supported by MySQL may differ from those supported by your operating system.

<code>ar_AE</code> : Arabic - United Arab Emirates	<code>ar_BH</code> : Arabic - Bahrain
<code>ar_DZ</code> : Arabic - Algeria	<code>ar_EG</code> : Arabic - Egypt
<code>ar_IN</code> : Arabic - India	<code>ar_IQ</code> : Arabic - Iraq
<code>ar_JO</code> : Arabic - Jordan	<code>ar_KW</code> : Arabic - Kuwait
<code>ar_LB</code> : Arabic - Lebanon	<code>ar_LY</code> : Arabic - Libya
<code>ar_MA</code> : Arabic - Morocco	<code>ar_OM</code> : Arabic - Oman
<code>ar_QA</code> : Arabic - Qatar	<code>ar_SA</code> : Arabic - Saudi Arabia
<code>ar_SD</code> : Arabic - Sudan	<code>ar_SY</code> : Arabic - Syria
<code>ar_TN</code> : Arabic - Tunisia	<code>ar_YE</code> : Arabic - Yemen
<code>be_BY</code> : Belarusian - Belarus	<code>bg_BG</code> : Bulgarian - Bulgaria
<code>ca_ES</code> : Catalan - Spain	<code>cs_CZ</code> : Czech - Czech Republic
<code>da_DK</code> : Danish - Denmark	<code>de_AT</code> : German - Austria
<code>de_BE</code> : German - Belgium	<code>de_CH</code> : German - Switzerland
<code>de_DE</code> : German - Germany	<code>de_LU</code> : German - Luxembourg
<code>el_GR</code> : Greek - Greece	<code>en_AU</code> : English - Australia
<code>en_CA</code> : English - Canada	<code>en_GB</code> : English - United Kingdom
<code>en_IN</code> : English - India	<code>en_NZ</code> : English - New Zealand
<code>en_PH</code> : English - Philippines	<code>en_US</code> : English - United States
<code>en_ZA</code> : English - South Africa	<code>en_ZW</code> : English - Zimbabwe
<code>es_AR</code> : Spanish - Argentina	<code>es_BO</code> : Spanish - Bolivia
<code>es_CL</code> : Spanish - Chile	<code>es_CO</code> : Spanish - Columbia
<code>es_CR</code> : Spanish - Costa Rica	<code>es_DO</code> : Spanish - Dominican Republic

es_EC : Spanish - Ecuador	es_ES : Spanish - Spain
es_GT : Spanish - Guatemala	es_HN : Spanish - Honduras
es_MX : Spanish - Mexico	es_NI : Spanish - Nicaragua
es_PA : Spanish - Panama	es_PE : Spanish - Peru
es_PR : Spanish - Puerto Rico	es_PY : Spanish - Paraguay
es_SV : Spanish - El Salvador	es_US : Spanish - United States
es_UY : Spanish - Uruguay	es_VE : Spanish - Venezuela
et_EE : Estonian - Estonia	eu_ES : Basque - Basque
fi_FI : Finnish - Finland	fo_FO : Faroese - Faroe Islands
fr_BE : French - Belgium	fr_CA : French - Canada
fr_CH : French - Switzerland	fr_FR : French - France
fr_LU : French - Luxembourg	gl_ES : Galician - Spain
gu_IN : Gujarati - India	he_IL : Hebrew - Israel
hi_IN : Hindi - India	hr_HR : Croatian - Croatia
hu_HU : Hungarian - Hungary	id_ID : Indonesian - Indonesia
is_IS : Icelandic - Iceland	it_CH : Italian - Switzerland
it_IT : Italian - Italy	ja_JP : Japanese - Japan
ko_KR : Korean - Republic of Korea	lt_LT : Lithuanian - Lithuania
lv_LV : Latvian - Latvia	mk_MK : Macedonian - FYROM
mn_MN : Mongolia - Mongolian	ms_MY : Malay - Malaysia
nb_NO : Norwegian(Bokmål) - Norway	nl_BE : Dutch - Belgium
nl_NL : Dutch - The Netherlands	no_NO : Norwegian - Norway
pl_PL : Polish - Poland	pt_BR : Portugese - Brazil
pt_PT : Portugese - Portugal	ro_RO : Romanian - Romania
ru_RU : Russian - Russia	ru_UA : Russian - Ukraine
sk_SK : Slovak - Slovakia	sl_SI : Slovenian - Slovenia
sq_AL : Albanian - Albania	sr_RS : Serbian - Yugoslavia
sv_FI : Swedish - Finland	sv_SE : Swedish - Sweden
ta_IN : Tamil - India	te_IN : Telugu - India
th_TH : Thai - Thailand	tr_TR : Turkish - Turkey
uk_UA : Ukrainian - Ukraine	ur_PK : Urdu - Pakistan
vi_VN : Vietnamese - Viet Nam	zh_CN : Chinese - China
zh_HK : Chinese - Hong Kong	zh_TW : Chinese - Taiwan Province of China

[lc_time_names](#) [512] currently does not affect the [STR_TO_DATE\(\)](#) [1144] or [GET_FORMAT\(\)](#) [1141] function.

Chapter 11. Data Types

Table of Contents

11.1. Data Type Overview	1036
11.1.1. Numeric Type Overview	1036
11.1.2. Date and Time Type Overview	1039
11.1.3. String Type Overview	1041
11.2. Numeric Types	1044
11.2.1. Integer Types (Exact Value) - <code>INTEGER</code> , <code>INT</code> , <code>SMALLINT</code> , <code>TINYINT</code> , <code>MEDIUMINT</code> , <code>BIGINT</code>	1044
11.2.2. Fixed-Point Types (Exact Value) - <code>DECIMAL</code> , <code>NUMERIC</code>	1045
11.2.3. Floating-Point Types (Approximate Value) - <code>FLOAT</code> , <code>DOUBLE</code>	1045
11.2.4. Bit-Value Type - <code>BIT</code>	1046
11.2.5. Numeric Type Attributes	1046
11.2.6. Out-of-Range and Overflow Handling	1047
11.3. Date and Time Types	1048
11.3.1. The <code>DATE</code> , <code>DATETIME</code> , and <code>TIMESTAMP</code> Types	1050
11.3.2. The <code>TIME</code> Type	1051
11.3.3. The <code>YEAR</code> Type	1051
11.3.4. <code>YEAR(2)</code> Limitations and Migrating to <code>YEAR(4)</code>	1052
11.3.5. Automatic Initialization and Updating for <code>TIMESTAMP</code>	1054
11.3.6. Fractional Seconds in Time Values	1057
11.3.7. Conversion Between Date and Time Types	1057
11.3.8. Two-Digit Years in Dates	1058
11.4. String Types	1058
11.4.1. The <code>CHAR</code> and <code>VARCHAR</code> Types	1058
11.4.2. The <code>BINARY</code> and <code>VARBINARY</code> Types	1060
11.4.3. The <code>BLOB</code> and <code>TEXT</code> Types	1061
11.4.4. The <code>ENUM</code> Type	1062
11.4.5. The <code>SET</code> Type	1065
11.5. Data Type Default Values	1068
11.6. Data Type Storage Requirements	1069
11.7. Choosing the Right Type for a Column	1072
11.8. Using Data Types from Other Database Engines	1072

MySQL supports a number of data types in several categories: numeric types, date and time types, and string (character and byte) types. This chapter provides an overview of these data types, a more detailed description of the properties of the types in each category, and a summary of the data type storage requirements. The initial overview is intentionally brief. The more detailed descriptions later in the chapter should be consulted for additional information about particular data types, such as the permissible formats in which you can specify values.

MySQL also supports extensions for handling spatial data. For information about these data types, see [Section 12.17, “Spatial Extensions”](#).

Data type descriptions use these conventions:

- *M* indicates the maximum display width for integer types. For floating-point and fixed-point types, *M* is the total number of digits that can be stored (the precision). For string types, *M* is the maximum length. The maximum permissible value of *M* depends on the data type.

- `D` applies to floating-point and fixed-point types and indicates the number of digits following the decimal point (the scale). The maximum possible value is 30, but should be no greater than $M-2$.
- Square brackets (“[” and “]”) indicate optional parts of type definitions.

11.1. Data Type Overview

11.1.1. Numeric Type Overview

A summary of the numeric data types follows. For additional information about properties and storage requirements of the numeric types, see [Section 11.2, “Numeric Types”](#), and [Section 11.6, “Data Type Storage Requirements”](#).

`M` indicates the maximum display width for integer types. The maximum display width is 255. Display width is unrelated to the range of values a type can contain, as described in [Section 11.2, “Numeric Types”](#). For floating-point and fixed-point types, `M` is the total number of digits that can be stored.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

Numeric data types that permit the `UNSIGNED` attribute also permit `SIGNED`. However, these data types are signed by default, so the `SIGNED` attribute has no effect.

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

`SERIAL DEFAULT VALUE` in the definition of an integer column is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.



Warning

When you use subtraction between integer values where one is of type `UNSIGNED`, the result is unsigned unless the `NO_UNSIGNED_SUBTRACTION` [614] SQL mode is enabled. See [Section 12.10, “Cast Functions and Operators”](#).

- `BIT[(M)]`

A bit-field type. `M` indicates the number of bits per value, from 1 to 64. The default is 1 if `M` is omitted.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

A very small integer. The signed range is `-128` to `127`. The unsigned range is `0` to `255`.

- `BOOL, BOOLEAN`

These types are synonyms for `TINYINT(1)`. A value of zero is considered false. Nonzero values are considered true:

```
mysql> SELECT IF(0, 'true', 'false');
+-----+
| IF(0, 'true', 'false') |
+-----+
| false                   |
+-----+

mysql> SELECT IF(1, 'true', 'false');
+-----+
| IF(1, 'true', 'false') |
+-----+
| true                   |
+-----+
```

```
mysql> SELECT IF(2, 'true', 'false');
+-----+
| IF(2, 'true', 'false') |
+-----+
| true                    |
+-----+
```

However, the values `TRUE` and `FALSE` are merely aliases for `1` and `0`, respectively, as shown here:

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+-----+
| IF(0 = FALSE, 'true', 'false') |
+-----+
| true                            |
+-----+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-----+
| IF(1 = TRUE, 'true', 'false') |
+-----+
| true                            |
+-----+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-----+
| IF(2 = TRUE, 'true', 'false') |
+-----+
| false                          |
+-----+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+-----+
| IF(2 = FALSE, 'true', 'false') |
+-----+
| false                          |
+-----+
```

The last two statements display the results shown because `2` is equal to neither `1` nor `0`.

- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`

A small integer. The signed range is `-32768` to `32767`. The unsigned range is `0` to `65535`.

- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`

A medium-sized integer. The signed range is `-8388608` to `8388607`. The unsigned range is `0` to `16777215`.

- `INT[(M)] [UNSIGNED] [ZEROFILL]`

A normal-size integer. The signed range is `-2147483648` to `2147483647`. The unsigned range is `0` to `4294967295`.

- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]`

This type is a synonym for `INT`.

- `BIGINT[(M)] [UNSIGNED] [ZEROFILL]`

A large integer. The signed range is `-9223372036854775808` to `9223372036854775807`. The unsigned range is `0` to `18446744073709551615`.

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

Some things you should be aware of with respect to `BIGINT` columns:

- All arithmetic is done using signed `BIGINT` or `DOUBLE` values, so you should not use unsigned big integers larger than `9223372036854775807` (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a `BIGINT` value to a `DOUBLE`.

MySQL can handle `BIGINT` in the following cases:

- When using integers to store large unsigned values in a `BIGINT` column.
- In `MIN(col_name) [1206]` or `MAX(col_name) [1205]`, where `col_name` refers to a `BIGINT` column.
- When using operators (`+ [1121]`, `- [1122]`, `* [1122]`, and so on) where both operands are integers.
- You can always store an exact integer value in a `BIGINT` column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
- The `- [1122]`, `+ [1121]`, and `* [1122]` operators use `BIGINT` arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than `9223372036854775807`.
- `DECIMAL(M[,D]) [UNSIGNED] [ZEROFILL]`

A packed “exact” fixed-point number. `M` is the total number of digits (the precision) and `D` is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the “-” sign are not counted in `M`. If `D` is 0, values have no decimal point or fractional part. The maximum number of digits (`M`) for `DECIMAL` is 65. The maximum number of supported decimals (`D`) is 30. If `D` is omitted, the default is 0. If `M` is omitted, the default is 10.

`UNSIGNED`, if specified, disallows negative values.

All basic calculations (`+`, `-`, `*`, `/`) with `DECIMAL` columns are done with a precision of 65 digits.

- `DEC(M[,D]) [UNSIGNED] [ZEROFILL]`, `NUMERIC(M[,D]) [UNSIGNED] [ZEROFILL]`, `FIXED(M[,D]) [UNSIGNED] [ZEROFILL]`

These types are synonyms for `DECIMAL`. The `FIXED` synonym is available for compatibility with other database systems.

- `FLOAT(M,D) [UNSIGNED] [ZEROFILL]`

A small (single-precision) floating-point number. Permissible values are `-3.402823466E+38` to `-1.175494351E-38`, 0, and `1.175494351E-38` to `3.402823466E+38`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

`M` is the total number of digits and `D` is the number of digits following the decimal point. If `M` and `D` are omitted, values are stored to the limits permitted by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

`UNSIGNED`, if specified, disallows negative values.

Using `FLOAT` might give you some unexpected problems because all calculations in MySQL are done with double precision. See [Section C.5.5.7, “Solving Problems with No Matching Rows”](#).

- `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`

A normal-size (double-precision) floating-point number. Permissible values are `-1.7976931348623157E+308` to `-2.2250738585072014E-308`, `0`, and `2.2250738585072014E-308` to `1.7976931348623157E+308`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

`M` is the total number of digits and `D` is the number of digits following the decimal point. If `M` and `D` are omitted, values are stored to the limits permitted by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

`UNSIGNED`, if specified, disallows negative values.

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`, `REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

These types are synonyms for `DOUBLE`. Exception: If the `REAL_AS_FLOAT [616]` SQL mode is enabled, `REAL` is a synonym for `FLOAT` rather than `DOUBLE`.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

A floating-point number. `p` represents the precision in bits, but MySQL uses this value only to determine whether to use `FLOAT` or `DOUBLE` for the resulting data type. If `p` is from 0 to 24, the data type becomes `FLOAT` with no `M` or `D` values. If `p` is from 25 to 53, the data type becomes `DOUBLE` with no `M` or `D` values. The range of the resulting column is the same as for the single-precision `FLOAT` or double-precision `DOUBLE` data types described earlier in this section.

`FLOAT(p)` syntax is provided for ODBC compatibility.

11.1.2. Date and Time Type Overview

A summary of the temporal data types follows. For additional information about properties and storage requirements of the temporal types, see [Section 11.3, “Date and Time Types”](#), and [Section 11.6, “Data Type Storage Requirements”](#). For descriptions of functions that operate on temporal values, see [Section 12.7, “Date and Time Functions”](#).

For the `DATE` and `DATETIME` range descriptions, “supported” means that although earlier values might work, there is no guarantee.

- `DATE`

A date. The supported range is `'1000-01-01'` to `'9999-12-31'`. MySQL displays `DATE` values in `'YYYY-MM-DD'` format, but permits assignment of values to `DATE` columns using either strings or numbers.

- `DATETIME`

A date and time combination. The supported range is `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`. MySQL displays `DATETIME` values in `'YYYY-MM-DD HH:MM:SS'` format, but permits assignment of values to `DATETIME` columns using either strings or numbers.

- `TIMESTAMP`

A timestamp. The range is '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC. `TIMESTAMP` values are stored as the number of seconds since the epoch ('1970-01-01 00:00:00' UTC). A `TIMESTAMP` cannot represent the value '1970-01-01 00:00:00' because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing '0000-00-00 00:00:00', the “zero” `TIMESTAMP` value.

Unless specified otherwise, the first `TIMESTAMP` column in a table is defined to be automatically set to the date and time of the most recent modification if not explicitly assigned a value. This makes `TIMESTAMP` useful for recording the timestamp of an `INSERT` or `UPDATE` operation. You can also set any `TIMESTAMP` column to the current date and time by assigning it a `NULL` value, unless it has been defined with the `NULL` attribute to permit `NULL` values. The automatic initialization and updating to the current date and time can be specified using `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses, as described in [Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP`”](#).



Note

The `TIMESTAMP` format that was used prior to MySQL 4.1 is not supported in MySQL 5.5; see *MySQL 3.23, 4.0, 4.1 Reference Manual* for information regarding the old format.

- `TIME`

A time. The range is '-838:59:59' to '838:59:59'. MySQL displays `TIME` values in 'HH:MM:SS' format, but permits assignment of values to `TIME` columns using either strings or numbers.

- `YEAR[(2|4)]`

A year in two-digit or four-digit format. The default is four-digit format. `YEAR(2)` or `YEAR(4)` differ in display format, but have the same range of values. In four-digit format, values display as 1901 to 2155, and 0000. In two-digit format, values display as 70 to 69, representing years from 1970 to 2069. MySQL displays `YEAR` values in `YYYY` or `YY` format, but permits assignment of values to `YEAR` columns using either strings or numbers.



Note

The `YEAR(2)` data type has certain issues that you should consider before choosing to use it. As of MySQL 5.5.27, `YEAR(2)` is deprecated. For more information, see [Section 11.3.4, “`YEAR\(2\)` Limitations and Migrating to `YEAR\(4\)`”](#).

For additional information about `YEAR` display format and interpretation of input values, see [Section 11.3.3, “The `YEAR` Type”](#).

The `SUM()` [\[1206\]](#) and `AVG()` [\[1203\]](#) aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```



Note

The MySQL server can be run with the `MAXDB` [\[618\]](#) SQL mode enabled. In this case, `TIMESTAMP` is identical with `DATETIME`. If this mode is enabled at the time

that a table is created, `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and there is no automatic initialization or updating to the current date and time. See [Section 5.1.7, “Server SQL Modes”](#).

11.1.3. String Type Overview

A summary of the string data types follows. For additional information about properties and storage requirements of the string types, see [Section 11.4, “String Types”](#), and [Section 11.6, “Data Type Storage Requirements”](#).

In some cases, MySQL may change a string column to a type different from that given in a `CREATE TABLE` or `ALTER TABLE` statement. See [Section 13.1.17.2, “Silent Column Specification Changes”](#).

MySQL interprets length specifications in character column definitions in character units. This applies to `CHAR`, `VARCHAR`, and the `TEXT` types.

Column definitions for many string data types can include attributes that specify the character set or collation of the column. These attributes apply to the `CHAR`, `VARCHAR`, the `TEXT` types, `ENUM`, and `SET` data types:

- The `CHARACTER SET` attribute specifies the character set, and the `COLLATE` attribute specifies a collation for the character set. For example:

```
CREATE TABLE t
(
  c1 VARCHAR(20) CHARACTER SET utf8,
  c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
);
```

This table definition creates a column named `c1` that has a character set of `utf8` with the default collation for that character set, and a column named `c2` that has a character set of `latin1` and a case-sensitive collation.

The rules for assigning the character set and collation when either or both of the `CHARACTER SET` and `COLLATE` attributes are missing are described in [Section 10.1.3.4, “Column Character Set and Collation”](#).

`CHARSET` is a synonym for `CHARACTER SET`.

- Specifying the `CHARACTER SET binary` attribute for a character data type causes the column to be created as the corresponding binary data type: `CHAR` becomes `BINARY`, `VARCHAR` becomes `VARBINARY`, and `TEXT` becomes `BLOB`. For the `ENUM` and `SET` data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

- The `ASCII` attribute is shorthand for `CHARACTER SET latin1`.
- The `UNICODE` attribute is shorthand for `CHARACTER SET ucs2`.
- The `BINARY` attribute is shorthand for specifying the binary collation of the column character set. In this case, sorting and comparison are based on numeric character values.

Character column sorting and comparison are based on the character set assigned to the column. For the `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, and `SET` data types, you can declare a column with a binary collation or the `BINARY` attribute to cause sorting and comparison to use the underlying character code values rather than a lexical ordering.

Section 10.1, “Character Set Support”, provides additional information about use of character sets in MySQL.

- `[NATIONAL] CHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

A fixed-length string that is always right-padded with spaces to the specified length when stored. *M* represents the column length in characters. The range of *M* is 0 to 255. If *M* is omitted, the length is 1.



Note

Trailing spaces are removed when `CHAR` values are retrieved unless the `PAD_CHAR_TO_FULL_LENGTH` [616] SQL mode is enabled.

`CHAR` is shorthand for `CHARACTER`. `NATIONAL CHAR` (or its equivalent short form, `NCHAR`) is the standard SQL way to define that a `CHAR` column should use some predefined character set. MySQL 4.1 and up uses `utf8` as this predefined character set. Section 10.1.3.6, “National Character Set”.

The `CHAR BYTE` data type is an alias for the `BINARY` data type. This is a compatibility feature.

MySQL permits you to create a column of type `CHAR(0)`. This is useful primarily when you have to be compliant with old applications that depend on the existence of a column but that do not actually use its value. `CHAR(0)` is also quite nice when you need a column that can take only two values: A column that is defined as `CHAR(0) NULL` occupies only one bit and can take only the values `NULL` and `''` (the empty string).

- `[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

A variable-length string. *M* represents the maximum column length in characters. The range of *M* is 0 to 65,535. The effective maximum length of a `VARCHAR` is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. For example, `utf8` characters can require up to three bytes per character, so a `VARCHAR` column that uses the `utf8` character set can be declared to be a maximum of 21,844 characters. See Section E.10.4, “Table Column-Count and Row-Size Limits”.

MySQL stores `VARCHAR` values as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A `VARCHAR` column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.



Note

MySQL 5.5 follows the standard SQL specification, and does *not* remove trailing spaces from `VARCHAR` values.

`VARCHAR` is shorthand for `CHARACTER VARYING`. `NATIONAL VARCHAR` is the standard SQL way to define that a `VARCHAR` column should use some predefined character set. MySQL 4.1 and up uses `utf8` as this predefined character set. [Section 10.1.3.6, “National Character Set”](#). `NVARCHAR` is shorthand for `NATIONAL VARCHAR`.

- `BINARY(M)`

The `BINARY` type is similar to the `CHAR` type, but stores binary byte strings rather than nonbinary character strings. `M` represents the column length in bytes.

- `VARBINARY(M)`

The `VARBINARY` type is similar to the `VARCHAR` type, but stores binary byte strings rather than nonbinary character strings. `M` represents the maximum column length in bytes.

- `TINYBLOB`

A `BLOB` column with a maximum length of 255 ($2^8 - 1$) bytes. Each `TINYBLOB` value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- `TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

A `TEXT` column with a maximum length of 255 ($2^8 - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. Each `TINYTEXT` value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- `BLOB(M)`

A `BLOB` column with a maximum length of 65,535 ($2^{16} - 1$) bytes. Each `BLOB` value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

An optional length `M` can be given for this type. If this is done, MySQL creates the column as the smallest `BLOB` type large enough to hold values `M` bytes long.

- `TEXT(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

A `TEXT` column with a maximum length of 65,535 ($2^{16} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. Each `TEXT` value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

An optional length `M` can be given for this type. If this is done, MySQL creates the column as the smallest `TEXT` type large enough to hold values `M` characters long.

- `MEDIUMBLOB`

A `BLOB` column with a maximum length of 16,777,215 ($2^{24} - 1$) bytes. Each `MEDIUMBLOB` value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- `MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

A `TEXT` column with a maximum length of 16,777,215 ($2^{24} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. Each `MEDIUMTEXT` value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- `LOBLOB`

A `BLOB` column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) bytes. The effective maximum length of `LOBLOB` columns depends on the configured maximum packet size in the client/server

protocol and available memory. Each `LONGBLOB` value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- `LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

A `TEXT` column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. The effective maximum length of `LONGTEXT` columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each `LONGTEXT` value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- `ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]`

An enumeration. A string object that can have only one value, chosen from the list of values `'value1'`, `'value2'`, ..., `NULL` or the special `''` error value. An `ENUM` column can have a maximum of 65,535 distinct values. `ENUM` values are represented internally as integers.

- `SET('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]`

A set. A string object that can have zero or more values, each of which must be chosen from the list of values `'value1'`, `'value2'`, ... A `SET` column can have a maximum of 64 members. `SET` values are represented internally as integers.

11.2. Numeric Types

MySQL supports all standard SQL numeric data types. These types include the exact numeric data types (`INTEGER`, `SMALLINT`, `DECIMAL`, and `NUMERIC`), as well as the approximate numeric data types (`FLOAT`, `REAL`, and `DOUBLE PRECISION`). The keyword `INT` is a synonym for `INTEGER`, and the keywords `DEC` and `FIXED` are synonyms for `DECIMAL`. MySQL treats `DOUBLE` as a synonym for `DOUBLE PRECISION` (a nonstandard extension). MySQL also treats `REAL` as a synonym for `DOUBLE PRECISION` (a nonstandard variation), unless the `REAL_AS_FLOAT` [616] SQL mode is enabled.

The `BIT` data type stores bit-field values and is supported for `MyISAM`, `MEMORY`, `InnoDB`, and `NDBCLUSTER` tables.

For information about how MySQL handles assignment of out-of-range values to columns and overflow during expression evaluation, see [Section 11.2.6, “Out-of-Range and Overflow Handling”](#).

For information about numeric type storage requirements, see [Section 11.6, “Data Type Storage Requirements”](#).

The data type used for the result of a calculation on numeric operands depends on the types of the operands and the operations performed on them. For more information, see [Section 12.6.1, “Arithmetic Operators”](#).

11.2.1. Integer Types (Exact Value) - `INTEGER`, `INT`, `SMALLINT`, `TINYINT`, `MEDIUMINT`, `BIGINT`

MySQL supports the SQL standard integer types `INTEGER` (or `INT`) and `SMALLINT`. As an extension to the standard, MySQL also supports the integer types `TINYINT`, `MEDIUMINT`, and `BIGINT`. The following table shows the required storage and range for each integer type.

Type	Storage (Bytes)	Minimum Value (Signed/Unsigned)	Maximum Value (Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

11.2.2. Fixed-Point Types (Exact Value) - [DECIMAL](#), [NUMERIC](#)

The [DECIMAL](#) and [NUMERIC](#) types store exact numeric data values. These types are used when it is important to preserve exact precision, for example with monetary data. In MySQL, [NUMERIC](#) is implemented as [DECIMAL](#), so the following remarks about [DECIMAL](#) apply equally to [NUMERIC](#).

MySQL 5.5 stores [DECIMAL](#) values in binary format. See [Section 12.18, “Precision Math”](#).

In a [DECIMAL](#) column declaration, the precision and scale can be (and usually is) specified; for example:

```
salary DECIMAL(5,2)
```

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.

Standard SQL requires that [DECIMAL\(5,2\)](#) be able to store any value with five digits and two decimals, so values that can be stored in the `salary` column range from `-999.99` to `999.99`.

In standard SQL, the syntax [DECIMAL\(M\)](#) is equivalent to [DECIMAL\(M,0\)](#). Similarly, the syntax [DECIMAL](#) is equivalent to [DECIMAL\(M,0\)](#), where the implementation is permitted to decide the value of *M*. MySQL supports both of these variant forms of [DECIMAL](#) syntax. The default value of *M* is 10.

If the scale is 0, [DECIMAL](#) values contain no decimal point or fractional part.

The maximum number of digits for [DECIMAL](#) is 65, but the actual range for a given [DECIMAL](#) column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are permitted by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the permissible number of digits.)

11.2.3. Floating-Point Types (Approximate Value) - [FLOAT](#), [DOUBLE](#)

The [FLOAT](#) and [DOUBLE](#) types represent approximate numeric data values. MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

For [FLOAT](#), the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword [FLOAT](#) in parentheses. MySQL also supports this optional precision

specification, but the precision value is used only to determine storage size. A precision from 0 to 23 results in a 4-byte single-precision `FLOAT` column. A precision from 24 to 53 results in an 8-byte double-precision `DOUBLE` column.

MySQL permits a nonstandard syntax: `FLOAT(M,D)` or `REAL(M,D)` or `DOUBLE PRECISION(M,D)`. Here, “(M,D)” means that values can be stored with up to *M* digits in total, of which *D* digits may be after the decimal point. For example, a column defined as `FLOAT(7,4)` will look like `-999.9999` when displayed. MySQL performs rounding when storing values, so if you insert `999.00009` into a `FLOAT(7,4)` column, the approximate result is `999.0001`.

Because floating-point values are approximate and not stored as exact values, attempts to treat them as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. For more information, see [Section C.5.5.8, “Problems with Floating-Point Values”](#)

For maximum portability, code requiring storage of approximate numeric data values should use `FLOAT` or `DOUBLE PRECISION` with no specification of precision or number of digits.

11.2.4. Bit-Value Type - BIT

The `BIT` data type is used to store bit-field values. A type of `BIT(M)` enables storage of *M*-bit values. *M* can range from 1 to 64.

To specify bit values, `b'value'` notation can be used. *value* is a binary value written using zeros and ones. For example, `b'111'` and `b'10000000'` represent 7 and 128, respectively. See [Section 9.1.6, “Bit-Field Literals”](#).

If you assign a value to a `BIT(M)` column that is less than *M* bits long, the value is padded on the left with zeros. For example, assigning a value of `b'101'` to a `BIT(6)` column is, in effect, the same as assigning `b'000101'`.

11.2.5. Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, `INT(4)` specifies an `INT` with a display width of four digits. This optional display width may be used by applications to display integer values having a width less than the width specified for the column by left-padding them with spaces. (That is, this width is present in the metadata returned with result sets. Whether it is used or not is up to the application.)

The display width does *not* constrain the range of values that can be stored in the column. Nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as `SMALLINT(3)` has the usual `SMALLINT` range of `-32768` to `32767`, and values outside the range permitted by three digits are displayed in full using more than three digits.

When used in conjunction with the optional (nonstandard) attribute `ZEROFILL`, the default padding of spaces is replaced with zeros. For example, for a column declared as `INT(4) ZEROFILL`, a value of `5` is retrieved as `0005`.



Note

The `ZEROFILL` attribute is ignored when a column is involved in expressions or `UNION` queries.

If you store values larger than the display width in an integer column that has the `ZEROFILL` attribute, you may experience problems when MySQL generates temporary tables for some complicated joins. In these cases, MySQL assumes that the data values fit within the column display width.

All integer types can have an optional (nonstandard) attribute `UNSIGNED`. Unsigned type can be used to permit only nonnegative numbers in a column or when you need a larger upper numeric range for the column. For example, if an `INT` column is `UNSIGNED`, the size of the column's range is the same but its endpoints shift from `-2147483648` and `2147483647` up to `0` and `4294967295`.

Floating-point and fixed-point types also can be `UNSIGNED`. As with integer types, this attribute prevents negative values from being stored in the column. Unlike the integer types, the upper range of column values remains the same.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

Integer or floating-point data types can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is `value+1`, where `value` is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`.

11.2.6. Out-of-Range and Overflow Handling

When MySQL stores a value in a numeric column that is outside the permissible range of the column data type, the result depends on the SQL mode in effect at the time:

- If strict SQL mode is enabled, MySQL rejects the out-of-range value with an error, and the insert fails, in accordance with the SQL standard.
- If no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the range and stores the resulting value instead.

When an out-of-range value is assigned to an integer column, MySQL stores the value representing the corresponding endpoint of the column data type range. If you store `256` into a `TINYINT` or `TINYINT UNSIGNED` column, MySQL stores `127` or `255`, respectively.

When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Column-assignment conversions that occur due to clipping when MySQL is not operating in strict mode are reported as warnings for `ALTER TABLE`, `LOAD DATA INFILE`, `UPDATE`, and multiple-row `INSERT` statements. In strict mode, these statements fail, and some or all the values will not be inserted or changed, depending on whether the table is a transactional table and other factors. For details, see [Section 5.1.7, "Server SQL Modes"](#).

As of MySQL 5.5.5, overflow during numeric expression evaluation results in an error. For example, the largest signed `BIGINT` value is `9223372036854775807`, so the following expression produces an error:

```
mysql> SELECT 9223372036854775807 + 1;
ERROR 1690 (22003): BIGINT value is out of range in '(9223372036854775807 + 1)'
```

To enable the operation to succeed in this case, convert the value to unsigned;

```
mysql> SELECT CAST(9223372036854775807 AS UNSIGNED) + 1;
+-----+
| CAST(9223372036854775807 AS UNSIGNED) + 1 |
+-----+
|                               9223372036854775808 |
+-----+
```

Whether overflow occurs depends on the range of the operands, so another way to handle the preceding expression is to use exact-value arithmetic because `DECIMAL` values have a larger range than integers:

```
mysql> SELECT 9223372036854775807.0 + 1;
+-----+
| 9223372036854775807.0 + 1 |
+-----+
|          9223372036854775808.0 |
+-----+
```

Before MySQL 5.5.5, overflow handling during numeric expression evaluation depends on the types of the operands:

- Integer overflow results in silent wraparound.
- `DECIMAL` overflow results in a truncated result and a warning.
- Floating-point overflow produces a `NULL` result.

Subtraction between integer values, where one is of type `UNSIGNED`, produces an unsigned result by default. Prior to MySQL 5.5.5, if the result would otherwise have been negative, it becomes the maximum integer value:

```
mysql> SET sql_mode = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
|          18446744073709551615 |
+-----+
```

As of MySQL 5.5.5, if the result would otherwise have been negative, an error results:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

If the `NO_UNSIGNED_SUBTRACTION` [614] SQL mode is enabled, the result is negative:

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
|                      -1 |
+-----+
```

If the result of such an operation is used to update an `UNSIGNED` integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if `NO_UNSIGNED_SUBTRACTION` [614] is enabled. If strict SQL mode is enabled, an error occurs and the column remains unchanged.

11.3. Date and Time Types

The date and time types for representing temporal values are `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, and `YEAR`. Each temporal type has a range of valid values, as well as a “zero” value that may be used when you specify an invalid value that MySQL cannot represent. The `TIMESTAMP` type has special automatic updating behavior, described later. For temporal type storage requirements, see [Section 11.6, “Data Type Storage Requirements”](#).

Keep in mind these general considerations when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to

be assigned to or compared to a date or time type). For a description of the permitted formats for date and time types, see [Section 9.1.3, “Date and Time Literals”](#). It is expected that you supply valid values. Unpredictable results may occur if you use values in other formats.

- Although MySQL tries to interpret values in several formats, date parts must always be given in year-month-day order (for example, `'98-09-04'`), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, `'09-04-98'`, `'04-09-98'`).
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:
 - Year values in the range `70-99` are converted to `1970-1999`.
 - Year values in the range `00-69` are converted to `2000-2069`.

See also [Section 11.3.8, “Two-Digit Years in Dates”](#).

- Conversion of values from one temporal type to another occurs according to the rules in [Section 11.3.7, “Conversion Between Date and Time Types”](#).
- MySQL automatically converts a date or time value to a number if the value is used in a numeric context and vice versa.
- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise invalid for the type, it converts the value to the “zero” value for that type. The exception is that out-of-range `TIME` values are clipped to the appropriate endpoint of the `TIME` range.
- By setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See [Section 5.1.7, “Server SQL Modes”](#).) You can get MySQL to accept certain dates, such as `'2009-11-31'`, by enabling the `ALLOW_INVALID_DATES` [612] SQL mode. This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 1 to 12 and that the day is in the range from 1 to 31.
- MySQL permits you to store dates where the day or month and day are zero in a `DATE` or `DATETIME` column. This is extremely useful for applications that need to store birthdates for which you may not know the exact date. In this case, you simply store the date as `'2009-00-00'` or `'2009-01-00'`. If you store dates such as these, you should not expect to get correct results for functions such as `DATE_SUB()` [1139] or `DATE_ADD()` [1135] that require complete dates. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` [615] SQL mode.
- MySQL permits you to store a “zero” value of `'0000-00-00'` as a “dummy date.” This is in some cases more convenient than using `NULL` values, and uses less data and index space. To disallow `'0000-00-00'`, enable the `NO_ZERO_DATE` [615] SQL mode.
- “Zero” date or time values used through Connector/ODBC are converted automatically to `NULL` because ODBC cannot handle such values.

The following table shows the format of the “zero” value for each type. The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values `'0'` or `0`, which are easier to write. For temporal types that include a date part (`DATE`, `DATETIME`, and `TIMESTAMP`), use of these values produces warnings if the `NO_ZERO_DATE` [615] SQL mode is enabled.

Data Type	“Zero” Value
<code>DATE</code>	<code>'0000-00-00'</code>

Data Type	“Zero” Value
<code>TIME</code>	'00:00:00'
<code>DATETIME</code>	'0000-00-00 00:00:00'
<code>TIMESTAMP</code>	'0000-00-00 00:00:00'
<code>YEAR</code>	0000

11.3.1. The `DATE`, `DATETIME`, and `TIMESTAMP` Types

The `DATE`, `DATETIME`, and `TIMESTAMP` types are related. This section describes their characteristics, how they are similar, and how they differ. MySQL recognizes `DATE`, `DATETIME`, and `TIMESTAMP` values in several formats, described in [Section 9.1.3, “Date and Time Literals”](#). For the `DATE` and `DATETIME` range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The `DATE` type is used for values with a date part but no time part. MySQL retrieves and displays `DATE` values in 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.

The `DATETIME` type is used for values that contain both date and time parts. MySQL retrieves and displays `DATETIME` values in 'YYYY-MM-DD HH:MM:SS' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

The `TIMESTAMP` data type is used for values that contain both date and time parts. `TIMESTAMP` has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

MySQL converts `TIMESTAMP` values from the current time zone to UTC for storage, and back from UTC to the current time zone for retrieval. (This does not occur for other types such as `DATETIME`.) By default, the current time zone for each connection is the server's time. The time zone can be set on a per-connection basis. As long as the time zone setting remains constant, you get back the same value you store. If you store a `TIMESTAMP` value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the `time_zone` [567] system variable. For more information, see [Section 10.6, “MySQL Server Time Zone Support”](#).

The `TIMESTAMP` data type offers automatic initialization and updating to the current date and time. For more information, see [Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP`”](#).

A `DATETIME` or `TIMESTAMP` value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `DATETIME` or `TIMESTAMP` columns. For information about fractional seconds support in MySQL, see [Section 11.3.6, “Fractional Seconds in Time Values”](#).

Invalid `DATE`, `DATETIME`, or `TIMESTAMP` values are converted to the “zero” value of the appropriate type ('0000-00-00' or '0000-00-00 00:00:00').

Be aware of certain properties of date value interpretation in MySQL:

- MySQL permits a “relaxed” format for values specified as strings, in which any punctuation character may be used as the delimiter between date parts or time parts. In some cases, this syntax can be deceiving. For example, a value such as '10:11:12' might look like a time value because of the “:” delimiter, but is interpreted as the year '2010-11-12' if used in a date context. The value '10:45:15' is converted to '0000-00-00' because '45' is not a valid month.
- The server requires that month and day values be valid, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as '2004-04-31' are converted to '0000-00-00' and a warning is generated. With strict mode enabled, invalid dates generate an error.

To permit such dates, enable `ALLOW_INVALID_DATES` [612]. See [Section 5.1.7, “Server SQL Modes”](#), for more information.

- MySQL does not accept `TIMESTAMP` values that include a zero in the day or month column or values that are not a valid date. The sole exception to this rule is the special “zero” value `'0000-00-00 00:00:00'`.
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:
 - Year values in the range `00-69` are converted to `2000-2069`.
 - Year values in the range `70-99` are converted to `1970-1999`.

See also [Section 11.3.8, “Two-Digit Years in Dates”](#).



Note

The MySQL server can be run with the `MAXDB` [618] SQL mode enabled. In this case, `TIMESTAMP` is identical with `DATETIME`. If this mode is enabled at the time that a table is created, `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and there is no automatic initialization or updating to the current date and time. See [Section 5.1.7, “Server SQL Modes”](#).

11.3.2. The `TIME` Type

MySQL retrieves and displays `TIME` values in `'HH:MM:SS'` format (or `'HHH:MM:SS'` format for large hours values). `TIME` values may range from `'-838:59:59'` to `'838:59:59'`. The hours part may be so large because the `TIME` type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

MySQL recognizes `TIME` values in several formats, described in [Section 9.1.3, “Date and Time Literals”](#). Some of these formats can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `TIME` columns. For information about fractional seconds support in MySQL, see [Section 11.3.6, “Fractional Seconds in Time Values”](#).

Be careful about assigning abbreviated values to a `TIME` column. MySQL interprets abbreviated `TIME` values with colons as time of the day. That is, `'11:12'` means `'11:12:00'`, not `'00:11:12'`. MySQL interprets abbreviated values without colons using the assumption that the two rightmost digits represent seconds (that is, as elapsed time rather than as time of day). For example, you might think of `'1112'` and `1112` as meaning `'11:12:00'` (12 minutes after 11 o'clock), but MySQL interprets them as `'00:11:12'` (11 minutes, 12 seconds). Similarly, `'12'` and `12` are interpreted as `'00:00:12'`.

By default, values that lie outside the `TIME` range but are otherwise valid are clipped to the closest endpoint of the range. For example, `'-850:00:00'` and `'850:00:00'` are converted to `'-838:59:59'` and `'838:59:59'`. Invalid `TIME` values are converted to `'00:00:00'`. Note that because `'00:00:00'` is itself a valid `TIME` value, there is no way to tell, from a value of `'00:00:00'` stored in a table, whether the original value was specified as `'00:00:00'` or whether it was invalid.

For more restrictive treatment of invalid `TIME` values, enable strict SQL mode to cause errors to occur. See [Section 5.1.7, “Server SQL Modes”](#).

11.3.3. The `YEAR` Type

The `YEAR` type is a 1-byte type used to represent year values. It can be declared as `YEAR(4)` or `YEAR(2)` to specify a display width of four or two characters. The default is four characters if no width is given.



Note

The `YEAR(2)` data type has certain issues that you should consider before choosing to use it. As of MySQL 5.5.27, `YEAR(2)` is deprecated. For more information, see [Section 11.3.4, “YEAR\(2\) Limitations and Migrating to YEAR\(4\)”](#).

`YEAR(4)` and `YEAR(2)` differ in display format, but have the same range of values. For 4-digit format, MySQL displays `YEAR` values in `YYYY` format, with a range of `1901` to `2155`, or `0000`. For 2-digit format, MySQL displays only the last two (least significant) digits; for example, `70` (1970 or 2070) or `69` (2069).

You can specify input `YEAR` values in a variety of formats:

- As a 4-digit string in the range `'1901'` to `'2155'`.
- As a 4-digit number in the range `1901` to `2155`.
- As a 1- or 2-digit string in the range `'0'` to `'99'`. Values in the ranges `'0'` to `'69'` and `'70'` to `'99'` are converted to `YEAR` values in the ranges `2000` to `2069` and `1970` to `1999`.
- As a 1- or 2-digit number in the range `1` to `99`. Values in the ranges `1` to `69` and `70` to `99` are converted to `YEAR` values in the ranges `2001` to `2069` and `1970` to `1999`.

Inserting a numeric `0` has a different effect for `YEAR(2)` and `YEAR(4)`. For `YEAR(2)`, the result has a display value of `00` and an internal value of `2000`. For `YEAR(4)`, the result has a display value of `0000` and an internal value of `0000`. To specify zero for `YEAR(4)` and have it be interpreted as `2000`, specify it as a string `'0'` or `'00'`.

- As the result of a function that returns a value that is acceptable in a `YEAR` context, such as `NOW()` [1143].

Invalid `YEAR` values are converted to `0000`.

See also [Section 11.3.8, “Two-Digit Years in Dates”](#).

11.3.4. YEAR(2) Limitations and Migrating to YEAR(4)

Although the internal range of values for `YEAR(4)` and `YEAR(2)` is the same (`1901` to `2155`, and `0000`), the display width for `YEAR(2)` makes that type inherently ambiguous because displayed values indicate only the last two digits of the internal values. The result can be a loss of information under certain circumstances. For this reason, consider avoiding `YEAR(2)` throughout your applications and using `YEAR(4)` wherever you need a `YEAR` data type. This section describes problems that can occur when using `YEAR(2)` and provides information about migrating existing `YEAR(2)` columns to `YEAR(4)`. Note that migration will become necessary at some point because support for `YEAR` data types with display values other than 4, most notably `YEAR(2)`, is reduced as of MySQL 5.6.6 and will be removed entirely in a future release.

YEAR(2) Limitations

Issues with the `YEAR(2)` data type include ambiguity of displayed values, and possible loss of information when values are dumped and reloaded or converted to strings.

- Displayed `YEAR(2)` values can be ambiguous. It is possible for up to three `YEAR(2)` values that have different internal values to have the same displayed value, as the following example demonstrates:

```
mysql> CREATE TABLE t (y2 YEAR(2), y4 YEAR(4));
```

```

Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t (y2) VALUES(1912),(2012),(2112);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> UPDATE t SET y4 = y2;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3 Changed: 3 Warnings: 0

mysql> SELECT * FROM t;
+-----+-----+
| y2    | y4    |
+-----+-----+
| 12    | 1912  |
| 12    | 2012  |
| 12    | 2112  |
+-----+-----+
3 rows in set (0.00 sec)

```

- If you use `mysqldump` to dump the table created in the preceding item, the dump file represents all `y2` values using the same 2-digit representation (12). If you reload the table from the dump file, all resulting rows have internal value 2012 and display value 12, thus losing the distinctions among them.
- Conversion of a `YEAR(2)` or `YEAR(4)` data value to string form uses the display width of the `YEAR` type. Suppose that `YEAR(2)` and `YEAR(4)` columns both contain the value 1970. Assigning each column to a string results in a value of '70' or '1970', respectively. That is, loss of information occurs for conversion from `YEAR(2)` to string.
- Values outside the range from 1970 to 2069 are stored incorrectly when inserted into a `YEAR(2)` column in a `CSV` table. For example, inserting 2111 results in a display value of 11 but an internal value of 2011.

To avoid these problems, use `YEAR(4)` rather than `YEAR(2)`. Suggestions regarding migration strategies appear later in this section.

Migrating from `YEAR(2)` to `YEAR(4)`

Should you decide to convert `YEAR(2)` columns to `YEAR(4)`, use `ALTER TABLE`. Suppose that a table `t1` has this definition:

```
CREATE TABLE t1 (ycol YEAR(2) NOT NULL DEFAULT '70');
```

Modify the column using `ALTER TABLE` as follows. Remember to include any column attributes such as `NOT NULL` or `DEFAULT`:

```
ALTER TABLE t1 MODIFY ycol YEAR(4) NOT NULL DEFAULT '1970';
```

The `ALTER TABLE` statement converts the table without changing `YEAR(2)` values. If the server is a replication master, the `ALTER TABLE` statement replicates to slaves and makes the corresponding table change on each one.

One migration method should be avoided: Do not dump your data with `mysqldump` and reload the dump file after upgrading. This has the potential to change `YEAR(2)` values, as described previously.

A migration from `YEAR(2)` to `YEAR(4)` should also involve examining application code for the possibility of changed behavior under conditions such as these:

- Code that expects selecting a `YEAR` column to produce exactly two digits.
- Code that does not account for different handling for inserts of numeric 0: Inserting 0 into `YEAR(2)` or `YEAR(4)` results in an internal value of 2000 or 0000, respectively.

11.3.5. Automatic Initialization and Updating for `TIMESTAMP`



Note

In older versions of MySQL (prior to 4.1), the properties of the `TIMESTAMP` data type differed significantly in several ways from what is described in this section (see the *MySQL 3.23, 4.0, 4.1 Reference Manual* for details); these include syntax extensions which are deprecated in MySQL 5.1, and no longer supported in MySQL 5.5. This has implications for performing a dump and restore or replicating between MySQL Server versions. If you are using columns that are defined using the old `TIMESTAMP(N)` syntax, see [Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”](#), prior to upgrading to MySQL 5.5.

The `TIMESTAMP` data type offers automatic initialization and updating to the current date and time (that is, the current timestamp). You can choose whether to use these properties and which column should have them:

- One `TIMESTAMP` column in a table can have the current timestamp as the default value for initializing the column, as the auto-update value, or both. It is not possible to have the current timestamp be the default value for one column and the auto-update value for another column.
- If the column is auto-initialized, it is set to the current timestamp for inserted rows that specify no value for the column.
- If the column is auto-updated, it is automatically updated to the current timestamp when the value of any other column in the row is changed from its current value. The column remains unchanged if all other columns are set to their current values. To prevent the column from updating when other columns change, explicitly set it to its current value. To update the column even when other columns do not change, explicitly set it to the value it should have (for example, set it to `CURRENT_TIMESTAMP [1135]`).

In addition, you can initialize or update any `TIMESTAMP` column to the current date and time by assigning it a `NULL` value, unless it has been defined with the `NULL` attribute to permit `NULL` values.

To specify automatic properties, use the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses. The order of the clauses does not matter. If both are present in a column definition, either can occur first. Any of the synonyms for `CURRENT_TIMESTAMP [1135]` have the same meaning as `CURRENT_TIMESTAMP [1135]`. These are `CURRENT_TIMESTAMP () [1135]`, `NOW () [1143]`, `LOCALTIME [1142]`, `LOCALTIME () [1142]`, `LOCALTIMESTAMP [1142]`, and `LOCALTIMESTAMP () [1142]`.

Use of `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` is specific to `TIMESTAMP`. The `DEFAULT` clause also can be used to specify a constant (nonautomatic) default value; for example, `DEFAULT 0` or `DEFAULT '2000-01-01 00:00:00'`.



Note

The following examples that use `DEFAULT 0` do not work if the `NO_ZERO_DATE [615]` SQL mode is enabled because that mode causes “zero” date values (specified, for example, as `0 '0000-00-00 00:00:00'`) to be rejected. Be aware that the `TRADITIONAL [618]` SQL mode includes `NO_ZERO_DATE [615]`.

The following rules describe the possibilities for defining the first `TIMESTAMP` column in a table with the current timestamp for both the default and auto-update values, for one but not the other, or for neither:

- With both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP`, the column has the current timestamp for its default value and is automatically updated to the current timestamp.

```
CREATE TABLE t1 (
```

```
ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

- With neither `DEFAULT CURRENT_TIMESTAMP` nor `ON UPDATE CURRENT_TIMESTAMP`, it is the same as specifying both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP`.

```
CREATE TABLE t1 (
  ts TIMESTAMP
);
```

- With a `DEFAULT` clause but no `ON UPDATE CURRENT_TIMESTAMP` clause, the column has the given default value and is not automatically updated to the current timestamp.

The default depends on whether the `DEFAULT` clause specifies `CURRENT_TIMESTAMP` or a constant value. With `CURRENT_TIMESTAMP`, the default is the current timestamp.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

With a constant, the default is the given value. In this case, the column has no automatic properties at all.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT 0
);
```

- With an `ON UPDATE CURRENT_TIMESTAMP` clause and a constant `DEFAULT` clause, the column is automatically updated to the current timestamp and has the given constant default value.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
);
```

- With an `ON UPDATE CURRENT_TIMESTAMP` clause but no `DEFAULT` clause, the column is automatically updated to the current timestamp. The default is 0 unless the column is defined with the `NULL` attribute, in which case the default is `NULL`.

```
CREATE TABLE t1 (
  ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP      -- default 0
);
CREATE TABLE t2 (
  ts TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP -- default NULL
);
```

It need not be the first `TIMESTAMP` column in a table that is automatically initialized or updated to the current timestamp. However, to specify automatic initialization or updating for a different `TIMESTAMP` column, you must suppress the automatic properties for the first one. Then, for the other `TIMESTAMP` column, the rules for the `DEFAULT` and `ON UPDATE` clauses are the same as for the first `TIMESTAMP` column, except that if you omit both clauses, no automatic initialization or updating occurs.

To suppress automatic properties for the first `TIMESTAMP` column, do either of the following:

- Define the column with a `DEFAULT` clause that specifies a constant default value.
- Specify the `NULL` attribute. This also causes the column to permit `NULL` values, which means that you cannot assign the current timestamp by setting the column to `NULL`. Assigning `NULL` sets the column to `NULL`.

Consider these table definitions:

```
CREATE TABLE t1 (
```

```

ts1 TIMESTAMP DEFAULT 0,
ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t2 (
  ts1 TIMESTAMP NULL,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t3 (
  ts1 TIMESTAMP NULL DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP);

```

The tables have these properties:

- In each table definition, the first `TIMESTAMP` column has no automatic initialization or updating.
- The tables differ in how the `ts1` column handles `NULL` values. For `t1`, `ts1` is `NOT NULL` and assigning it a value of `NULL` sets it to the current timestamp. For `t2` and `t3`, `ts1` permits `NULL` and assigning it a value of `NULL` sets it to `NULL`.
- `t2` and `t3` differ in the default value for `ts1`. For `t2`, `ts1` is defined to permit `NULL`, so the default is also `NULL` in the absence of an explicit `DEFAULT` clause. For `t3`, `ts1` permits `NULL` but has an explicit default of 0.

`TIMESTAMP` Initialization and the `NULL` Attribute

By default, `TIMESTAMP` columns are `NOT NULL`, cannot contain `NULL` values, and assigning `NULL` assigns the current timestamp. To permit a `TIMESTAMP` column to contain `NULL`, explicitly declare it with the `NULL` attribute. In this case, the default value also becomes `NULL` unless overridden with a `DEFAULT` clause that specifies a different default value. `DEFAULT NULL` can be used to explicitly specify `NULL` as the default value. (For a `TIMESTAMP` column not declared with the `NULL` attribute, `DEFAULT NULL` is invalid.) If a `TIMESTAMP` column permits `NULL` values, assigning `NULL` sets it to `NULL`, not to the current timestamp.

The following table contains several `TIMESTAMP` columns that permit `NULL` values:

```

CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);

```

A `TIMESTAMP` column that permits `NULL` values does *not* take on the current timestamp at insert time except under one of the following conditions:

- Its default value is defined as `CURRENT_TIMESTAMP` [1135] and no value is specified for the column
- `CURRENT_TIMESTAMP` [1135] or any of its synonyms such as `NOW()` [1143] is explicitly inserted into the column

In other words, a `TIMESTAMP` column defined to permit `NULL` values auto-initializes only if its definition includes `DEFAULT CURRENT_TIMESTAMP`:

```

CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);

```

If the `TIMESTAMP` column permits `NULL` values but its definition does not include `DEFAULT CURRENT_TIMESTAMP`, you must explicitly insert a value corresponding to the current date and time.

Suppose that tables `t1` and `t2` have these definitions:

```

CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT NULL);

```

To set the `TIMESTAMP` column in either table to the current timestamp at insert time, explicitly assign it that value. For example:

```
INSERT INTO t1 VALUES (NOW());
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

11.3.6. Fractional Seconds in Time Values

A trailing fractional seconds part is permissible for temporal values in contexts such as literal values, and in the arguments to or return values from some temporal functions. Example:

```
mysql> SELECT MICROSECOND('2010-12-10 14:12:09.019473');
+-----+
| MICROSECOND('2010-12-10 14:12:09.019473') |
+-----+
|                                     19473 |
+-----+
```

However, when MySQL stores a value into a column of any temporal data type, it discards any fractional part and does not store it.

11.3.7. Conversion Between Date and Time Types

To some extent, you can convert a value from one temporal type to another. However, there may be some alteration of the value or loss of information. In all cases, conversion between temporal types is subject to the range of valid values for the resulting type. For example, although `DATE`, `DATETIME`, and `TIMESTAMP` values all can be specified using the same set of formats, the types do not all have the same range of values. `TIMESTAMP` values cannot be earlier than 1970 UTC or later than '2038-01-19 03:14:07' UTC. This means that a date such as '1968-01-01', while valid as a `DATE` or `DATETIME` value, is not valid as a `TIMESTAMP` value and is converted to 0.

Conversion of `DATE` values:

- Conversion to a `DATETIME` or `TIMESTAMP` value adds a time part of '00:00:00' because the `DATE` value contains no time information.
- Conversion to a `TIME` value is not useful; the result is '00:00:00'.

Conversion of `DATETIME` and `TIMESTAMP` values:

- Conversion to a `DATE` value discards the time part because the `DATE` type contains no time information.
- Conversion to a `TIME` value discards the date part because the `TIME` type contains no date information.

Conversion of `TIME` values:

MySQL converts a time value to a date or date-and-time value by parsing the string value of the time as a date or date-and-time. This is unlikely to be useful. For example, '23:12:31' interpreted as a date becomes '2032-12-31'. Time values not valid as dates become '0000-00-00' or `NULL`.

Explicit conversion can be used to override implicit conversion. For example, in comparison of `DATE` and `DATETIME` values, the `DATE` value is coerced to the `DATETIME` type by adding a time part of '00:00:00'. To perform the comparison by ignoring the time part of the `DATETIME` value instead, use the `CAST()` [1168] function in the following way:

```
date_col = CAST(datetime_col AS DATE)
```

Conversion of `TIME` or `DATETIME` values to numeric form (for example, by adding +0) results in a double-precision value with a microseconds part of .000000:

```
mysql> SELECT CURTIME(), CURTIME()+0;
+-----+-----+
| CURTIME() | CURTIME()+0 |
+-----+-----+
| 10:41:36 | 104136.000000 |
+-----+-----+
mysql> SELECT NOW(), NOW()+0;
+-----+-----+
| NOW() | NOW()+0 |
+-----+-----+
| 2007-11-30 10:41:47 | 20071130104147.000000 |
+-----+-----+
```

11.3.8. Two-Digit Years in Dates

Date values with two-digit years are ambiguous because the century is unknown. Such values must be interpreted into four-digit form because MySQL stores years internally using four digits.

For `DATETIME`, `DATE`, and `TIMESTAMP` types, MySQL interprets dates specified with ambiguous year values using these rules:

- Year values in the range `00–69` are converted to `2000–2069`.
- Year values in the range `70–99` are converted to `1970–1999`.

For `YEAR`, the rules are the same, with this exception: A numeric `00` inserted into `YEAR(4)` results in `0000` rather than `2000`. To specify zero for `YEAR(4)` and have it be interpreted as `2000`, specify it as a string `'0'` or `'00'`.

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the values you require, you must provide unambiguous input containing four-digit year values.

`ORDER BY` properly sorts `YEAR` values that have two-digit years.

Some functions like `MIN()` [1206] and `MAX()` [1205] convert a `YEAR` to a number. This means that a value with a two-digit year does not work properly with these functions. The fix in this case is to convert the `YEAR` to four-digit year format.

11.4. String Types

The string types are `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `BLOB`, `TEXT`, `ENUM`, and `SET`. This section describes how these types work and how to use them in your queries. For string type storage requirements, see [Section 11.6, “Data Type Storage Requirements”](#).

11.4.1. The `CHAR` and `VARCHAR` Types

The `CHAR` and `VARCHAR` types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained.

The `CHAR` and `VARCHAR` types are declared with a length that indicates the maximum number of characters you want to store. For example, `CHAR(30)` can hold up to 30 characters.

The length of a `CHAR` column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When `CHAR` values are stored, they are right-padded with spaces to the specified length. When `CHAR` values are retrieved, trailing spaces are removed unless the `PAD_CHAR_TO_FULL_LENGTH` [616] SQL mode is enabled.

Values in `VARCHAR` columns are variable-length strings. The length can be specified as a value from 0 to 65,535. The effective maximum length of a `VARCHAR` is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. See [Section E.10.4, “Table Column-Count and Row-Size Limits”](#).

In contrast to `CHAR`, `VARCHAR` values are stored as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

If strict SQL mode is not enabled and you assign a value to a `CHAR` or `VARCHAR` column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.1.7, “Server SQL Modes”](#).

For `VARCHAR` columns, trailing spaces in excess of the column length are truncated prior to insertion and a warning is generated, regardless of the SQL mode in use. For `CHAR` columns, truncation of excess trailing spaces from inserted values is performed silently regardless of the SQL mode.

`VARCHAR` values are not padded when they are stored. Trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL.

The following table illustrates the differences between `CHAR` and `VARCHAR` by showing the result of storing various string values into `CHAR(4)` and `VARCHAR(4)` columns (assuming that the column uses a single-byte character set such as `latin1`).

Value	<code>CHAR(4)</code>	Storage Required	<code>VARCHAR(4)</code>	Storage Required
' '	' '	4 bytes	' '	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

The values shown as stored in the last row of the table apply *only when not using strict mode*; if MySQL is running in strict mode, values that exceed the column length are *not stored*, and an error results.

If a given value is stored into the `CHAR(4)` and `VARCHAR(4)` columns, the values retrieved from the columns are not always the same because trailing spaces are removed from `CHAR` columns upon retrieval. The following example illustrates this difference:

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO vc VALUES ('ab  ', 'ab  ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('(v, )'), CONCAT('(c, )') FROM vc;
+-----+-----+
| CONCAT('(v, )') | CONCAT('(c, )') |
+-----+-----+
| (ab  )          | (ab)            |
+-----+-----+
1 row in set (0.06 sec)
```

Values in `CHAR` and `VARCHAR` columns are sorted and compared according to the character set collation assigned to the column.

All MySQL collations are of type `PADSPACE`. This means that all `CHAR` and `VARCHAR` values in MySQL are compared without regard to any trailing spaces. For example:

```
mysql> CREATE TABLE names (myname CHAR(10), yourname VARCHAR(10));
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO names VALUES ('Monty ', 'Monty ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT myname = 'Monty ', yourname = 'Monty ' FROM names;
+-----+-----+
| myname = 'Monty ' | yourname = 'Monty ' |
+-----+-----+
| 1 | 1 |
+-----+-----+
1 row in set (0.00 sec)
```

This is true for all MySQL versions, and is not affected by the server SQL mode.



Note

For more information about MySQL character sets and collations, see [Section 10.1, “Character Set Support”](#).

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error.

11.4.2. The `BINARY` and `VARBINARY` Types

The `BINARY` and `VARBINARY` types are similar to `CHAR` and `VARCHAR`, except that they contain binary strings rather than nonbinary strings. That is, they contain byte strings rather than character strings. This means that they have no character set, and sorting and comparison are based on the numeric values of the bytes in the values.

The permissible maximum length is the same for `BINARY` and `VARBINARY` as it is for `CHAR` and `VARCHAR`, except that the length for `BINARY` and `VARBINARY` is a length in bytes rather than in characters.

The `BINARY` and `VARBINARY` data types are distinct from the `CHAR BINARY` and `VARCHAR BINARY` data types. For the latter types, the `BINARY` attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary collation for the column character set to be used, and the column itself contains nonbinary character strings rather than binary byte strings. For example, `CHAR(5) BINARY` is treated as `CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin`, assuming that the default character set is `latin1`. This differs from `BINARY(5)`, which stores 5-bytes binary strings that have no character set or collation. For information about differences between nonbinary string binary collations and binary strings, see [Section 10.1.7.6, “The `_bin` and `binary` Collations”](#).

If strict SQL mode is not enabled and you assign a value to a `BINARY` or `VARBINARY` column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For cases of truncation, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.1.7, “Server SQL Modes”](#).

When `BINARY` values are stored, they are right-padded with the pad value to the specified length. The pad value is `0x00` (the zero byte). Values are right-padded with `0x00` on insert, and no trailing bytes are removed on select. All bytes are significant in comparisons, including `ORDER BY` and `DISTINCT` operations. `0x00` bytes and spaces are different in comparisons, with `0x00 < space`.

Example: For a `BINARY(3)` column, 'a ' becomes 'a \0' when inserted. 'a\0' becomes 'a\0\0' when inserted. Both inserted values remain unchanged when selected.

For `VARBINARY`, there is no padding on insert and no bytes are stripped on select. All bytes are significant in comparisons, including `ORDER BY` and `DISTINCT` operations. `0x00` bytes and spaces are different in comparisons, with `0x00 < space`.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad bytes will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a\0' causes a duplicate-key error.

You should consider the preceding padding and stripping characteristics carefully if you plan to use the `BINARY` data type for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how `0x00`-padding of `BINARY` values affects column value comparisons:

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;
+-----+-----+-----+
| HEX(c) | c = 'a' | c = 'a\0\0' |
+-----+-----+-----+
| 610000 |      0 |           1 |
+-----+-----+-----+
1 row in set (0.09 sec)
```

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use `VARBINARY` or one of the `BLOB` data types instead.

11.4.3. The BLOB and TEXT Types

A `BLOB` is a binary large object that can hold a variable amount of data. The four `BLOB` types are `TINYBLOB`, `BLOB`, `MEDIUMBLOB`, and `LONGBLOB`. These differ only in the maximum length of the values they can hold. The four `TEXT` types are `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, and `LONGTEXT`. These correspond to the four `BLOB` types and have the same maximum lengths and storage requirements. See [Section 11.6, “Data Type Storage Requirements”](#).

`BLOB` values are treated as binary strings (byte strings). They have no character set, and sorting and comparison are based on the numeric values of the bytes in column values. `TEXT` values are treated as nonbinary strings (character strings). They have a character set, and values are sorted and compared based on the collation of the character set.

If strict SQL mode is not enabled and you assign a value to a `BLOB` or `TEXT` column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.1.7, “Server SQL Modes”](#).

Truncation of excess trailing spaces from values to be inserted into `TEXT` columns always generates a warning, regardless of the SQL mode.

For `TEXT` and `BLOB` columns, there is no padding on insert and no bytes are stripped on select.

If a `TEXT` column is indexed, index entry comparisons are space-padded at the end. This means that, if the index requires unique values, duplicate-key errors will occur for values that differ only in the number of trailing spaces. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error. This is not true for `BLOB` columns.

In most respects, you can regard a `BLOB` column as a `VARBINARY` column that can be as large as you like. Similarly, you can regard a `TEXT` column as a `VARCHAR` column. `BLOB` and `TEXT` differ from `VARBINARY` and `VARCHAR` in the following ways:

- For indexes on `BLOB` and `TEXT` columns, you must specify an index prefix length. For `CHAR` and `VARCHAR`, a prefix length is optional. See [Section 8.3.4, “Column Indexes”](#).
- `BLOB` and `TEXT` columns cannot have `DEFAULT` values.

If you use the `BINARY` attribute with a `TEXT` data type, the column is assigned the binary collation of the column character set.

`LONG` and `LONG VARCHAR` map to the `MEDIUMTEXT` data type. This is a compatibility feature.

MySQL Connector/ODBC defines `BLOB` values as `LONGVARBINARY` and `TEXT` values as `LONGVARCHAR`.

Because `BLOB` and `TEXT` values can be extremely long, you might encounter some constraints in using them:

- Only the first `max_sort_length [524]` bytes of the column are used when sorting. The default value of `max_sort_length [524]` is 1024. You can make more bytes significant in sorting or grouping by increasing the value of `max_sort_length [524]` at server startup or runtime. Any client can change the value of its session `max_sort_length [524]` variable:

```
mysql> SET max_sort_length = 2000;
mysql> SELECT id, comment FROM t
-> ORDER BY comment;
```

- Instances of `BLOB` or `TEXT` columns in the result of a query that is processed using a temporary table causes the server to use a table on disk rather than in memory because the `MEMORY` storage engine does not support those data types (see [Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#)). Use of disk incurs a performance penalty, so include `BLOB` or `TEXT` columns in the query result only if they are really needed. For example, avoid using `SELECT *`, which selects all columns.
- The maximum size of a `BLOB` or `TEXT` object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by changing the value of the `max_allowed_packet [519]` variable, but you must do so for both the server and your client program. For example, both `mysql` and `mysqldump` enable you to change the client-side `max_allowed_packet [519]` value. See [Section 8.11.2, “Tuning Server Parameters”](#), [Section 4.5.1, “mysql — The MySQL Command-Line Tool”](#), and [Section 4.5.4, “mysqldump — A Database Backup Program”](#). You may also want to compare the packet sizes and the size of the data objects you are storing with the storage requirements, see [Section 11.6, “Data Type Storage Requirements”](#)

Each `BLOB` or `TEXT` value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

In some cases, it may be desirable to store binary data such as media files in `BLOB` or `TEXT` columns. You may find MySQL's string handling functions useful for working with such data. See [Section 12.5, “String Functions”](#). For security and other reasons, it is usually preferable to do so using application code rather than giving application users the `FILE [696]` privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (<http://forums.mysql.com/>).

11.4.4. The ENUM Type

An `ENUM` is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time. It has these advantages:

- Compact data storage in situations where a column has a limited set of possible values. The strings you specify as input values are automatically encoded as numbers. See [Section 11.6, “Data Type Storage Requirements”](#) for the storage requirements for ENUM types.
- Readable queries and output. The numbers are translated back to the corresponding strings in query results.

and these potential issues to consider:

- If you make enumeration values that look like numbers, it is easy to mix up the literal values with their internal index numbers, as explained in [Section 11.4.4, “Limitations”](#).
- Using ENUM columns in ORDER BY clauses requires extra care, as explained in [Section 11.4.4, “Sorting”](#).

Creating and Using ENUM Columns

An enumeration value must be a quoted string literal. For example, you can create a table with an ENUM column like this:

```
CREATE TABLE shirts (
  name VARCHAR(40),
  size ENUM('x-small', 'small', 'medium', 'large', 'x-large')
);
INSERT INTO shirts (name, size) VALUES ('dress shirt','large'), ('t-shirt','medium'), ('polo shirt','small');
SELECT name, size FROM shirts WHERE size = 'medium';
+-----+-----+
| name   | size  |
+-----+-----+
| t-shirt | medium |
+-----+-----+
UPDATE shirts SET size = 'small' WHERE size = 'large';
COMMIT;
```

Inserting 1 million rows into this table with a value of 'medium' would require 1 million bytes of storage, as opposed to 6 million bytes if you stored the actual string 'medium' in a VARCHAR column.

Index Values for Enumeration Literals

Each enumeration value has an index:

- The elements listed in the column specification are assigned index numbers, beginning with 1.
- The index value of the empty string error value is 0. This means that you can use the following SELECT statement to find rows into which invalid ENUM values were assigned:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- The index of the NULL value is NULL.
- The term “index” here refers to a position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as ENUM('Mercury', 'Venus', 'Earth') can have any of the values shown here. The index of each value is also shown.

Value	Index
NULL	NULL
' '	0

Value	Index
'Mercury'	1
'Venus'	2
'Earth'	3

An enumeration can have a maximum of 65,535 elements.

If you retrieve an `ENUM` value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an `ENUM` column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

Functions such as `SUM()` [1206] or `AVG()` [1203] that expect a numeric argument cast the argument to a number if necessary. For `ENUM` values, the index number is used in the calculation.

Handling of Enumeration Literals

Trailing spaces are automatically deleted from `ENUM` member values in the table definition when a table is created.

When retrieved, values stored into an `ENUM` column are displayed using the lettercase that was used in the column definition. Note that `ENUM` columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

If you store a number into an `ENUM` column, the number is treated as the index into the possible values, and the value stored is the enumeration member with that index. (However, this does *not* work with `LOAD DATA`, which treats all input as strings.) If the numeric value is quoted, it is still interpreted as an index if there is no matching string in the list of enumeration values. For these reasons, it is not advisable to define an `ENUM` column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of '0', '1', and '2', but numeric index values of 1, 2, and 3:

```
numbers ENUM('0','1','2')
```

If you store 2, it is interpreted as an index value, and becomes '1' (the value with index 2). If you store '2', it matches an enumeration value, so it is stored as '2'. If you store '3', it does not match any enumeration value, so it is treated as an index and becomes '2' (the value with index 3).

```
mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');
mysql> SELECT * FROM t;
```

```
+-----+
| numbers |
+-----+
| 1       |
| 2       |
| 2       |
+-----+
```

To determine all possible values for an `ENUM` column, use `SHOW COLUMNS FROM tbl_name LIKE 'enum_col'` and parse the `ENUM` definition in the `Type` column of the output.

In the C API, `ENUM` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see [Section 22.8.1, "C API Data Structures"](#).

Empty or NULL Enumeration Values

An enumeration value can also be the empty string (' ') or `NULL` under certain circumstances:

- If you insert an invalid value into an `ENUM` (that is, a string not present in the list of permitted values), the empty string is inserted instead as a special error value. This string can be distinguished from a “normal” empty string by the fact that this string has the numeric value 0. See [Section 11.4.4, “Index Values for Enumeration Literals”](#) for details about the numeric indexes for the enumeration values.

If strict SQL mode is enabled, attempts to insert invalid `ENUM` values result in an error.

- If an `ENUM` column is declared to permit `NULL`, the `NULL` value is a valid value for the column, and the default value is `NULL`. If an `ENUM` column is declared `NOT NULL`, its default value is the first element of the list of permitted values.

Sorting

`ENUM` values are sorted based on their index numbers, which depend on the order in which the enumeration members were listed in the column specification. For example, `'b'` sorts before `'a'` for `ENUM('b', 'a')`. The empty string sorts before nonempty strings, and `NULL` values sort before all other enumeration values.

To prevent unexpected results when using the `ORDER BY` clause on an `ENUM` column, use one of these techniques:

- Specify the `ENUM` list in alphabetic order.
- Make sure that the column is sorted lexically rather than by index number by coding `ORDER BY CAST(col AS CHAR)` or `ORDER BY CONCAT(col)`.

Limitations

An enumeration value cannot be an expression, even one that evaluates to a string value.

For example, this `CREATE TABLE` statement does *not* work because the `CONCAT` function cannot be used to construct an enumeration value:

```
CREATE TABLE sizes (
  size ENUM('small', CONCAT('med','ium'), 'large')
);
```

You also cannot employ a user variable as an enumeration value. This pair of statements do *not* work:

```
SET @mysize = 'medium';

CREATE TABLE sizes (
  size ENUM('small', @mysize, 'large')
);
```

We strongly recommend that you do *not* use numbers as enumeration values, because it does not save on storage over the appropriate `TINYINT` or `SMALLINT` type, and it is easy to mix up the strings and the underlying number values (which might not be the same) if you quote the `ENUM` values incorrectly. If you do use a number as an enumeration value, always enclose it in quotation marks. If the quotation marks are omitted, the number is regarded as an index. See [Section 11.4.4, “Handling of Enumeration Literals”](#) to see how even a quoted number could be mistakenly used as a numeric index value.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

11.4.5. The SET Type

A `SET` is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. `SET` column values that consist of multiple set

members are specified with members separated by commas (“,”). A consequence of this is that SET member values should not themselves contain commas.

For example, a column specified as SET('one', 'two') NOT NULL can have any of these values:

```
' '
'one'
'two'
'one,two'
```

A SET can have a maximum of 64 different members.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

Trailing spaces are automatically deleted from SET member values in the table definition when a table is created.

When retrieved, values stored in a SET column are displayed using the lettercase that was used in the column definition. Note that SET columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

MySQL stores SET values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a SET value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a SET column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as SET('a', 'b', 'c', 'd'), the members have the following decimal and binary values.

SET Member	Decimal Value	Binary Value
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. For example, suppose that a column is specified as SET('a', 'b', 'c', 'd'):

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

Then all these values appear as 'a,d' when retrieved:

```
mysql> SELECT col FROM myset;
+-----+
| col   |
+-----+
| a,d   |
+-----+
5 rows in set (0.04 sec)
```

If you set a SET column to an unsupported value, the value is ignored and a warning is issued:

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+-----+-----+-----+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+-----+
| col   |
+-----+
| a,d   |
+-----+
6 rows in set (0.01 sec)
```

If strict SQL mode is enabled, attempts to insert invalid SET values result in an error.

SET values are sorted numerically. NULL values sort before non-NULL SET values.

Functions such as SUM() [1206] or AVG() [1203] that expect a numeric argument cast the argument to a number if necessary. For SET values, the cast operation causes the numeric value to be used.

Normally, you search for SET values using the FIND_IN_SET() [1103] function or the LIKE [1111] operator:

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

The first statement finds rows where *set_col* contains the *value* set member. The second is similar, but not the same: It finds rows where *set_col* contains *value* anywhere, even as a substring of another set member.

The following statements also are permitted:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to 'val1,val2'

returns different results than comparing values to `'val2,val1'`. You should specify the values in the same order they are listed in the column definition.

To determine all possible values for a `SET` column, use `SHOW COLUMNS FROM tbl_name LIKE set_col` and parse the `SET` definition in the `Type` column of the output.

In the C API, `SET` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see [Section 22.8.1, “C API Data Structures”](#).

11.5. Data Type Default Values

The `DEFAULT value` clause in a data type specification indicates a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` [1143] or `CURRENT_DATE` [1135]. The exception is that you can specify `CURRENT_TIMESTAMP` [1135] as the default for a `TIMESTAMP` column. See [Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP`”](#).

`BLOB` and `TEXT` columns cannot be assigned a default value.

If a column definition includes no explicit `DEFAULT` value, MySQL determines the default value as follows:

If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause.

If the column cannot take `NULL` as the value, MySQL defines the column with no explicit `DEFAULT` clause. Exception: If the column is defined as part of a `PRIMARY KEY` but not explicitly as `NOT NULL`, MySQL creates it as a `NOT NULL` column (because `PRIMARY KEY` columns must be `NOT NULL`), but also assigns it a `DEFAULT` clause using the implicit default value. To prevent this, include an explicit `NOT NULL` in the definition of any `PRIMARY KEY` column.

For data entry into a `NOT NULL` column that has no explicit `DEFAULT` clause, if an `INSERT` or `REPLACE` statement includes no value for the column, or an `UPDATE` statement sets the column to `NULL`, MySQL handles the column according to the SQL mode in effect at the time:

- If strict SQL mode is enabled, an error occurs for transactional tables and the statement is rolled back. For nontransactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows will have been inserted.
- If strict mode is not enabled, MySQL sets the column to the implicit default value for the column data type.

Suppose that a table `t` is defined as follows:

```
CREATE TABLE t (i INT NOT NULL);
```

In this case, `i` has no explicit default, so in strict mode each of the following statements produce an error and no row is inserted. When not using strict mode, only the third statement produces an error; the implicit default is inserted for the first two statements, but the third fails because `DEFAULT(i)` [1198] cannot produce a value:

```
INSERT INTO t VALUES();
INSERT INTO t VALUES(DEFAULT);
INSERT INTO t VALUES(DEFAULT(i));
```

See [Section 5.1.7, “Server SQL Modes”](#).

For a given table, you can use the `SHOW CREATE TABLE` statement to see which columns have an explicit `DEFAULT` clause.

Implicit defaults are defined as follows:

- For numeric types, the default is `0`, with the exception that for integer or floating-point types declared with the `AUTO_INCREMENT` attribute, the default is the next value in the sequence.
- For date and time types other than `TIMESTAMP`, the default is the appropriate “zero” value for the type. For the first `TIMESTAMP` column in a table, the default value is the current date and time. See [Section 11.3, “Date and Time Types”](#).
- For string types other than `ENUM`, the default value is the empty string. For `ENUM`, the default is the first enumeration value.

`SERIAL DEFAULT VALUE` in the definition of an integer column is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

11.6. Data Type Storage Requirements

The storage requirements for data vary, according to the storage engine being used for the table in question. Different storage engines use different methods for recording the raw data and different data types. In addition, some engines may compress the information in a given row, either on a column or entire row basis, making calculation of the storage requirements for a given table or column structure.

However, all storage engines must communicate and exchange information on a given row within a table using the same structure, and this information is consistent, irrespective of the storage engine used to write the information to disk.

This sections includes some guidelines and information for the storage requirements for each data type supported by MySQL, including details for the internal format and the sizes used by storage engines that used a fixed size representation for different types. Information is listed by category or storage engine.

The internal representation of a table has a maximum row size of 65,535 bytes, even if the storage engine is capable of supporting larger rows. This figure excludes `BLOB` or `TEXT` columns, which contribute only 9 to 12 bytes toward this size. For `BLOB` and `TEXT` data, the information is stored internally in a different area of memory than the row buffer. Different storage engines handle the allocation and storage of this data in different ways, according to the method they use for handling the corresponding types. For more information, see [Chapter 14, Storage Engines](#), and [Section E.10.4, “Table Column-Count and Row-Size Limits”](#).



Important

For tables using the `NDBCLUSTER` storage engine, there is the factor of *4-byte alignment* to be taken into account when calculating storage requirements. This means that all `NDB` data storage is done in multiples of 4 bytes. Thus, a column value that would take 15 bytes in a table using a storage engine other than `NDB` requires 16 bytes in an `NDB` table. This requirement applies in addition to any other considerations that are discussed in this section. For example, in `NDBCLUSTER` tables, the `TINYINT`, `SMALLINT`, `MEDIUMINT`, and `INTEGER (INT)` column types each require 4 bytes storage per record due to the alignment factor.

An exception to this rule is the `BIT` type, which is *not* 4-byte aligned. In MySQL Cluster tables, a `BIT(M)` column takes *M* bits of storage space. However, if a table definition contains 1 or more `BIT` columns (up to 32 `BIT` columns), then `NDBCLUSTER` reserves 4 bytes (32 bits) per row for these. If a table definition contains more than 32 `BIT` columns (up to 64 such columns), then `NDBCLUSTER` reserves 8 bytes (that is, 64 bits) per row.

In addition, while a `NULL` itself does not require any storage space, `NDBCLUSTER` reserves 4 bytes per row if the table definition contains any columns defined as `NULL`, up to 32 `NULL` columns. (If a MySQL Cluster table is defined with more than 32 `NULL` columns up to 64 `NULL` columns, then 8 bytes per row is reserved.)

When calculating storage requirements for MySQL Cluster tables, you must also remember that every table using the `NDBCLUSTER` storage engine requires a primary key; if no primary key is defined by the user, then a “hidden” primary key will be created by `NDB`. This hidden primary key consumes 31-35 bytes per table record.

You may find the `ndb_size.pl` utility to be useful for estimating `NDB` storage requirements. This Perl script connects to a current MySQL (non-Cluster) database and creates a report on how much space that database would require if it used the `NDBCLUSTER` storage engine. See [Section 17.4.22, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”](#), for more information.

Storage Requirements for Numeric Types

Data Type	Storage Required
<code>TINYINT</code>	1 byte
<code>SMALLINT</code>	2 bytes
<code>MEDIUMINT</code>	3 bytes
<code>INT</code> , <code>INTEGER</code>	4 bytes
<code>BIGINT</code>	8 bytes
<code>FLOAT(p)</code>	4 bytes if $0 \leq p \leq 24$, 8 bytes if $25 \leq p \leq 53$
<code>FLOAT</code>	4 bytes
<code>DOUBLE [PRECISION]</code> , <code>REAL</code>	8 bytes
<code>DECIMAL(M,D)</code> , <code>NUMERIC(M,D)</code>	Varies; see following discussion
<code>BIT(M)</code>	approximately $(M+7)/8$ bytes

Values for `DECIMAL` (and `NUMERIC`) columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4

Storage Requirements for Date and Time Types

Data Type	Storage Required
DATE	3 bytes
TIME	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
YEAR	1 byte

For details about internal representation of temporal values, see [MySQL Internals: Important Algorithms and Structures](#).

Storage Requirements for String Types

In the following table, *M* represents the declared column length in characters for nonbinary string types and bytes for binary string types. *L* represents the actual length in bytes of a given string value.

Data Type	Storage Required
CHAR(<i>M</i>)	$M \times w$ bytes, $0 \leq M \leq 255$, where <i>w</i> is the number of bytes required for the maximum-length character in the character set
BINARY(<i>M</i>)	<i>M</i> bytes, $0 \leq M \leq 255$
VARCHAR(<i>M</i>), VARBINARY(<i>M</i>)	<i>L</i> + 1 bytes if column values require 0 – 255 bytes, <i>L</i> + 2 bytes if values may require more than 255 bytes
TINYBLOB, TINYTEXT	<i>L</i> + 1 bytes, where $L < 2^8$
BLOB, TEXT	<i>L</i> + 2 bytes, where $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	<i>L</i> + 3 bytes, where $L < 2^{24}$
LOB, LONGTEXT	<i>L</i> + 4 bytes, where $L < 2^{32}$
ENUM('value1', 'value2', ...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET('value1', 'value2', ...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Variable-length string types are stored using a length prefix plus data. The length prefix requires from one to four bytes depending on the data type, and the value of the prefix is *L* (the byte length of the string). For example, storage for a `MEDIUMTEXT` value requires *L* bytes to store the value plus three bytes to store the length of the value.

To calculate the number of bytes used to store a particular `CHAR`, `VARCHAR`, or `TEXT` column value, you must take into account the character set used for that column and whether the value contains multi-byte characters. In particular, when using the `utf8` (or `utf8mb4`) Unicode character set, you must keep in mind that not all characters use the same number of bytes and can require up to three (four) bytes per character. For a breakdown of the storage used for different categories of `utf8` or `utf8mb4` characters, see [Section 10.1.10, “Unicode Support”](#).

`VARCHAR`, `VARBINARY`, and the `BLOB` and `TEXT` types are variable-length types. For each, the storage requirements depend on these factors:

- The actual length of the column value
- The column's maximum possible length

- The character set used for the column, because some character sets contain multi-byte characters

For example, a `VARCHAR(255)` column can hold a string with a maximum length of 255 characters. Assuming that the column uses the `latin1` character set (one byte per character), the actual storage required is the length of the string (L), plus one byte to record the length of the string. For the string `'abcd'`, L is 4 and the storage requirement is five bytes. If the same column is instead declared to use the `ucs2` double-byte character set, the storage requirement is 10 bytes: The length of `'abcd'` is eight bytes and the column requires two bytes to store lengths because the maximum length is greater than 255 (up to 510 bytes).

The effective maximum number of *bytes* that can be stored in a `VARCHAR` or `VARBINARY` column is subject to the maximum row size of 65,535 bytes, which is shared among all columns. For a `VARCHAR` column that stores multi-byte characters, the effective maximum number of *characters* is less. For example, `utf8` characters can require up to three bytes per character, so a `VARCHAR` column that uses the `utf8` character set can be declared to be a maximum of 21,844 characters. See [Section E.10.4, “Table Column-Count and Row-Size Limits”](#).

The `NDBCLUSTER` storage engine supports variable-width columns. This means that a `VARCHAR` column in a MySQL Cluster table requires the same amount of storage as would any other storage engine, with the exception that such values are 4-byte aligned. Thus, the string `'abcd'` stored in a `VARCHAR(50)` column using the `latin1` character set requires 8 bytes (rather than 6 bytes for the same column value in a `MyISAM` table).

`TEXT` and `BLOB` columns are implemented differently in the NDB Cluster storage engine, wherein each row in a `TEXT` column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which is stored in a hidden table. The rows in this second table are always 2,000 bytes long. This means that the size of a `TEXT` column is 256 if $size \leq 256$ (where $size$ represents the size of the row); otherwise, the size is $256 + size + (2000 - (size - 256) \% 2000)$.

The size of an `ENUM` object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See [Section 11.4.4, “The ENUM Type”](#).

The size of a `SET` object is determined by the number of different set members. If the set size is N , the object occupies $(N+7)/8$ bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A `SET` can have a maximum of 64 members. See [Section 11.4.5, “The SET Type”](#).

11.7. Choosing the Right Type for a Column

For optimum storage, you should try to use the most precise type in all cases. For example, if an integer column is used for values in the range from 1 to 99999, `MEDIUMINT UNSIGNED` is the best type. Of the types that represent all the required values, this type uses the least amount of storage.

All basic calculations (+, -, *, and /) with `DECIMAL` columns are done with precision of 65 decimal (base 10) digits. See [Section 11.1.1, “Numeric Type Overview”](#).

If accuracy is not too important or if speed is the highest priority, the `DOUBLE` type may be good enough. For high precision, you can always convert to a fixed-point type stored in a `BIGINT`. This enables you to do all calculations with 64-bit integers and then convert results back to floating-point values as necessary.

`PROCEDURE ANALYSE` can be used to obtain suggestions for optimal column data types. For more information, see [Section 8.4.2.4, “Using PROCEDURE ANALYSE”](#).

11.8. Using Data Types from Other Database Engines

To facilitate the use of code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL.

Other Vendor Type	MySQL Type
BOOL	TINYINT
BOOLEAN	TINYINT
CHARACTER VARYING(<i>M</i>)	VARCHAR(<i>M</i>)
FIXED	DECIMAL
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a `DESCRIBE tbl_name` statement, MySQL reports the table structure using the equivalent MySQL types. For example:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE t;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| a     | tinyint(1)   | YES  |     | NULL    |       |
| b     | double       | YES  |     | NULL    |       |
| c     | mediumtext   | YES  |     | NULL    |       |
| d     | decimal(10,0)| YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

Chapter 12. Functions and Operators

Table of Contents

12.1. Function and Operator Reference	1076
12.2. Type Conversion in Expression Evaluation	1083
12.3. Operators	1086
12.3.1. Operator Precedence	1087
12.3.2. Comparison Functions and Operators	1088
12.3.3. Logical Operators	1094
12.3.4. Assignment Operators	1095
12.4. Control Flow Functions	1097
12.5. String Functions	1099
12.5.1. String Comparison Functions	1111
12.5.2. Regular Expressions	1114
12.6. Numeric Functions and Operators	1120
12.6.1. Arithmetic Operators	1121
12.6.2. Mathematical Functions	1123
12.7. Date and Time Functions	1131
12.8. What Calendar Is Used By MySQL?	1152
12.9. Full-Text Search Functions	1153
12.9.1. Natural Language Full-Text Searches	1154
12.9.2. Boolean Full-Text Searches	1157
12.9.3. Full-Text Searches with Query Expansion	1160
12.9.4. Full-Text Stopwords	1160
12.9.5. Full-Text Restrictions	1164
12.9.6. Fine-Tuning MySQL Full-Text Search	1164
12.9.7. Adding a Collation for Full-Text Indexing	1166
12.10. Cast Functions and Operators	1168
12.11. XML Functions	1171
12.12. Bit Functions	1182
12.13. Encryption and Compression Functions	1183
12.14. Information Functions	1190
12.15. Miscellaneous Functions	1198
12.16. Functions and Modifiers for Use with <code>GROUP BY</code> Clauses	1203
12.16.1. <code>GROUP BY</code> (Aggregate) Functions	1203
12.16.2. <code>GROUP BY</code> Modifiers	1207
12.16.3. MySQL Extensions to <code>GROUP BY</code>	1209
12.17. Spatial Extensions	1211
12.17.1. Introduction to MySQL Spatial Support	1211
12.17.2. The OpenGIS Geometry Model	1212
12.17.3. Supported Spatial Data Formats	1218
12.17.4. Creating a Spatially Enabled MySQL Database	1219
12.17.5. Spatial Analysis Functions	1225
12.17.6. Optimizing Spatial Analysis	1234
12.17.7. MySQL Conformance and Compatibility	1237
12.18. Precision Math	1237
12.18.1. Types of Numeric Values	1238
12.18.2. <code>DECIMAL</code> Data Type Changes	1238
12.18.3. Expression Handling	1240
12.18.4. Rounding Behavior	1241
12.18.5. Precision Math Examples	1242

Expressions can be used at several points in SQL statements, such as in the `ORDER BY` or `HAVING` clauses of `SELECT` statements, in the `WHERE` clause of a `SELECT`, `DELETE`, or `UPDATE` statement, or in `SET` statements. Expressions can be written using literal values, column values, `NULL`, built-in functions, stored functions, user-defined functions, and operators. This chapter describes the functions and operators that are permitted for writing expressions in MySQL. Instructions for writing stored functions and user-defined functions are given in [Section 19.2, “Using Stored Routines \(Procedures and Functions\)”](#), and [Section 23.3, “Adding New Functions to MySQL”](#). See [Section 9.2.4, “Function Name Parsing and Resolution”](#), for the rules describing how the server interprets references to different kinds of functions.

An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for a particular function or operator.



Note

By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. However, spaces around function arguments are permitted.

You can tell the MySQL server to accept spaces after function names by starting it with the `--sql-mode=IGNORE_SPACE` [464] option. (See [Section 5.1.7, “Server SQL Modes”](#).) Individual client programs can request this behavior by using the `CLIENT_IGNORE_SPACE` option for `mysql_real_connect()`. In either case, all function names become reserved words.

For the sake of brevity, most examples in this chapter display the output from the `mysql` program in abbreviated form. Rather than showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----+
| mod(29,9) |
+-----+
|          2 |
+-----+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
-> 2
```

12.1. Function and Operator Reference

Table 12.1. Functions/Operators

Name	Description
ABS() [1123]	Return the absolute value
ACOS() [1124]	Return the arc cosine
ADDDATE() [1134]	Add time values (intervals) to a date value
ADDTIME() [1134]	Add time
AES_DECRYPT() [1185]	Decrypt using AES
AES_ENCRYPT() [1185]	Encrypt using AES
AND, && [1094]	Logical AND
ASCII() [1101]	Return numeric value of left-most character
ASIN() [1124]	Return the arc sine

Name	Description
= [1096]	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)
:= [1095]	Assign a value
ATAN2 (), ATAN () [1124]	Return the arc tangent of the two arguments
ATAN () [1124]	Return the arc tangent
AVG () [1203]	Return the average value of the argument
BENCHMARK () [1190]	Repeatedly execute an expression
BETWEEN ... AND ... [1091]	Check whether a value is within a range of values
BIN () [1101]	Return a string containing binary representation of a number
BINARY [1168]	Cast a string to a binary string
BIT_AND () [1204]	Return bitwise and
BIT_COUNT () [1183]	Return the number of bits that are set
BIT_LENGTH () [1101]	Return length of argument in bits
BIT_OR () [1204]	Return bitwise or
BIT_XOR () [1204]	Return bitwise xor
& [1182]	Bitwise AND
~ [1183]	Invert bits
[1182]	Bitwise OR
^ [1182]	Bitwise XOR
CASE [1097]	Case operator
CAST () [1168]	Cast a value as a certain type
CEIL () [1124]	Return the smallest integer value not less than the argument
CEILING () [1125]	Return the smallest integer value not less than the argument
CHAR_LENGTH () [1102]	Return number of characters in argument
CHAR () [1101]	Return the character for each integer passed
CHARACTER_LENGTH () [1102]	A synonym for CHAR_LENGTH()
CHARSET () [1191]	Return the character set of the argument
COALESCE () [1092]	Return the first non-NULL argument
COERCIBILITY () [1191]	Return the collation coercibility value of the string argument
COLLATION () [1191]	Return the collation of the string argument
COMPRESS () [1185]	Return result as a binary string
CONCAT_WS () [1102]	Return concatenate with separator
CONCAT () [1102]	Return concatenated string
CONNECTION_ID () [1191]	Return the connection ID (thread ID) for the connection
CONV () [1125]	Convert numbers between different number bases
CONVERT_TZ () [1134]	Convert from one timezone to another
CONVERT () [1168]	Cast a value as a certain type
COS () [1125]	Return the cosine

Name	Description
<code>COT()</code> [1125]	Return the cotangent
<code>COUNT(DISTINCT)</code> [1204]	Return the count of a number of different values
<code>COUNT()</code> [1204]	Return a count of the number of rows returned
<code>CRC32()</code> [1125]	Compute a cyclic redundancy check value
<code>CURDATE()</code> [1135]	Return the current date
<code>CURRENT_DATE()</code> , <code>CURRENT_DATE</code> [1135]	Synonyms for <code>CURDATE()</code>
<code>CURRENT_TIME()</code> , <code>CURRENT_TIME</code> [1135]	Synonyms for <code>CURTIME()</code>
<code>CURRENT_TIMESTAMP()</code> , <code>CURRENT_TIMESTAMP</code> [1135]	Synonyms for <code>NOW()</code>
<code>CURRENT_USER()</code> , <code>CURRENT_USER</code> [1192]	The authenticated user name and host name
<code>CURTIME()</code> [1135]	Return the current time
<code>DATABASE()</code> [1193]	Return the default (current) database name
<code>DATE_ADD()</code> [1135]	Add time values (intervals) to a date value
<code>DATE_FORMAT()</code> [1138]	Format date as specified
<code>DATE_SUB()</code> [1139]	Subtract a time value (interval) from a date
<code>DATE()</code> [1135]	Extract the date part of a date or datetime expression
<code>DATEDIFF()</code> [1135]	Subtract two dates
<code>DAY()</code> [1139]	Synonym for <code>DAYOFMONTH()</code>
<code>DAYNAME()</code> [1140]	Return the name of the weekday
<code>DAYOFMONTH()</code> [1140]	Return the day of the month (0-31)
<code>DAYOFWEEK()</code> [1140]	Return the weekday index of the argument
<code>DAYOFYEAR()</code> [1140]	Return the day of the year (1-366)
<code>DECODE()</code> [1186]	Decodes a string encrypted using <code>ENCODE()</code>
<code>DEFAULT()</code> [1198]	Return the default value for a table column
<code>DEGREES()</code> [1125]	Convert radians to degrees
<code>DES_DECRYPT()</code> [1186]	Decrypt a string
<code>DES_ENCRYPT()</code> [1186]	Encrypt a string
<code>DIV</code> [1122]	Integer division
<code>/</code> [1122]	Division operator
<code>ELT()</code> [1102]	Return string at index number
<code>ENCODE()</code> [1187]	Encode a string
<code>ENCRYPT()</code> [1187]	Encrypt a string
<code><=></code> [1090]	NULL-safe equal to operator
<code>=</code> [1089]	Equal operator
<code>EXP()</code> [1126]	Raise to the power of

Name	Description
EXPORT_SET() [1103]	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
EXTRACT() [1140]	Extract part of a date
ExtractValue() [1173]	Extracts a value from an XML string using XPath notation
FIELD() [1103]	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET() [1103]	Return the index position of the first argument within the second argument
FLOOR() [1126]	Return the largest integer value not greater than the argument
FORMAT() [1103]	Return a number formatted to specified number of decimal places
FOUND_ROWS() [1193]	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
FROM_DAYS() [1140]	Convert a day number to a date
FROM_UNIXTIME() [1140]	Format UNIX timestamp as a date
GET_FORMAT() [1141]	Return a date format string
GET_LOCK() [1199]	Get a named lock
>= [1090]	Greater than or equal operator
> [1090]	Greater than operator
GREATEST() [1092]	Return the largest argument
GROUP_CONCAT() [1205]	Return a concatenated string
HEX() [1104]	Return a hexadecimal representation of a decimal or string value
HOUR() [1142]	Extract the hour
IF() [1097]	If/else construct
IFNULL() [1098]	Null if/else construct
IN() [1092]	Check whether a value is within a set of values
INET_ATON() [1199]	Return the numeric value of an IP address
INET_NTOA() [1200]	Return the IP address from a numeric value
INSERT() [1104]	Insert a substring at the specified position up to the specified number of characters
INSTR() [1104]	Return the index of the first occurrence of substring
INTERVAL() [1093]	Return the index of the argument that is less than the first argument
IS_FREE_LOCK() [1200]	Checks whether the named lock is free
IS NOT NULL [1091]	NOT NULL value test
IS NOT [1090]	Test a value against a boolean
IS NULL [1091]	NULL value test
IS_USED_LOCK() [1200]	Checks whether the named lock is in use. Return connection identifier if true.

Name	Description
IS [1090]	Test a value against a boolean
ISNULL() [1093]	Test whether the argument is NULL
LAST_DAY [1142]	Return the last day of the month for the argument
LAST_INSERT_ID() [1194]	Value of the AUTOINCREMENT column for the last INSERT
LCASE() [1104]	Synonym for LOWER()
LEAST() [1093]	Return the smallest argument
<< [1183]	Left shift
LEFT() [1104]	Return the leftmost number of characters as specified
LENGTH() [1104]	Return the length of a string in bytes
<= [1090]	Less than or equal operator
< [1090]	Less than operator
LIKE [1111]	Simple pattern matching
LN() [1126]	Return the natural logarithm of the argument
LOAD_FILE() [1105]	Load the named file
LOCALTIME(), LOCALTIME [1142]	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP() [1142]	Synonym for NOW()
LOCATE() [1105]	Return the position of the first occurrence of substring
LOG10() [1127]	Return the base-10 logarithm of the argument
LOG2() [1127]	Return the base-2 logarithm of the argument
LOG() [1126]	Return the natural logarithm of the first argument
LOWER() [1105]	Return the argument in lowercase
LPAD() [1106]	Return the string argument, left-padded with the specified string
LTRIM() [1106]	Remove leading spaces
MAKE_SET() [1106]	Return a set of comma-separated strings that have the corresponding bit in bits set
MAKEDATE() [1142]	Create a date from the year and day of year
MAKETIME [1142]	MAKETIME()
MASTER_POS_WAIT() [1200]	Block until the slave has read and applied all updates up to the specified position
MATCH [1153]	Perform full-text search
MAX() [1205]	Return the maximum value
MD5() [1187]	Calculate MD5 checksum
MICROSECOND() [1142]	Return the microseconds from argument
MID() [1106]	Return a substring starting from the specified position
MIN() [1206]	Return the minimum value
- [1122]	Minus operator
MINUTE() [1142]	Return the minute from the argument

Name	Description
MOD() [1127]	Return the remainder
% or MOD [1122]	Modulo operator
MONTH() [1143]	Return the month from the date passed
MONTHNAME() [1143]	Return the name of the month
NAME_CONST() [1201]	Causes the column to have the given name
NOT BETWEEN ... AND ... [1092]	Check whether a value is not within a range of values
!=, <> [1090]	Not equal operator
NOT IN() [1093]	Check whether a value is not within a set of values
NOT LIKE [1113]	Negation of simple pattern matching
NOT REGEXP [1114]	Negation of REGEXP
NOT, ! [1094]	Negates value
NOW() [1143]	Return the current date and time
NULLIF() [1099]	Return NULL if expr1 = expr2
OCT() [1106]	Return a string containing octal representation of a number
OCTET_LENGTH() [1106]	A synonym for LENGTH()
OLD_PASSWORD() [1187]	Return the value of the pre-4.1 implementation of PASSWORD
, OR [1095]	Logical OR
ORD() [1106]	Return character code for leftmost character of the argument
PASSWORD() [1188]	Calculate and return a password string
PERIOD_ADD() [1144]	Add a period to a year-month
PERIOD_DIFF() [1144]	Return the number of months between periods
PI() [1127]	Return the value of pi
+ [1121]	Addition operator
POSITION() [1107]	A synonym for LOCATE()
POW() [1128]	Return the argument raised to the specified power
POWER() [1128]	Return the argument raised to the specified power
PROCEDURE ANALYSE()	Analyze the results of a query
QUARTER() [1144]	Return the quarter from a date argument
QUOTE() [1107]	Escape the argument for use in an SQL statement
RADIANS() [1128]	Return argument converted to radians
RAND() [1128]	Return a random floating-point value
REGEXP [1114]	Pattern matching using regular expressions
RELEASE_LOCK() [1201]	Releases the named lock
REPEAT() [1107]	Repeat a string the specified number of times
REPLACE() [1107]	Replace occurrences of a specified string
REVERSE() [1107]	Reverse the characters in a string
>> [1183]	Right shift
RIGHT() [1107]	Return the specified rightmost number of characters

Name	Description
RLIKE [1114]	Synonym for REGEXP
ROUND() [1129]	Round the argument
ROW_COUNT() [1196]	The number of rows updated
RPAD() [1107]	Append string the specified number of times
RTRIM() [1108]	Remove trailing spaces
SCHEMA() [1197]	A synonym for DATABASE()
SEC_TO_TIME() [1144]	Converts seconds to 'HH:MM:SS' format
SECOND() [1144]	Return the second (0-59)
SESSION_USER() [1197]	Synonym for USER()
SHA1(), SHA() [1189]	Calculate an SHA-1 160-bit checksum
SHA2() [1189]	Calculate an SHA-2 checksum
SIGN() [1130]	Return the sign of the argument
SIN() [1130]	Return the sine of the argument
SLEEP() [1201]	Sleep for a number of seconds
SOUNDEX() [1108]	Return a soundex string
SOUNDS LIKE [1108]	Compare sounds
SPACE() [1108]	Return a string of the specified number of spaces
SQRT() [1130]	Return the square root of the argument
STD() [1206]	Return the population standard deviation
STDDEV_POP() [1206]	Return the population standard deviation
STDDEV_SAMP() [1206]	Return the sample standard deviation
STDDEV() [1206]	Return the population standard deviation
STR_TO_DATE() [1144]	Convert a string to a date
STRCMP() [1113]	Compare two strings
SUBDATE() [1145]	A synonym for DATE_SUB() when invoked with three arguments
SUBSTR() [1109]	Return the substring as specified
SUBSTRING_INDEX() [1109]	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING() [1109]	Return the substring as specified
SUBTIME() [1145]	Subtract times
SUM() [1206]	Return the sum
SYSDATE() [1146]	Return the time at which the function executes
SYSTEM_USER() [1197]	Synonym for USER()
TAN() [1131]	Return the tangent of the argument
TIME_FORMAT() [1148]	Format as time
TIME_TO_SEC() [1148]	Return the argument converted to seconds
TIME() [1146]	Extract the time portion of the expression passed

Name	Description
<code>TIMEDIFF()</code> [1146]	Subtract time
<code>*</code> [1122]	Multiplication operator
<code>TIMESTAMP()</code> [1147]	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
<code>TIMESTAMPADD()</code> [1147]	Add an interval to a datetime expression
<code>TIMESTAMPDIFF()</code> [1147]	Subtract an interval from a datetime expression
<code>TO_DAYS()</code> [1148]	Return the date argument converted to days
<code>TO_SECONDS()</code> [1149]	Return the date or datetime argument converted to seconds since Year 0
<code>TRIM()</code> [1109]	Remove leading and trailing spaces
<code>TRUNCATE()</code> [1131]	Truncate to specified number of decimal places
<code>UCASE()</code> [1110]	Synonym for <code>UPPER()</code>
<code>-</code> [1122]	Change the sign of the argument
<code>UNCOMPRESS()</code> [1189]	Uncompress a string compressed
<code>UNCOMPRESSED_LENGTH()</code> [1189]	Return the length of a string before compression
<code>UNHEX()</code> [1110]	Return a string containing hex representation of a number
<code>UNIX_TIMESTAMP()</code> [1150]	Return a UNIX timestamp
<code>UpdateXML()</code> [1174]	Return replaced XML fragment
<code>UPPER()</code> [1110]	Convert to uppercase
<code>USER()</code> [1197]	The user name and host name provided by the client
<code>UTC_DATE()</code> [1150]	Return the current UTC date
<code>UTC_TIME()</code> [1150]	Return the current UTC time
<code>UTC_TIMESTAMP()</code> [1151]	Return the current UTC date and time
<code>UUID_SHORT()</code> [1202]	Return an integer-valued universal identifier
<code>UUID()</code> [1201]	Return a Universal Unique Identifier (UUID)
<code>VALUES()</code> [1202]	Defines the values to be used during an <code>INSERT</code>
<code>VAR_POP()</code> [1207]	Return the population standard variance
<code>VAR_SAMP()</code> [1207]	Return the sample variance
<code>VARIANCE()</code> [1207]	Return the population standard variance
<code>VERSION()</code> [1198]	Returns a string that indicates the MySQL server version
<code>WEEK()</code> [1151]	Return the week number
<code>WEEKDAY()</code> [1152]	Return the weekday index
<code>WEEKOFYEAR()</code> [1152]	Return the calendar week of the date (0-53)
<code>XOR</code> [1095]	Logical XOR
<code>YEAR()</code> [1152]	Return the year
<code>YEARWEEK()</code> [1152]	Return the year and week

12.2. Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For example, MySQL automatically converts numbers to strings as necessary, and vice versa.

```
mysql> SELECT 1+'1';
-> 2
mysql> SELECT CONCAT(2,' test');
-> '2 test'
```

It is also possible to convert a number to a string explicitly using the `CAST()` [1168] function. Conversion occurs implicitly with the `CONCAT()` [1102] function because it expects string arguments.

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
-> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
-> 38.8, '38.8'
```

See later in this section for information about the character set of implicit number-to-string conversions.

The following rules describe how conversion occurs for comparison operations:

- If one or both arguments are `NULL`, the result of the comparison is `NULL`, except for the `NULL`-safe `<=>` [1090] equality comparison operator. For `NULL <=> NULL`, the result is true. No conversion is needed.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a `TIMESTAMP` or `DATETIME` column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. Note that this is not done for the arguments to `IN()` [1092]! To be safe, always use complete datetime, date, or time strings when doing comparisons. For example, to achieve best results when using `BETWEEN` [1091] with date or time values, use `CAST()` [1168] to explicitly convert the values to the desired data type.
- If one of the arguments is a decimal value, comparison depends on the other argument. The arguments are compared as decimal values if the other argument is a decimal or integer value, or as floating-point values if the other argument is a floating-point value.
- In all other cases, the arguments are compared as floating-point (real) numbers.

For information about conversion of values from one temporal type to another, see [Section 11.3.7, “Conversion Between Date and Time Types”](#).

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
-> 0
mysql> SELECT 7 > '6x';
-> 1
mysql> SELECT 0 > 'x6';
-> 0
mysql> SELECT 0 = 'x6';
-> 1
```

For comparisons of a string column with a number, MySQL cannot use an index on the column to look up the value quickly. If `str_col` is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value 1, such as '1', '1 ', or '1a'.

Comparisons that use floating-point numbers (or values that are converted to floating-point numbers) are approximate because such numbers are inexact. This might lead to results that appear inconsistent:

```
mysql> SELECT '18015376320243458' = 18015376320243458;
-> 1
mysql> SELECT '18015376320243459' = 18015376320243459;
-> 0
```

Such results can occur because the values are converted to floating-point numbers, which have only 53 bits of precision and are subject to rounding:

```
mysql> SELECT '18015376320243459'+0.0;
-> 1.8015376320243e+16
```

Furthermore, the conversion from string to floating-point and from integer to floating-point do not necessarily occur the same way. The integer may be converted to floating-point by the CPU, whereas the string is converted digit by digit in an operation that involves floating-point multiplications.

The results shown will vary on different systems, and can be affected by factors such as computer architecture or the compiler version or optimization level. One way to avoid such problems is to use `CAST()` [1168] so that a value will not be converted implicitly to a float-point number:

```
mysql> SELECT CAST('18015376320243459' AS UNSIGNED) = 18015376320243459;
-> 1
```

For more information about floating-point comparisons, see [Section C.5.5.8, “Problems with Floating-Point Values”](#).

As of MySQL 5.5.3, the server includes `dtoa`, a conversion library that provides the basis for improved conversion between string or `DECIMAL` values and approximate-value (`FLOAT/DOUBLE`) numbers:

- Consistent conversion results across platforms, which eliminates, for example, Unix versus Windows conversion differences.
- Accurate representation of values in cases where results previously did not provide sufficient precision, such as for values close to IEEE limits.
- Conversion of numbers to string format with the best possible precision. The precision of `dtoa` is always the same or better than that of the standard C library functions.

Because the conversions produced by this library differ in some cases from previous results, the potential exists for incompatibilities in applications that rely on previous results. For example, applications that depend on a specific exact result from previous conversions might need adjustment to accommodate additional precision.

The `dtoa` library provides conversions with the following properties. *D* represents a value with a `DECIMAL` or string representation, and *F* represents a floating-point number in native binary (IEEE) format.

- *F* -> *D* conversion is done with the best possible precision, returning *D* as the shortest string that yields *F* when read back in and rounded to the nearest value in native binary format as specified by IEEE.
- *D* -> *F* conversion is done such that *F* is the nearest native binary number to the input decimal string *D*.

These properties imply that $F \rightarrow D \rightarrow F$ conversions are lossless unless F is `-inf`, `+inf`, or `NaN`. The latter values are not supported because the SQL standard defines them as invalid values for `FLOAT` or `DOUBLE`.

For $D \rightarrow F \rightarrow D$ conversions, a sufficient condition for losslessness is that D uses 15 or fewer digits of precision, is not a denormal value, `-inf`, `+inf`, or `NaN`. In some cases, the conversion is lossless even if D has more than 15 digits of precision, but this is not always the case.

As of MySQL 5.5.3, implicit conversion of a numeric or temporal value to string produces a value that has a character set and collation determined by the `character_set_connection` [487] and `collation_connection` [489] system variables. (These variables commonly are set with `SET NAMES`. For information about connection character sets, see [Section 10.1.4, "Connection Character Sets and Collations"](#).)

This change means that such a conversion results in a character (nonbinary) string (a `CHAR`, `VARCHAR`, or `LONGTEXT` value), except when the connection character set is set to `binary`. In that case, the conversion result is a binary string (a `BINARY`, `VARBINARY`, or `LOBLOB` value).

Before MySQL 5.5.3, an implicit conversion always produced a binary string, regardless of the connection character set. Such implicit conversions to string typically occur for functions that are passed numeric or temporal values when string values are more usual, and thus could have effects beyond the type of the converted value. Consider the expression `CONCAT(1, 'abc')` [1102]. The numeric argument `1` was converted to the binary string `'1'` and the concatenation of that value with the nonbinary string `'abc'` produced the binary string `'1abc'`.

Some functions are unaffected by this change in behavior:

- `CHAR()` [1101] without a `USING` clause still returns `VARBINARY`.
- Functions that previously returned `utf8` strings still do so. Examples include `CHARSET()` [1191] and `COLLATION()` [1191].
- Encryption and compression functions that expect string arguments and previously returned binary strings are unaffected if the return value can contain non-ASCII characters. Examples include `AES_ENCRYPT()` [1185] and `COMPRESS()` [1185]. If the return value contains only ASCII characters, the function now returns a character string with the connection character set and collation. Examples include `MD5()` [1187] and `PASSWORD()` [1188].

12.3. Operators

Table 12.2. Operators

Name	Description
<code>AND</code> , <code>&&</code> [1094]	Logical AND
<code>=</code> [1096]	Assign a value (as part of a <code>SET</code> statement, or as part of the <code>SET</code> clause in an <code>UPDATE</code> statement)
<code>:=</code> [1095]	Assign a value
<code>BETWEEN ... AND ...</code> [1091]	Check whether a value is within a range of values
<code>BINARY</code> [1168]	Cast a string to a binary string
<code>&</code> [1182]	Bitwise AND
<code>~</code> [1183]	Invert bits
<code> </code> [1182]	Bitwise OR

Name	Description
<code>^</code> [1182]	Bitwise XOR
<code>CASE</code> [1097]	Case operator
<code>DIV</code> [1122]	Integer division
<code>/</code> [1122]	Division operator
<code><=></code> [1090]	NULL-safe equal to operator
<code>=</code> [1089]	Equal operator
<code>>=</code> [1090]	Greater than or equal operator
<code>></code> [1090]	Greater than operator
<code>IS NOT NULL</code> [1091]	NOT NULL value test
<code>IS NOT</code> [1090]	Test a value against a boolean
<code>IS NULL</code> [1091]	NULL value test
<code>IS</code> [1090]	Test a value against a boolean
<code><<</code> [1183]	Left shift
<code><=</code> [1090]	Less than or equal operator
<code><</code> [1090]	Less than operator
<code>LIKE</code> [1111]	Simple pattern matching
<code>-</code> [1122]	Minus operator
<code>%</code> or <code>MOD</code> [1122]	Modulo operator
<code>NOT BETWEEN ... AND ...</code> [1092]	Check whether a value is not within a range of values
<code>!=, <></code> [1090]	Not equal operator
<code>NOT LIKE</code> [1113]	Negation of simple pattern matching
<code>NOT REGEXP</code> [1114]	Negation of REGEXP
<code>NOT, !</code> [1094]	Negates value
<code> , OR</code> [1095]	Logical OR
<code>+</code> [1121]	Addition operator
<code>REGEXP</code> [1114]	Pattern matching using regular expressions
<code>>></code> [1183]	Right shift
<code>RLIKE</code> [1114]	Synonym for REGEXP
<code>SOUNDS LIKE</code> [1108]	Compare sounds
<code>*</code> [1122]	Multiplication operator
<code>-</code> [1122]	Change the sign of the argument
<code>XOR</code> [1095]	Logical XOR

12.3.1. Operator Precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
```

```

^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
&&, AND
XOR
||, OR
= (assignment), :=
    
```

The precedence of = depends on whether it is used as a comparison operator (= [1089]) or as an assignment operator (= [1096]). When used as a comparison operator, it has the same precedence as <=> [1090], >= [1090], > [1090], <= [1090], < [1090], <> [1090], != [1090], IS [1090], LIKE [1111], REGEXP [1114], and IN [1092]. When used as an assignment operator, it has the same precedence as := [1095]. Section 13.7.4, “SET Syntax”, and Section 9.4, “User-Defined Variables”, explain how MySQL determines which interpretation of = should apply.

The meaning of some operators depends on the SQL mode:

- By default, || [1095] is a logical OR [1095] operator. With PIPES_AS_CONCAT [616] enabled, || [1095] is string concatenation, with a precedence between ^ [1182] and the unary operators.
- By default, ! [1094] has a higher precedence than NOT. With HIGH_NOT_PRECEDENCE [613] enabled, ! [1094] and NOT have the same precedence.

See Section 5.1.7, “Server SQL Modes”.

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses. For example:

```

mysql> SELECT 1+2*3;
-> 7
mysql> SELECT (1+2)*3;
-> 9
    
```

12.3.2. Comparison Functions and Operators

Table 12.3. Comparison Operators

Name	Description
BETWEEN ... AND ... [1091]	Check whether a value is within a range of values
COALESCE() [1092]	Return the first non-NULL argument
<=> [1090]	NULL-safe equal to operator
= [1089]	Equal operator
>= [1090]	Greater than or equal operator
> [1090]	Greater than operator
GREATEST() [1092]	Return the largest argument
IN() [1092]	Check whether a value is within a set of values
INTERVAL() [1093]	Return the index of the argument that is less than the first argument

Name	Description
<code>IS NOT NULL</code> [1091]	NOT NULL value test
<code>IS NOT</code> [1090]	Test a value against a boolean
<code>IS NULL</code> [1091]	NULL value test
<code>IS</code> [1090]	Test a value against a boolean
<code>ISNULL()</code> [1093]	Test whether the argument is NULL
<code>LEAST()</code> [1093]	Return the smallest argument
<code><=</code> [1090]	Less than or equal operator
<code><</code> [1090]	Less than operator
<code>LIKE</code> [1111]	Simple pattern matching
<code>NOT BETWEEN ... AND ...</code> [1092]	Check whether a value is not within a range of values
<code>!=, <></code> [1090]	Not equal operator
<code>NOT IN()</code> [1093]	Check whether a value is not within a set of values
<code>NOT LIKE</code> [1113]	Negation of simple pattern matching
<code>STRCMP()</code> [1113]	Compare two strings

Comparison operations result in a value of 1 (TRUE), 0 (FALSE), or NULL. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

The following relational comparison operators can be used to compare not only scalar operands, but row operands:

```
= > < >= <= <> !=
```

For examples of row comparisons, see [Section 13.2.10.5, “Row Subqueries”](#).

Some of the functions in this section return values other than 1 (TRUE), 0 (FALSE), or NULL. For example, `LEAST()` [1093] and `GREATEST()` [1092]. However, the value they return is based on comparison operations performed according to the rules described in [Section 12.2, “Type Conversion in Expression Evaluation”](#).

To convert a value to a specific type for comparison purposes, you can use the `CAST()` [1168] function. String values can be converted to a different character set using `CONVERT()` [1168]. See [Section 12.10, “Cast Functions and Operators”](#).

By default, string comparisons are not case sensitive and use the current character set. The default is `latin1` (cp1252 West European), which also works well for English.

- `=` [1089]

Equal:

```
mysql> SELECT 1 = 0;
-> 0
mysql> SELECT '0' = 0;
-> 1
mysql> SELECT '0.0' = 0;
-> 1
mysql> SELECT '0.01' = 0;
```

```
mysql> SELECT '.01' = 0.01;
-> 0
```

- `<=>` [1090]

`NULL`-safe equal. This operator performs an equality comparison like the `=` [1089] operator, but returns `1` rather than `NULL` if both operands are `NULL`, and `0` rather than `NULL` if one operand is `NULL`.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
-> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
-> 1, NULL, NULL
```

- `<>` [1090], `!=` [1090]

Not equal:

```
mysql> SELECT '.01' <> '0.01';
-> 1
mysql> SELECT .01 <> '0.01';
-> 0
mysql> SELECT 'zapp' <> 'zappp';
-> 1
```

- `<=` [1090]

Less than or equal:

```
mysql> SELECT 0.1 <= 2;
-> 1
```

- `<` [1090]

Less than:

```
mysql> SELECT 2 < 2;
-> 0
```

- `>=` [1090]

Greater than or equal:

```
mysql> SELECT 2 >= 2;
-> 1
```

- `>` [1090]

Greater than:

```
mysql> SELECT 2 > 2;
-> 0
```

- `IS boolean_value` [1090]

Tests a value against a boolean value, where `boolean_value` can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
-> 1, 1, 1
```

- `IS NOT boolean_value` [1090]

Tests a value against a boolean value, where `boolean_value` can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
-> 1, 1, 0
```

- [IS NULL \[1091\]](#)

Tests whether a value is `NULL`.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
-> 0, 0, 1
```

To work well with ODBC programs, MySQL supports the following extra features when using `IS NULL [1091]`:

- If `sql_auto_is_null [554]` variable is set to 1, then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID() [1194]` function. For details, including the return value after a multiple-row insert, see [Section 12.14, “Information Functions”](#). If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL [1091]` comparison can be disabled by setting `sql_auto_is_null = 0 [554]`. See [Section 5.1.4, “Server System Variables”](#).

The default value of `sql_auto_is_null [554]` is 0 as of MySQL 5.5.3, and 1 for earlier versions.

- For `DATE` and `DATETIME` columns that are declared as `NOT NULL`, you can find the special date `'0000-00-00'` by using a statement like this:

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

This is needed to get some ODBC applications to work because ODBC does not support a `'0000-00-00'` date value.

See [Section 22.1.7.1.1, “Obtaining Auto-Increment Values”](#), and the description for the `FLAG_AUTO_IS_NULL` option at [Section 22.1.4.2, “Connector/ODBC Connection Parameters”](#).

- [IS NOT NULL \[1091\]](#)

Tests whether a value is not `NULL`.

```
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
-> 1, 1, 0
```

- [expr BETWEEN min AND max \[1091\]](#)

If `expr` is greater than or equal to `min` and `expr` is less than or equal to `max`, `BETWEEN [1091]` returns 1, otherwise it returns 0. This is equivalent to the expression `(min <= expr AND expr <= max)` if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described in [Section 12.2, “Type Conversion in Expression Evaluation”](#), but applied to all the three arguments.

```
mysql> SELECT 2 BETWEEN 1 AND 3, 2 BETWEEN 3 AND 1;
-> 1, 0
mysql> SELECT 1 BETWEEN 2 AND 3;
```

```

-> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
-> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
-> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
-> 0

```

For best results when using [BETWEEN \[1091\]](#) with date or time values, use [CAST\(\) \[1168\]](#) to explicitly convert the values to the desired data type. Examples: If you compare a [DATETIME](#) to two [DATE](#) values, convert the [DATE](#) values to [DATETIME](#) values. If you use a string constant such as '2001-1-1' in a comparison to a [DATE](#), cast the string to a [DATE](#).

- [expr NOT BETWEEN min AND max \[1092\]](#)

This is the same as `NOT (expr BETWEEN min AND max)`.

- [COALESCE\(value, ...\) \[1092\]](#)

Returns the first non-[NULL](#) value in the list, or [NULL](#) if there are no non-[NULL](#) values.

```

mysql> SELECT COALESCE(NULL,1);
-> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
-> NULL

```

- [GREATEST\(value1,value2,...\) \[1092\]](#)

With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for [LEAST\(\) \[1093\]](#).

```

mysql> SELECT GREATEST(2,0);
-> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
-> 767.0
mysql> SELECT GREATEST('B','A','C');
-> 'C'

```

[GREATEST\(\) \[1092\]](#) returns [NULL](#) if any argument is [NULL](#).

- [expr IN \(value,...\) \[1092\]](#)

Returns [1](#) if [expr](#) is equal to any of the values in the [IN](#) list, else returns [0](#). If all values are constants, they are evaluated according to the type of [expr](#) and sorted. The search for the item then is done using a binary search. This means [IN](#) is very quick if the [IN](#) value list consists entirely of constants. Otherwise, type conversion takes place according to the rules described in [Section 12.2, "Type Conversion in Expression Evaluation"](#), but applied to all the arguments.

```

mysql> SELECT 2 IN (0,3,5,7);
-> 0
mysql> SELECT 'wefwf' IN ('wee','wefwf','weg');
-> 1

```

You should never mix quoted and unquoted values in an [IN](#) list because the comparison rules for quoted values (such as strings) and unquoted values (such as numbers) differ. Mixing types may therefore lead to inconsistent results. For example, do not write an [IN](#) expression like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
```

Instead, write it like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
```

The number of values in the `IN` list is only limited by the `max_allowed_packet` [519] value.

To comply with the SQL standard, `IN` returns `NULL` not only if the expression on the left hand side is `NULL`, but also if no match is found in the list and one of the expressions in the list is `NULL`.

`IN()` syntax can also be used to write certain types of subqueries. See Section 13.2.10.3, “Subqueries with `ANY`, `IN`, or `SOME`”.

- `expr NOT IN (value,...)` [1093]

This is the same as `NOT (expr IN (value,...))`.

- `ISNULL(expr)` [1093]

If `expr` is `NULL`, `ISNULL()` [1093] returns 1, otherwise it returns 0.

```
mysql> SELECT ISNULL(1+1);
-> 0
mysql> SELECT ISNULL(1/0);
-> 1
```

`ISNULL()` [1093] can be used instead of `=` [1089] to test whether a value is `NULL`. (Comparing a value to `NULL` using `=` [1089] always yields false.)

The `ISNULL()` [1093] function shares some special behaviors with the `IS NULL` [1091] comparison operator. See the description of `IS NULL` [1091].

- `INTERVAL(N,N1,N2,N3,...)` [1093]

Returns 0 if $N < N1$, 1 if $N < N2$ and so on or -1 if N is `NULL`. All arguments are treated as integers. It is required that $N1 < N2 < N3 < \dots < Nn$ for this function to work correctly. This is because a binary search is used (very fast).

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
-> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
-> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
-> 0
```

- `LEAST(value1,value2,...)` [1093]

With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

- If any argument is `NULL`, the result is `NULL`. No comparison is needed.
- If the return value is used in an `INTEGER` context or all arguments are integer-valued, they are compared as integers.
- If the return value is used in a `REAL` context or all arguments are real-valued, they are compared as reals.
- If the arguments comprise a mix of numbers and strings, they are compared as numbers.
- If any argument is a nonbinary (character) string, the arguments are compared as nonbinary strings.
- In all other cases, the arguments are compared as binary strings.

```
mysql> SELECT LEAST(2,0);
-> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
-> 3.0
mysql> SELECT LEAST('B','A','C');
-> 'A'
```

Note that the preceding conversion rules can produce strange results in some borderline cases:

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);
-> -9223372036854775808
```

This happens because MySQL reads `9223372036854775808.0` in an integer context. The integer representation is not good enough to hold the value, so it wraps to a signed integer.

12.3.3. Logical Operators

Table 12.4. Logical Operators

Name	Description
AND, && [1094]	Logical AND
NOT, ! [1094]	Negates value
, OR [1095]	Logical OR
XOR [1095]	Logical XOR

In SQL, all logical operators evaluate to `TRUE`, `FALSE`, or `NULL` (`UNKNOWN`). In MySQL, these are implemented as 1 (`TRUE`), 0 (`FALSE`), and `NULL`. Most of this is common to different SQL database servers, although some servers may return any nonzero value for `TRUE`.

MySQL evaluates any nonzero, non-`NULL` value to `TRUE`. For example, the following statements all assess to `TRUE`:

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

- NOT [1094], ! [1094]

Logical NOT. Evaluates to 1 if the operand is 0, to 0 if the operand is nonzero, and `NOT NULL` returns `NULL`.

```
mysql> SELECT NOT 10;
-> 0
mysql> SELECT NOT 0;
-> 1
mysql> SELECT NOT NULL;
-> NULL
mysql> SELECT ! (1+1);
-> 0
mysql> SELECT ! 1+1;
-> 1
```

The last example produces 1 because the expression evaluates the same way as `(!1)+1`.

- AND [1094], && [1094]

Logical AND. Evaluates to `1` if all operands are nonzero and not `NULL`, to `0` if one or more operands are `0`, otherwise `NULL` is returned.

```
mysql> SELECT 1 && 1;
-> 1
mysql> SELECT 1 && 0;
-> 0
mysql> SELECT 1 && NULL;
-> NULL
mysql> SELECT 0 && NULL;
-> 0
mysql> SELECT NULL && 0;
-> 0
```

- [OR \[1095\]](#), [|| \[1095\]](#)

Logical OR. When both operands are non-`NULL`, the result is `1` if any operand is nonzero, and `0` otherwise. With a `NULL` operand, the result is `1` if the other operand is nonzero, and `NULL` otherwise. If both operands are `NULL`, the result is `NULL`.

```
mysql> SELECT 1 || 1;
-> 1
mysql> SELECT 1 || 0;
-> 1
mysql> SELECT 0 || 0;
-> 0
mysql> SELECT 0 || NULL;
-> NULL
mysql> SELECT 1 || NULL;
-> 1
```

- [XOR \[1095\]](#)

Logical XOR. Returns `NULL` if either operand is `NULL`. For non-`NULL` operands, evaluates to `1` if an odd number of operands is nonzero, otherwise `0` is returned.

```
mysql> SELECT 1 XOR 1;
-> 0
mysql> SELECT 1 XOR 0;
-> 1
mysql> SELECT 1 XOR NULL;
-> NULL
mysql> SELECT 1 XOR 1 XOR 1;
-> 1
```

`a XOR b` is mathematically equal to `(a AND (NOT b)) OR ((NOT a) and b)`.

12.3.4. Assignment Operators

Table 12.5. Assignment Operators

Name	Description
<code>=</code> [1096]	Assign a value (as part of a <code>SET</code> statement, or as part of the <code>SET</code> clause in an <code>UPDATE</code> statement)
<code>:=</code> [1095]	Assign a value

- [:= \[1095\]](#)

Assignment operator. Causes the user variable on the left hand side of the operator to take on the value to its right. The value on the right hand side may be a literal value, another variable storing a value, or

any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same `SET` statement. You can perform multiple assignments in the same statement-

Unlike `=` [1096], the `:=` [1095] operator is never interpreted as a comparison operator. This means you can use `:=` [1095] in any valid SQL statement (not just in `SET` statements) to assign a value to a variable.

```
mysql> SELECT @var1, @var2;
      -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
      -> 1, NULL
mysql> SELECT @var1, @var2;
      -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
      -> 1, 1
mysql> SELECT @var1, @var2;
      -> 1, 1

mysql> SELECT @var1:=COUNT(*) FROM t1;
      -> 4
mysql> SELECT @var1;
      -> 4
```

You can make value assignments using `:=` [1095] in other statements besides `SELECT`, such as `UPDATE`, as shown here:

```
mysql> SELECT @var1;
      -> 4
mysql> SELECT * FROM t1;
      -> 1, 3, 5, 7

mysql> UPDATE t1 SET c1 = 2 WHERE c1 = @var1:= 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT @var1;
      -> 1
mysql> SELECT * FROM t1;
      -> 2, 3, 5, 7
```

While it is also possible both to set and to read the value of the same variable in a single SQL statement using the `:=` [1095] operator, this is not recommended. Section 9.4, “User-Defined Variables”, explains why you should avoid doing this.

- `=` [1096]

This operator is used to perform value assignments in two cases, described in the next two paragraphs.

Within a `SET` statement, `=` is treated as an assignment operator that causes the user variable on the left hand side of the operator to take on the value to its right. (In other words, when used in a `SET` statement, `=` is treated identically to `:=` [1095].) The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same `SET` statement.

In the `SET` clause of an `UPDATE` statement, `=` also acts as an assignment operator; in this case, however, it causes the column named on the left hand side of the operator to assume the value given to the right, provided any `WHERE` conditions that are part of the `UPDATE` are met. You can make multiple assignments in the same `SET` clause of an `UPDATE` statement.

In any other context, = is treated as a [comparison operator](#) [1089].

```
mysql> SELECT @var1, @var2;
-> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
-> 1, NULL
mysql> SELECT @var1, @var2;
-> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
-> 1, 1
mysql> SELECT @var1, @var2;
-> 1, 1
```

For more information, see [Section 13.7.4, “SET Syntax”](#), [Section 13.2.11, “UPDATE Syntax”](#), and [Section 13.2.10, “Subquery Syntax”](#).

12.4. Control Flow Functions

Table 12.6. Flow Control Operators

Name	Description
CASE [1097]	Case operator
IF() [1097]	If/else construct
IFNULL() [1098]	Null if/else construct
NULLIF() [1099]	Return NULL if <code>expr1 = expr2</code>

- `CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN result ...] [ELSE result] END` [1097]

```
CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END [1097]
```

The first version returns the *result* where *value=compare_value*. The second version returns the result for the first condition that is true. If there was no matching result value, the result after `ELSE` is returned, or `NULL` if there is no `ELSE` part.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
->      WHEN 2 THEN 'two' ELSE 'more' END;
-> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
-> 'true'
mysql> SELECT CASE BINARY 'B'
->      WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
-> NULL
```

The return type of a `CASE` [1097] expression is the compatible aggregated type of all return values, but also depends on the context in which it is used. If used in a string context, the result is returned as a string. If used in a numeric context, the result is returned as a decimal, real, or integer value.



Note

The syntax of the `CASE` [1097] *expression* shown here differs slightly from that of the SQL `CASE statement` described in [Section 13.6.5.1, “CASE Syntax”](#), for use inside stored programs. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

- `IF(expr1,expr2,expr3)` [1097]

If *expr1* is TRUE (*expr1* <> 0 and *expr1* <> NULL) then `IF()` [1097] returns *expr2*; otherwise it returns *expr3*. `IF()` [1097] returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IF(1>2,2,3);
-> 3
mysql> SELECT IF(1<2,'yes','no');
-> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
-> 'no'
```

If only one of *expr2* or *expr3* is explicitly NULL, the result type of the `IF()` [1097] function is the type of the non-NULL expression.

The default return type of `IF()` [1097] (which may matter when it is stored into a temporary table) is calculated as follows.

Expression	Return Value
<i>expr2</i> or <i>expr3</i> returns a string	string
<i>expr2</i> or <i>expr3</i> returns a floating-point value	floating-point
<i>expr2</i> or <i>expr3</i> returns an integer	integer

If *expr2* and *expr3* are both strings, the result is case sensitive if either string is case sensitive.



Note

There is also an *IF statement*, which differs from the `IF()` [1097] function described here. See [Section 13.6.5.2, “IF Syntax”](#).

- `IFNULL(expr1,expr2)` [1098]

If *expr1* is not NULL, `IFNULL()` [1098] returns *expr1*; otherwise it returns *expr2*. `IFNULL()` [1098] returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IFNULL(1,0);
-> 1
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0,'yes');
-> 'yes'
```

The default result value of `IFNULL(expr1,expr2)` [1098] is the more “general” of the two expressions, in the order `STRING`, `REAL`, or `INTEGER`. Consider the case of a table based on expressions or where MySQL must internally store a value returned by `IFNULL()` [1098] in a temporary table:

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| test  | varbinary(4) | NO   |     |         |       |
+-----+-----+-----+-----+-----+-----+
```

In this example, the type of the `test` column is `VARBINARY(4)`.

- `NULLIF(expr1,expr2)` [1099]

Returns `NULL` if `expr1 = expr2` is true, otherwise returns `expr1`. This is the same as `CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END` [1097].

```
mysql> SELECT NULLIF(1,1);
-> NULL
mysql> SELECT NULLIF(1,2);
-> 1
```

Note that MySQL evaluates `expr1` twice if the arguments are not equal.

12.5. String Functions

Table 12.7. String Operators

Name	Description
<code>ASCII()</code> [1101]	Return numeric value of left-most character
<code>BIN()</code> [1101]	Return a string containing binary representation of a number
<code>BIT_LENGTH()</code> [1101]	Return length of argument in bits
<code>CHAR_LENGTH()</code> [1102]	Return number of characters in argument
<code>CHAR()</code> [1101]	Return the character for each integer passed
<code>CHARACTER_LENGTH()</code> [1102]	A synonym for <code>CHAR_LENGTH()</code>
<code>CONCAT_WS()</code> [1102]	Return concatenate with separator
<code>CONCAT()</code> [1102]	Return concatenated string
<code>ELT()</code> [1102]	Return string at index number
<code>EXPORT_SET()</code> [1103]	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
<code>FIELD()</code> [1103]	Return the index (position) of the first argument in the subsequent arguments
<code>FIND_IN_SET()</code> [1103]	Return the index position of the first argument within the second argument
<code>FORMAT()</code> [1103]	Return a number formatted to specified number of decimal places
<code>HEX()</code> [1104]	Return a hexadecimal representation of a decimal or string value
<code>INSERT()</code> [1104]	Insert a substring at the specified position up to the specified number of characters
<code>INSTR()</code> [1104]	Return the index of the first occurrence of substring
<code>LCASE()</code> [1104]	Synonym for <code>LOWER()</code>
<code>LEFT()</code> [1104]	Return the leftmost number of characters as specified
<code>LENGTH()</code> [1104]	Return the length of a string in bytes
<code>LIKE</code> [1111]	Simple pattern matching
<code>LOAD_FILE()</code> [1105]	Load the named file
<code>LOCATE()</code> [1105]	Return the position of the first occurrence of substring
<code>LOWER()</code> [1105]	Return the argument in lowercase

Name	Description
LPAD() [1106]	Return the string argument, left-padded with the specified string
LTRIM() [1106]	Remove leading spaces
MAKE_SET() [1106]	Return a set of comma-separated strings that have the corresponding bit in bits set
MATCH [1153]	Perform full-text search
MID() [1106]	Return a substring starting from the specified position
NOT LIKE [1113]	Negation of simple pattern matching
NOT REGEXP [1114]	Negation of REGEXP
OCT() [1106]	Return a string containing octal representation of a number
OCTET_LENGTH() [1106]	A synonym for LENGTH()
ORD() [1106]	Return character code for leftmost character of the argument
POSITION() [1107]	A synonym for LOCATE()
QUOTE() [1107]	Escape the argument for use in an SQL statement
REGEXP [1114]	Pattern matching using regular expressions
REPEAT() [1107]	Repeat a string the specified number of times
REPLACE() [1107]	Replace occurrences of a specified string
REVERSE() [1107]	Reverse the characters in a string
RIGHT() [1107]	Return the specified rightmost number of characters
RLIKE [1114]	Synonym for REGEXP
RPAD() [1107]	Append string the specified number of times
RTRIM() [1108]	Remove trailing spaces
SOUNDEX() [1108]	Return a soundex string
SOUNDS LIKE [1108]	Compare sounds
SPACE() [1108]	Return a string of the specified number of spaces
STRCMP() [1113]	Compare two strings
SUBSTR() [1109]	Return the substring as specified
SUBSTRING_INDEX() [1109]	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING() [1109]	Return the substring as specified
TRIM() [1109]	Remove leading and trailing spaces
UCASE() [1110]	Synonym for UPPER()
UNHEX() [1110]	Return a string containing hex representation of a number
UPPER() [1110]	Convert to uppercase

String-valued functions return `NULL` if the length of the result would be greater than the value of the `max_allowed_packet` [519] system variable. See [Section 8.11.2, “Tuning Server Parameters”](#).

For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, noninteger arguments are rounded to the nearest integer.

- `ASCII(str)` [1101]

Returns the numeric value of the leftmost character of the string `str`. Returns 0 if `str` is the empty string. Returns `NULL` if `str` is `NULL`. `ASCII()` [1101] works for 8-bit characters.

```
mysql> SELECT ASCII('2');
      -> 50
mysql> SELECT ASCII(2);
      -> 50
mysql> SELECT ASCII('dx');
      -> 100
```

See also the `ORD()` [1106] function.

- `BIN(N)` [1101]

Returns a string representation of the binary value of `N`, where `N` is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,2)` [1125]. Returns `NULL` if `N` is `NULL`.

```
mysql> SELECT BIN(12);
      -> '1100'
```

- `BIT_LENGTH(str)` [1101]

Returns the length of the string `str` in bits.

```
mysql> SELECT BIT_LENGTH('text');
      -> 32
```

- `CHAR(N,... [USING charset_name])` [1101]

`CHAR()` [1101] interprets each argument `N` as an integer and returns a string consisting of the characters given by the code values of those integers. `NULL` values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
      -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
      -> 'MMM'
```

`CHAR()` [1101] arguments larger than 255 are converted into multiple result bytes. For example, `CHAR(256)` [1101] is equivalent to `CHAR(1,0)` [1101], and `CHAR(256*256)` [1101] is equivalent to `CHAR(1,0,0)` [1101]:

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+-----+-----+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+-----+-----+
| 0100          | 0100          |
+-----+-----+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+-----+-----+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+-----+-----+
| 010000         | 010000         |
+-----+-----+
```

By default, `CHAR()` [1101] returns a binary string. To produce a string in a given character set, use the optional `USING` clause:

```
mysql> SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
+-----+-----+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+-----+-----+
```

binary	utf8

If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from `CHAR()` [1101] becomes `NULL`.

- `CHAR_LENGTH(str)` [1102]

Returns the length of the string `str`, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five 2-byte characters, `LENGTH()` [1104] returns 10, whereas `CHAR_LENGTH()` [1102] returns 5.

- `CHARACTER_LENGTH(str)` [1102]

`CHARACTER_LENGTH()` [1102] is a synonym for `CHAR_LENGTH()` [1102].

- `CONCAT(str1,str2,...)` [1102]

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent string form. This is a nonbinary string as of MySQL 5.5.3. Before 5.5.3, it is a binary string; to avoid that and produce a nonbinary string, you can use an explicit type cast, as in this example:

```
SELECT CONCAT(CAST(int_col AS CHAR), char_col);
```

`CONCAT()` [1102] returns `NULL` if any argument is `NULL`.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
-> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
-> NULL
mysql> SELECT CONCAT(14.3);
-> '14.3'
```

For quoted strings, concatenation can be performed by placing the strings next to each other:

```
mysql> SELECT 'My' 'S' 'QL';
-> 'MySQL'
```

- `CONCAT_WS(separator,str1,str2,...)` [1102]

`CONCAT_WS()` [1102] stands for Concatenate With Separator and is a special form of `CONCAT()` [1102]. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

```
mysql> SELECT CONCAT_WS(',', 'First name', 'Second name', 'Last Name');
-> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(',', 'First name', NULL, 'Last Name');
-> 'First name,Last Name'
```

`CONCAT_WS()` [1102] does not skip empty strings. However, it does skip any `NULL` values after the separator argument.

- `ELT(N,str1,str2,str3,...)` [1102]

Returns `str1` if `N = 1`, `str2` if `N = 2`, and so on. Returns `NULL` if `N` is less than 1 or greater than the number of arguments. `ELT()` [1102] is the complement of `FIELD()` [1103].

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
```

```
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
-> 'foo'
```

- `EXPORT_SET(bits,on,off[,separator[,number_of_bits]])` [1103]

Returns a string such that for every bit set in the value *bits*, you get an *on* string and for every bit not set in the value, you get an *off* string. Bits in *bits* are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the *separator* string (the default being the comma character “,”). The number of bits examined is given by *number_of_bits*, which has a default of 64 if not specified. *number_of_bits* is silently clipped to 64 if larger than 64. It is treated as an unsigned integer, so a value of -1 is effectively the same as 64.

```
mysql> SELECT EXPORT_SET(5,'Y','N',' ',',',4);
-> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6,'1','0',' ',',',10);
-> '0,1,1,0,0,0,0,0,0,0'
```

- `FIELD(str,str1,str2,str3,...)` [1103]

Returns the index (position) of *str* in the *str1*, *str2*, *str3*, ... list. Returns 0 if *str* is not found.

If all arguments to `FIELD()` [1103] are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

If *str* is `NULL`, the return value is 0 because `NULL` fails equality comparison with any value. `FIELD()` [1103] is the complement of `ELT()` [1102].

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 0
```

- `FIND_IN_SET(str,strlist)` [1103]

Returns a value in the range of 1 to *N* if the string *str* is in the string list *strlist* consisting of *N* substrings. A string list is a string composed of substrings separated by “,” characters. If the first argument is a constant string and the second is a column of type `SET`, the `FIND_IN_SET()` [1103] function is optimized to use bit arithmetic. Returns 0 if *str* is not in *strlist* or if *strlist* is the empty string. Returns `NULL` if either argument is `NULL`. This function does not work properly if the first argument contains a comma (“,”) character.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
-> 2
```

- `FORMAT(X,D[,locale])` [1103]

Formats the number *X* to a format like ‘#,###,###.##’, rounded to *D* decimal places, and returns the result as a string. If *D* is 0, the result has no decimal point or fractional part.

The optional third parameter enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` [512] system variable (see Section 10.7, “MySQL Server Locale Support”). If no locale is specified, the default is ‘en_US’.

```
mysql> SELECT FORMAT(12332.123456, 4);
-> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
-> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
```

```
mysql> SELECT FORMAT(12332.2,2,'de_DE');
-> '12.332,20'
```

- [HEX\(*str*\)](#) [1104], [HEX\(*N*\)](#) [1104]

For a string argument *str*, [HEX\(\)](#) [1104] returns a hexadecimal string representation of *str* where each character in *str* is converted to two hexadecimal digits. The inverse of this operation is performed by the [UNHEX\(\)](#) [1110] function.

For a numeric argument *N*, [HEX\(\)](#) [1104] returns a hexadecimal string representation of the value of *N* treated as a longlong ([BIGINT](#)) number. This is equivalent to [CONV\(*N*,10,16\)](#) [1125]. The inverse of this operation is performed by [CONV\(HEX\(*N*\),16,10\)](#) [1125].

```
mysql> SELECT 0x616263, HEX('abc'), UNHEX(HEX('abc'));
-> 'abc', 616263, 'abc'
mysql> SELECT HEX(255), CONV(HEX(255),16,10);
-> 'FF', 255
```

- [INSERT\(*str*,*pos*,*len*,*newstr*\)](#) [1104]

Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string *newstr*. Returns the original string if *pos* is not within the length of the string. Replaces the rest of the string from position *pos* if *len* is not within the length of the rest of the string. Returns [NULL](#) if any argument is [NULL](#).

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
-> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
-> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
-> 'QuWhat'
```

This function is multi-byte safe.

- [INSTR\(*str*,*substr*\)](#) [1104]

Returns the position of the first occurrence of substring *substr* in string *str*. This is the same as the two-argument form of [LOCATE\(\)](#) [1105], except that the order of the arguments is reversed.

```
mysql> SELECT INSTR('foobarbar', 'bar');
-> 4
mysql> SELECT INSTR('xbar', 'foobar');
-> 0
```

This function is multi-byte safe, and is case sensitive only if at least one argument is a binary string.

- [LCASE\(*str*\)](#) [1104]

[LCASE\(\)](#) [1104] is a synonym for [LOWER\(\)](#) [1105].

- [LEFT\(*str*,*len*\)](#) [1104]

Returns the leftmost *len* characters from the string *str*, or [NULL](#) if any argument is [NULL](#).

```
mysql> SELECT LEFT('foobarbar', 5);
-> 'fooba'
```

This function is multi-byte safe.

- [LENGTH\(*str*\)](#) [1104]

Returns the length of the string *str*, measured in bytes. A multi-byte character counts as multiple bytes. This means that for a string containing five 2-byte characters, `LENGTH()` [1104] returns 10, whereas `CHAR_LENGTH()` [1102] returns 5.

```
mysql> SELECT LENGTH('text');
-> 4
```

- `LOAD_FILE(file_name)` [1105]

Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the `FILE` [696] privilege. The file must be readable by all and its size less than `max_allowed_packet` [519] bytes. If the `secure_file_priv` [550] system variable is set to a nonempty directory name, the file to be loaded must be located in that directory.

If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

The `character_set_filesystem` [487] system variable controls interpretation of file names that are given as literal strings.

```
mysql> UPDATE t
      SET blob_col=LOAD_FILE('/tmp/picture')
      WHERE id=1;
```

- `LOCATE(substr, str)` [1105], `LOCATE(substr, str, pos)` [1105]

The first syntax returns the position of the first occurrence of substring *substr* in string *str*. The second syntax returns the position of the first occurrence of substring *substr* in string *str*, starting at position *pos*. Returns 0 if *substr* is not in *str*.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
-> 4
mysql> SELECT LOCATE('xbar', 'foobar');
-> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
-> 7
```

This function is multi-byte safe, and is case-sensitive only if at least one argument is a binary string.

- `LOWER(str)` [1105]

Returns the string *str* with all characters changed to lowercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT LOWER('QUADRATICALLY');
-> 'quadratically'
```

`LOWER()` [1105] (and `UPPER()` [1110]) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-----+-----+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-----+-----+
| New York   | new york                           |
+-----+-----+
```

This function is multi-byte safe.

- [LPAD\(*str*, *len*, *padstr*\) \[1106\]](#)

Returns the string *str*, left-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.

```
mysql> SELECT LPAD('hi',4,'??');
-> '??hi'
mysql> SELECT LPAD('hi',1,'??');
-> 'h'
```

- [LTRIM\(*str*\) \[1106\]](#)

Returns the string *str* with leading space characters removed.

```
mysql> SELECT LTRIM('  barbar');
-> 'barbar'
```

This function is multi-byte safe.

- [MAKE_SET\(*bits*, *str1*, *str2*, ...\) \[1106\]](#)

Returns a set value (a string containing substrings separated by “,” characters) consisting of the strings that have the corresponding bit in *bits* set. *str1* corresponds to bit 0, *str2* to bit 1, and so on. *NULL* values in *str1*, *str2*, ... are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
-> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
-> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
-> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
-> ''
```

- [MID\(*str*, *pos*, *len*\) \[1106\]](#)

[MID\(*str*, *pos*, *len*\) \[1106\]](#) is a synonym for [SUBSTRING\(*str*, *pos*, *len*\) \[1109\]](#).

- [OCT\(*N*\) \[1106\]](#)

Returns a string representation of the octal value of *N*, where *N* is a longlong (*BIGINT*) number. This is equivalent to [CONV\(*N*, 10, 8\) \[1125\]](#). Returns *NULL* if *N* is *NULL*.

```
mysql> SELECT OCT(12);
-> '14'
```

- [OCTET_LENGTH\(*str*\) \[1106\]](#)

[OCTET_LENGTH\(\) \[1106\]](#) is a synonym for [LENGTH\(\) \[1104\]](#).

- [ORD\(*str*\) \[1106\]](#)

If the leftmost character of the string *str* is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
(1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 2562) ...
```

If the leftmost character is not a multi-byte character, [ORD\(\) \[1106\]](#) returns the same value as the [ASCII\(\) \[1101\]](#) function.

```
mysql> SELECT ORD('2');
-> 50
```

- [POSITION\(*substr* IN *str*\) \[1107\]](#)

[POSITION\(*substr* IN *str*\) \[1107\]](#) is a synonym for [LOCATE\(*substr*,*str*\) \[1105\]](#).

- [QUOTE\(*str*\) \[1107\]](#)

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of backslash (“\”), single quote (“'”), ASCII `NUL`, and Control+Z preceded by a backslash. If the argument is `NULL`, the return value is the word “NULL” without enclosing single quotation marks.

```
mysql> SELECT QUOTE('Don\t!');
-> 'Don\t!'
mysql> SELECT QUOTE(NULL);
-> NULL
```

For comparison, see the quoting rules for literal strings and within the C API in [Section 9.1.1, “String Literals”](#), and [Section 22.8.3.53, “mysql_real_escape_string\(\)”](#).

- [REPEAT\(*str*,*count*\) \[1107\]](#)

Returns a string consisting of the string *str* repeated *count* times. If *count* is less than 1, returns an empty string. Returns `NULL` if *str* or *count* are `NULL`.

```
mysql> SELECT REPEAT('MySQL', 3);
-> 'MySQLMySQLMySQL'
```

- [REPLACE\(*str*,*from_str*,*to_str*\) \[1107\]](#)

Returns the string *str* with all occurrences of the string *from_str* replaced by the string *to_str*. [REPLACE\(\) \[1107\]](#) performs a case-sensitive match when searching for *from_str*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```

This function is multi-byte safe.

- [REVERSE\(*str*\) \[1107\]](#)

Returns the string *str* with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
-> 'cba'
```

This function is multi-byte safe.

- [RIGHT\(*str*,*len*\) \[1107\]](#)

Returns the rightmost *len* characters from the string *str*, or `NULL` if any argument is `NULL`.

```
mysql> SELECT RIGHT('foobarbar', 4);
-> 'rbar'
```

This function is multi-byte safe.

- [RPAD\(*str*,*len*,*padstr*\) \[1107\]](#)

Returns the string *str*, right-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.

```
mysql> SELECT RPAD('hi',5,'?');
-> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
-> 'h'
```

This function is multi-byte safe.

- [RTRIM\(*str*\) \[1108\]](#)

Returns the string *str* with trailing space characters removed.

```
mysql> SELECT RTRIM('barbar ');
-> 'barbar'
```

This function is multi-byte safe.

- [SOUNDEX\(*str*\) \[1108\]](#)

Returns a soundex string from *str*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the [SOUNDEX\(\) \[1108\]](#) function returns an arbitrarily long string. You can use [SUBSTRING\(\) \[1109\]](#) on the result to get a standard soundex string. All nonalphabetic characters in *str* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.



Important

When using [SOUNDEX\(\) \[1108\]](#), you should be aware of the following limitations:

- This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.
- This function is not guaranteed to provide consistent results with strings that use multi-byte character sets, including `utf-8`.

We hope to remove these limitations in a future release. See Bug #22638 for more information.

```
mysql> SELECT SOUNDEX('Hello');
-> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
-> 'Q36324'
```



Note

This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- [expr1 SOUNDS LIKE expr2 \[1108\]](#)

This is the same as `SOUNDEX(expr1) = SOUNDEX(expr2)` [\[1108\]](#).

- [SPACE\(*N*\) \[1108\]](#)

Returns a string consisting of *N* space characters.

```
mysql> SELECT SPACE(6);
-> '      '
```

- `SUBSTR(str,pos)` [1109], `SUBSTR(str FROM pos)` [1109], `SUBSTR(str,pos,len)` [1109], `SUBSTR(str FROM pos FOR len)` [1109]

`SUBSTR()` [1109] is a synonym for `SUBSTRING()` [1109].

- `SUBSTRING(str,pos)` [1109], `SUBSTRING(str FROM pos)` [1109], `SUBSTRING(str,pos,len)` [1109], `SUBSTRING(str FROM pos FOR len)` [1109]

The forms without a *len* argument return a substring from string *str* starting at position *pos*. The forms with a *len* argument return a substring *len* characters long from string *str*, starting at position *pos*. The forms that use `FROM` are standard SQL syntax. It is also possible to use a negative value for *pos*. In this case, the beginning of the substring is *pos* characters from the end of the string, rather than the beginning. A negative value may be used for *pos* in any of the forms of this function.

For all forms of `SUBSTRING()` [1109], the position of the first character in the string from which the substring is to be extracted is reckoned as 1.

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
mysql> SELECT SUBSTRING('Sakila', -3);
-> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
-> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
-> 'ki'
```

This function is multi-byte safe.

If *len* is less than 1, the result is the empty string.

- `SUBSTRING_INDEX(str,delim,count)` [1109]

Returns the substring from string *str* before *count* occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned. `SUBSTRING_INDEX()` [1109] performs a case-sensitive match when searching for *delim*.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
-> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
-> 'mysql.com'
```

This function is multi-byte safe.

- `TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)` [1109], `TRIM([remstr FROM] str)` [1109]

Returns the string *str* with all *remstr* prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. *remstr* is optional and, if not specified, spaces are removed.

```
mysql> SELECT TRIM(' bar ');
-> 'bar'
```

```
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxx');
-> 'barx'
```

This function is multi-byte safe.

- [UCASE\(*str*\) \[1110\]](#)

[UCASE\(\)](#) [1110] is a synonym for [UPPER\(\)](#) [1110].

- [UNHEX\(*str*\) \[1110\]](#)

For a string argument *str*, [UNHEX\(*str*\) \[1110\]](#) performs the inverse operation of [HEX\(*str*\) \[1104\]](#). That is, it interprets each pair of characters in the argument as a hexadecimal number and converts it to the character represented by the number. The return value is a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
-> 'MySQL'
mysql> SELECT 0x4D7953514C;
-> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
-> 'string'
mysql> SELECT HEX(UNHEX('1267'));
-> '1267'
```

The characters in the argument string must be legal hexadecimal digits: '0' .. '9', 'A' .. 'F', 'a' .. 'f'. If the argument contains any nonhexadecimal digits, the result is [NULL](#):

```
mysql> SELECT UNHEX('GG');
+-----+
| UNHEX('GG') |
+-----+
| NULL        |
+-----+
```

A [NULL](#) result can occur if the argument to [UNHEX\(\)](#) [1110] is a [BINARY](#) column, because values are padded with 0x00 bytes when stored but those bytes are not stripped on retrieval. For example, '41' is stored into a [CHAR\(3\)](#) column as '41 ' and retrieved as '41' (with the trailing pad space stripped), so [UNHEX\(\)](#) [1110] for the column value returns 'A'. By contrast '41' is stored into a [BINARY\(3\)](#) column as '41\0' and retrieved as '41\0' (with the trailing pad 0x00 byte not stripped). '\0' is not a legal hexadecimal digit, so [UNHEX\(\)](#) [1110] for the column value returns [NULL](#).

For a numeric argument *N*, the inverse of [HEX\(*N*\) \[1104\]](#) is not performed by [UNHEX\(\)](#) [1110]. Use [CONV\(HEX\(*N*\), 16, 10\) \[1125\]](#) instead. See the description of [HEX\(\)](#) [1104].

- [UPPER\(*str*\) \[1110\]](#)

Returns the string *str* with all characters changed to uppercase according to the current character set mapping. The default is [latin1](#) (cp1252 West European).

```
mysql> SELECT UPPER('Hej');
-> 'HEJ'
```

See the description of [LOWER\(\)](#) [1105] for information that also applies to [UPPER\(\)](#) [1110], such as information about how to perform lettercase conversion of binary strings ([BINARY](#), [VARBINARY](#), [BLOB](#)) for which these functions are ineffective.

This function is multi-byte safe.

12.5.1. String Comparison Functions

Table 12.8. String Comparison Operators

Name	Description
<code>LIKE</code> [1111]	Simple pattern matching
<code>NOT LIKE</code> [1113]	Negation of simple pattern matching
<code>STRCMP()</code> [1113]	Compare two strings

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

- `expr LIKE pat [ESCAPE 'escape_char']` [1111]

Pattern matching using SQL simple regular expression comparison. Returns 1 (TRUE) or 0 (FALSE). If either `expr` or `pat` is NULL, the result is NULL.

The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

Per the SQL standard, `LIKE` [1111] performs matching on a per-character basis, thus it can produce results different from the `=` [1089] comparison operator:

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----+
|                                     0 |
+-----+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-----+
|                                     1 |
+-----+
```

In particular, trailing spaces are significant, which is not true for `CHAR` or `VARCHAR` comparisons performed with the `=` [1089] operator:

```
mysql> SELECT 'a' = 'a ', 'a' LIKE 'a ';
+-----+-----+
| 'a' = 'a ' | 'a' LIKE 'a ' |
+-----+-----+
|          1 |              0 |
+-----+-----+
1 row in set (0.00 sec)
```

With `LIKE` [1111] you can use the following two wildcard characters in the pattern.

Character	Description
<code>%</code>	Matches any number of characters, even zero characters
<code>_</code>	Matches exactly one character

```
mysql> SELECT 'David!' LIKE 'David_';
-> 1
mysql> SELECT 'David!' LIKE '%D%v%';
```

```
-> 1
```

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the `ESCAPE` character, “\” is assumed.

String	Description
<code>\%</code>	Matches one “%” character
<code>_</code>	Matches one “_” character

```
mysql> SELECT 'David!' LIKE 'David\_';
      -> 0
mysql> SELECT 'David_' LIKE 'David\_';
      -> 1
```

To specify a different escape character, use the `ESCAPE` clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
      -> 1
```

The escape sequence should be empty or one character long. The expression must evaluate as a constant at execution time. If the `NO_BACKSLASH_ESCAPES` [614] SQL mode is enabled, the sequence cannot be empty.

The following two statements illustrate that string comparisons are not case sensitive unless one of the operands is a binary string:

```
mysql> SELECT 'abc' LIKE 'ABC';
      -> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
      -> 0
```

In MySQL, `LIKE` [1111] is permitted on numeric expressions. (This is an extension to the standard SQL `LIKE` [1111].)

```
mysql> SELECT 10 LIKE '1%';
      -> 1
```



Note

Because MySQL uses C escape syntax in strings (for example, “\n” to represent a newline character), you must double any “\” that you use in `LIKE` [1111] strings. For example, to search for “\n”, specify it as “\\n”. To search for “\”, specify it as “\\””; this is because the backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against.

Exception: At the end of the pattern string, backslash can be specified as “\”. At the end of the string, backslash stands for itself because there is nothing following to escape. Suppose that a table contains the following values:

```
mysql> SELECT filename FROM t1;
+-----+
| filename |
+-----+
| C:       |
| C:\     |
| C:\Programs |
| C:\Programs\ |
+-----+
```

To test for values that end with backslash, you can match the values using either of the following patterns:

```
mysql> SELECT filename, filename LIKE '%\\' FROM t1;
+-----+-----+
| filename | filename LIKE '%\\' |
+-----+-----+
| C:       |                      0 |
| C:\      |                      1 |
| C:\Programs |                  0 |
| C:\Programs\ |                  1 |
+-----+-----+

mysql> SELECT filename, filename LIKE '%\\\' FROM t1;
+-----+-----+
| filename | filename LIKE '%\\\' |
+-----+-----+
| C:       |                      0 |
| C:\      |                      1 |
| C:\Programs |                  0 |
| C:\Programs\ |                  1 |
+-----+-----+
```

- `expr NOT LIKE pat [ESCAPE 'escape_char']` [1113]

This is the same as `NOT (expr LIKE pat [ESCAPE 'escape_char'])`.



Note

Aggregate queries involving `NOT LIKE` [1113] comparisons with columns containing `NULL` may yield unexpected results. For example, consider the following table and data:

```
CREATE TABLE foo (bar VARCHAR(10));
INSERT INTO foo VALUES (NULL), (NULL);
```

The query `SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%'` returns 0. You might assume that `SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%'` would return 2. However, this is not the case: The second query returns 0. This is because `NULL NOT LIKE expr` always returns `NULL`, regardless of the value of `expr`. The same is true for aggregate queries involving `NULL` and comparisons using `NOT RLIKE` [1114] or `NOT REGEXP` [1114]. In such cases, you must test explicitly for `NOT NULL` using `OR` [1095] (and not `AND` [1094]), as shown here:

```
SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
```

- `STRCMP(expr1,expr2)` [1113]

`STRCMP()` [1113] returns 0 if the strings are the same, -1 if the first argument is smaller than the second according to the current sort order, and 1 otherwise.

```
mysql> SELECT STRCMP('text', 'text2');
-> -1
mysql> SELECT STRCMP('text2', 'text');
-> 1
mysql> SELECT STRCMP('text', 'text');
-> 0
```

`STRCMP()` [1113] performs the comparison using the collation of the arguments.

```
mysql> SET @s1 = _latin1 'x' COLLATE latin1_general_ci;
mysql> SET @s2 = _latin1 'X' COLLATE latin1_general_ci;
mysql> SET @s3 = _latin1 'x' COLLATE latin1_general_cs;
mysql> SET @s4 = _latin1 'X' COLLATE latin1_general_cs;
mysql> SELECT STRCMP(@s1, @s2), STRCMP(@s3, @s4);
+-----+-----+
| STRCMP(@s1, @s2) | STRCMP(@s3, @s4) |
+-----+-----+
|                0 |                1 |
+-----+-----+
```

If the collations are incompatible, one of the arguments must be converted to be compatible with the other. See [Section 10.1.7.5, “Collation of Expressions”](#).

```
mysql> SELECT STRCMP(@s1, @s3);
ERROR 1267 (HY000) at line 10: Illegal mix of collations (latin1_general_ci,IMPLICIT) and (latin1_general_cs,IMPLICIT) for operation 'AND'
mysql> SELECT STRCMP(@s1, @s3 COLLATE latin1_general_ci);
+-----+
| STRCMP(@s1, @s3 COLLATE latin1_general_ci) |
+-----+
|                                           0 |
+-----+
```

12.5.2. Regular Expressions

Table 12.9. String Regular Expression Operators

Name	Description
<code>NOT REGEXP</code> [1114]	Negation of REGEXP
<code>REGEXP</code> [1114]	Pattern matching using regular expressions
<code>RLIKE</code> [1114]	Synonym for REGEXP

A regular expression is a powerful way of specifying a pattern for a complex search.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. MySQL uses the extended version to support pattern-matching operations performed with the `REGEXP` [1114] operator in SQL statements.

This section summarizes, with examples, the special characters and constructs that can be used in MySQL for `REGEXP` [1114] operations. It does not contain all the details that can be found in Henry Spencer's `regex(7)` manual page. That manual page is included in MySQL source distributions, in the `regex.7` file under the `regex` directory. See also [Section 3.3.4.7, “Pattern Matching”](#).

- `expr NOT REGEXP pat` [1114], `expr NOT RLIKE pat` [1114]

This is the same as `NOT (expr REGEXP pat)`.

- `expr REGEXP pat` [1114], `expr RLIKE pat` [1114]

Performs a pattern match of a string expression `expr` against a pattern `pat`. The pattern can be an extended regular expression. The syntax for regular expressions is discussed in [Section 12.5.2, “Regular Expressions”](#). Returns `1` if `expr` matches `pat`; otherwise it returns `0`. If either `expr` or `pat` is `NULL`, the result is `NULL`. `RLIKE` [1114] is a synonym for `REGEXP` [1114], provided for `mSQL` compatibility.

The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

**Note**

Because MySQL uses the C escape syntax in strings (for example, “\n” to represent the newline character), you must double any “\” that you use in your [REGEXP \[1114\]](#) strings.

[REGEXP \[1114\]](#) is not case sensitive, except when used with binary strings.

```
mysql> SELECT 'Monty!' REGEXP '.*';
-> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*\\.\\*line';
-> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
-> 1 0
mysql> SELECT 'a' REGEXP '^[a-d]';
-> 1
```

[REGEXP \[1114\]](#) and [RLIKE \[1114\]](#) use the character set and collations of the arguments when deciding the type of a character and performing the comparison. If the arguments have different character sets or collations, coercibility rules apply as described in [Section 10.1.7.5, “Collation of Expressions”](#).

**Warning**

The [REGEXP \[1114\]](#) and [RLIKE \[1114\]](#) operators work in byte-wise fashion, so they are not multi-byte safe and may produce unexpected results with multi-byte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|word` matches either the string `hello` or the string `word`.

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression for the [REGEXP \[1114\]](#) operator may use any of the following special characters and constructs:

- `^`

Match the beginning of a string.

```
mysql> SELECT 'fo\ngo' REGEXP '^fo$';          -> 0
mysql> SELECT 'fofo' REGEXP '^fo';           -> 1
```

- `$`

Match the end of a string.

```
mysql> SELECT 'fo\ngo' REGEXP '^fo\ngo$';     -> 1
mysql> SELECT 'fo\ngo' REGEXP '^fo$';         -> 0
```

- `.`

Match any character (including carriage return and newline).

```
mysql> SELECT 'fofo' REGEXP '^f.*$';          -> 1
mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';     -> 1
```

- `a*`

Match any sequence of zero or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';          -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';       -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';          -> 1
```

- `a+`

Match any sequence of one or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';         -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';         -> 0
```

- `a?`

Match either zero or one `a` character.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';         -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';       -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';     -> 0
```

- `de|abc`

Match either of the sequences `de` or `abc`.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';        -> 1
mysql> SELECT 'axe' REGEXP 'pi|apa';       -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';       -> 1
mysql> SELECT 'apa' REGEXP '^ (pi|apa)$';  -> 1
mysql> SELECT 'pi' REGEXP '^ (pi|apa)$';   -> 1
mysql> SELECT 'pix' REGEXP '^ (pi|apa)$';  -> 0
```

- `(abc)*`

Match zero or more instances of the sequence `abc`.

```
mysql> SELECT 'pi' REGEXP '^ (pi)*$';      -> 1
mysql> SELECT 'pip' REGEXP '^ (pi)*$';     -> 0
mysql> SELECT 'pipi' REGEXP '^ (pi)*$';    -> 1
```

- `{1}, {2,3}`

`{n}` or `{m,n}` notation provides a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. `m` and `n` are integers.

- `a*`

Can be written as `a{0,}`.

- `a+`

Can be written as `a{1,}`.

- `a?`

Can be written as `a{0,1}`.

To be more precise, `a{n}` matches exactly `n` instances of `a`. `a{n,}` matches `n` or more instances of `a`. `a{m,n}` matches `m` through `n` instances of `a`, inclusive.

`m` and `n` must be in the range from 0 to `RE_DUP_MAX` (default 255), inclusive. If both `m` and `n` are given, `m` must be less than or equal to `n`.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';      -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';      -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e';    -> 1
```

- `[a-dX], [^a-dX]`

Matches any character that is (or is not, if `^` is used) either `a`, `b`, `c`, `d` or `X`. A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example, `[0-9]` matches any decimal digit. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. Any character that does not have a defined special meaning inside a `[]` pair matches only itself.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';        -> 1
mysql> SELECT 'aXbc' REGEXP '^a-dXYZ]';        -> 0
mysql> SELECT 'aXbc' REGEXP '^a-dXYZ]+$';      -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';    -> 0
mysql> SELECT 'gheis' REGEXP '^a-dXYZ]+$';     -> 1
mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$';   -> 0
```

- `[.characters.]`

Within a bracket expression (written using `[` and `]`), matches the sequence of characters of that collating element. `characters` is either a single character or a character name like `newline`. The following table lists the permissible character names.

The following table shows the permissible character names and the characters that they match. For characters given as numeric values, the values are represented in octal.

Name	Character	Name	Character
NUL	0	SOH	001
STX	002	ETX	003
EOT	004	ENQ	005
ACK	006	BEL	007
alert	007	BS	010
backspace	'\b'	HT	011
tab	'\t'	LF	012
newline	'\n'	VT	013
vertical-tab	'\v'	FF	014
form-feed	'\f'	CR	015
carriage-return	'\r'	SO	016
SI	017	DLE	020
DC1	021	DC2	022
DC3	023	DC4	024
NAK	025	SYN	026

Regular Expressions

Name	Character	Name	Character
ETB	027	CAN	030
EM	031	SUB	032
ESC	033	IS4	034
FS	034	IS3	035
GS	035	IS2	036
RS	036	IS1	037
US	037	space	' '
exclamation-mark	'!'	quotation-mark	'\"'
number-sign	'#'	dollar-sign	'\$'
percent-sign	'%'	ampersand	'&'
apostrophe	'\''	left-parenthesis	'('
right-parenthesis	')'	asterisk	'*'
plus-sign	'+'	comma	','
hyphen	'-'	hyphen-minus	'-'
period	'.'	full-stop	'.'
slash	'/'	solidus	'/'
zero	'0'	one	'1'
two	'2'	three	'3'
four	'4'	five	'5'
six	'6'	seven	'7'
eight	'8'	nine	'9'
colon	':'	semicolon	';'
less-than-sign	'<'	equals-sign	'='
greater-than-sign	'>'	question-mark	'?'
commercial-at	'@'	left-square-bracket	'['
backslash	'\\'	reverse-solidus	'\/'
right-square-bracket	']'	circumflex	'^'
circumflex-accent	'^'	underscore	'_'
low-line	'_'	grave-accent	'`'
left-brace	'{'	left-curly-bracket	'{'
vertical-line	' '	right-brace	'}'
right-curly-bracket	'}'	tilde	'~'
DEL	177		

```
mysql> SELECT '~' REGEXP '[[.~.]]';          -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';     -> 1
```

- [=character_class=]

Within a bracket expression (written using [and]), [=character_class=] represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if o and (+) are the members of an equivalence class, [[=o=]], [[=(+)=]], and [o(+)] are all synonymous. An equivalence class may not be used as an endpoint of a range.

- [:character_class:]

Within a bracket expression (written using [and]), [:character_class:] represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the ctype(3) manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

Character Class Name	Meaning
alnum	Alphanumeric characters
alpha	Alphabetic characters
blank	Whitespace characters
cntrl	Control characters
digit	Digit characters
graph	Graphic characters
lower	Lowercase alphabetic characters
print	Graphic or space characters
punct	Punctuation characters
space	Space, tab, newline, and carriage return
upper	Uppercase alphabetic characters
xdigit	Hexadecimal digit characters

```
mysql> SELECT 'justalnums' REGEXP '[[[:alnum:]]+];      -> 1
mysql> SELECT '!!!' REGEXP '[[[:alnum:]]+];          -> 0
```

- [[:<:]], [[:>:]]

These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the alnum class or an underscore (_).

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';  -> 1
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]'; -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string 1+2 that contains the special + character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT '1+2' REGEXP '1+2';      -> 0
mysql> SELECT '1+2' REGEXP '1\+2';     -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';    -> 1
```

12.6. Numeric Functions and Operators

Table 12.10. Numeric Functions and Operators

Name	Description
<code>ABS()</code> [1123]	Return the absolute value
<code>ACOS()</code> [1124]	Return the arc cosine
<code>ASIN()</code> [1124]	Return the arc sine
<code>ATAN2(), ATAN()</code> [1124]	Return the arc tangent of the two arguments
<code>ATAN()</code> [1124]	Return the arc tangent
<code>CEIL()</code> [1124]	Return the smallest integer value not less than the argument
<code>CEILING()</code> [1125]	Return the smallest integer value not less than the argument
<code>CONV()</code> [1125]	Convert numbers between different number bases
<code>COS()</code> [1125]	Return the cosine
<code>COT()</code> [1125]	Return the cotangent
<code>CRC32()</code> [1125]	Compute a cyclic redundancy check value
<code>DEGREES()</code> [1125]	Convert radians to degrees
<code>DIV</code> [1122]	Integer division
<code>/</code> [1122]	Division operator
<code>EXP()</code> [1126]	Raise to the power of
<code>FLOOR()</code> [1126]	Return the largest integer value not greater than the argument
<code>LN()</code> [1126]	Return the natural logarithm of the argument
<code>LOG10()</code> [1127]	Return the base-10 logarithm of the argument
<code>LOG2()</code> [1127]	Return the base-2 logarithm of the argument
<code>LOG()</code> [1126]	Return the natural logarithm of the first argument
<code>-</code> [1122]	Minus operator
<code>MOD()</code> [1127]	Return the remainder
<code>%</code> or <code>MOD</code> [1122]	Modulo operator
<code>PI()</code> [1127]	Return the value of pi
<code>+</code> [1121]	Addition operator
<code>POW()</code> [1128]	Return the argument raised to the specified power
<code>POWER()</code> [1128]	Return the argument raised to the specified power
<code>RADIANS()</code> [1128]	Return argument converted to radians
<code>RAND()</code> [1128]	Return a random floating-point value
<code>ROUND()</code> [1129]	Round the argument
<code>SIGN()</code> [1130]	Return the sign of the argument
<code>SIN()</code> [1130]	Return the sine of the argument
<code>SQRT()</code> [1130]	Return the square root of the argument
<code>TAN()</code> [1131]	Return the tangent of the argument
<code>*</code> [1122]	Multiplication operator

Name	Description
<code>TRUNCATE()</code> [1131]	Truncate to specified number of decimal places
<code>-</code> [1122]	Change the sign of the argument

12.6.1. Arithmetic Operators

Table 12.11. Arithmetic Operators

Name	Description
<code>DIV</code> [1122]	Integer division
<code>/</code> [1122]	Division operator
<code>-</code> [1122]	Minus operator
<code>%</code> or <code>MOD</code> [1122]	Modulo operator
<code>+</code> [1121]	Addition operator
<code>*</code> [1122]	Multiplication operator
<code>-</code> [1122]	Change the sign of the argument

The usual arithmetic operators are available. The result is determined according to the following rules:

- In the case of `-` [1122], `+` [1121], and `*` [1122], the result is calculated with `BIGINT` (64-bit) precision if both operands are integers.
- If both operands are integers and any of them are unsigned, the result is an unsigned integer. For subtraction, if the `NO_UNSIGNED_SUBTRACTION` [614] SQL mode is enabled, the result is signed even if any operand is unsigned.
- If any of the operands of a `+` [1121], `-` [1122], `/` [1122], `*` [1122], `%` [1122] is a real or string value, the precision of the result is the precision of the operand with the maximum precision.
- In division performed with `/` [1122], the scale of the result when using two exact-value operands is the scale of the first operand plus the value of the `div_precision_increment` [497] system variable (which is 4 by default). For example, the result of the expression `5.05 / 0.014` has a scale of six decimal places (`360.714286`).

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, `(14620 / 9432456) / (24250 / 9432456)`, resolves first to `(0.0014) / (0.0026)`, with the final result having 8 decimal places (`0.60288653`).

Because of these rules and the way they are applied, care should be taken to ensure that components and subcomponents of a calculation use the appropriate level of precision. See [Section 12.10, “Cast Functions and Operators”](#).

For information about handling of overflow in numeric expression evaluation, see [Section 11.2.6, “Out-of-Range and Overflow Handling”](#).

Arithmetic operators apply to numbers. For other types of values, alternative operations may be available. For example, to add date values, use `DATE_ADD()` [1135]; see [Section 12.7, “Date and Time Functions”](#).

- `+` [1121]

Addition:

```
mysql> SELECT 3+5;
-> 8
```

- [- \[1122\]](#)

Subtraction:

```
mysql> SELECT 3-5;
-> -2
```

- [- \[1122\]](#)

Unary minus. This operator changes the sign of the operand.

```
mysql> SELECT - 2;
-> -2
```



Note

If this operator is used with a `BIGINT`, the return value is also a `BIGINT`. This means that you should avoid using `-` on integers that may have the value of -2^{63} .

- [* \[1122\]](#)

Multiplication:

```
mysql> SELECT 3*5;
-> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
-> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
-> out-of-range error
```

The last expression produces an error because the result of the integer multiplication exceeds the 64-bit range of `BIGINT` calculations. (See [Section 11.2, “Numeric Types”](#).)

- [/ \[1122\]](#)

Division:

```
mysql> SELECT 3/5;
-> 0.60
```

Division by zero produces a `NULL` result:

```
mysql> SELECT 102/(1-1);
-> NULL
```

A division is calculated with `BIGINT` arithmetic only if performed in a context where its result is converted to an integer.

- [DIV \[1122\]](#)

Integer division. Similar to `FLOOR()` [\[1126\]](#), but is safe with `BIGINT` values.

As of MySQL 5.5.3, if either operand has a noninteger type, the operands are converted to `DECIMAL` and divided using `DECIMAL` arithmetic before converting the result to `BIGINT`. If the result exceeds `BIGINT` range, an error occurs. Before MySQL 5.5.3, incorrect results may occur for noninteger operands that exceed `BIGINT` range.

```
mysql> SELECT 5 DIV 2;
-> 2
```

- [N % M \[1122\]](#), [N MOD M \[1122\]](#)

Modulo operation. Returns the remainder of N divided by M . For more information, see the description for the `MOD()` [1127] function in [Section 12.6.2, “Mathematical Functions”](#).

12.6.2. Mathematical Functions

Table 12.12. Mathematical Functions

Name	Description
<code>ABS()</code> [1123]	Return the absolute value
<code>ACOS()</code> [1124]	Return the arc cosine
<code>ASIN()</code> [1124]	Return the arc sine
<code>ATAN2()</code> , <code>ATAN()</code> [1124]	Return the arc tangent of the two arguments
<code>ATAN()</code> [1124]	Return the arc tangent
<code>CEIL()</code> [1124]	Return the smallest integer value not less than the argument
<code>CEILING()</code> [1125]	Return the smallest integer value not less than the argument
<code>CONV()</code> [1125]	Convert numbers between different number bases
<code>COS()</code> [1125]	Return the cosine
<code>COT()</code> [1125]	Return the cotangent
<code>CRC32()</code> [1125]	Compute a cyclic redundancy check value
<code>DEGREES()</code> [1125]	Convert radians to degrees
<code>EXP()</code> [1126]	Raise to the power of
<code>FLOOR()</code> [1126]	Return the largest integer value not greater than the argument
<code>LN()</code> [1126]	Return the natural logarithm of the argument
<code>LOG10()</code> [1127]	Return the base-10 logarithm of the argument
<code>LOG2()</code> [1127]	Return the base-2 logarithm of the argument
<code>LOG()</code> [1126]	Return the natural logarithm of the first argument
<code>MOD()</code> [1127]	Return the remainder
<code>PI()</code> [1127]	Return the value of pi
<code>POW()</code> [1128]	Return the argument raised to the specified power
<code>POWER()</code> [1128]	Return the argument raised to the specified power
<code>RADIANS()</code> [1128]	Return argument converted to radians
<code>RAND()</code> [1128]	Return a random floating-point value
<code>ROUND()</code> [1129]	Round the argument
<code>SIGN()</code> [1130]	Return the sign of the argument
<code>SIN()</code> [1130]	Return the sine of the argument
<code>SQRT()</code> [1130]	Return the square root of the argument
<code>TAN()</code> [1131]	Return the tangent of the argument
<code>TRUNCATE()</code> [1131]	Truncate to specified number of decimal places

All mathematical functions return `NULL` in the event of an error.

- `ABS(X)` [1123]

Returns the absolute value of X .

```
mysql> SELECT ABS(2);
-> 2
mysql> SELECT ABS(-32);
-> 32
```

This function is safe to use with `BIGINT` values.

- [ACOS\(\$X\$ \) \[1124\]](#)

Returns the arc cosine of X , that is, the value whose cosine is X . Returns `NULL` if X is not in the range -1 to 1 .

```
mysql> SELECT ACOS(1);
-> 0
mysql> SELECT ACOS(1.0001);
-> NULL
mysql> SELECT ACOS(0);
-> 1.5707963267949
```

- [ASIN\(\$X\$ \) \[1124\]](#)

Returns the arc sine of X , that is, the value whose sine is X . Returns `NULL` if X is not in the range -1 to 1 .

```
mysql> SELECT ASIN(0.2);
-> 0.20135792079033
mysql> SELECT ASIN('foo');
```

ASIN('foo')
0

```
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
```

Level	Code	Message
Warning	1292	Truncated incorrect DOUBLE value: 'foo'

- [ATAN\(\$X\$ \) \[1124\]](#)

Returns the arc tangent of X , that is, the value whose tangent is X .

```
mysql> SELECT ATAN(2);
-> 1.1071487177941
mysql> SELECT ATAN(-2);
-> -1.1071487177941
```

- [ATAN\(\$Y, X\$ \) \[1124\]](#), [ATAN2\(\$Y, X\$ \) \[1124\]](#)

Returns the arc tangent of the two variables X and Y . It is similar to calculating the arc tangent of Y / X , except that the signs of both arguments are used to determine the quadrant of the result.

```
mysql> SELECT ATAN(-2,2);
-> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
-> 1.5707963267949
```

- [CEIL\(\$X\$ \) \[1124\]](#)

`CEIL()` [1124] is a synonym for `CEILING()` [1125].

- `CEILING(X)` [1125]

Returns the smallest integer value not less than X .

```
mysql> SELECT CEILING(1.23);
-> 2
mysql> SELECT CEILING(-1.23);
-> -1
```

For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `CONV(N, from_base, to_base)` [1125]

Converts numbers between different number bases. Returns a string representation of the number N , converted from base `from_base` to base `to_base`. Returns `NULL` if any argument is `NULL`. The argument N is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If `to_base` is a negative number, N is regarded as a signed number. Otherwise, N is treated as unsigned. `CONV()` [1125] works with 64-bit precision.

```
mysql> SELECT CONV('a',16,2);
-> '1010'
mysql> SELECT CONV('6E',18,8);
-> '172'
mysql> SELECT CONV(-17,10,-18);
-> '-H'
mysql> SELECT CONV(10+'10'+ '10'+0xa,10,10);
-> '40'
```

- `COS(X)` [1125]

Returns the cosine of X , where X is given in radians.

```
mysql> SELECT COS(PI());
-> -1
```

- `COT(X)` [1125]

Returns the cotangent of X .

```
mysql> SELECT COT(12);
-> -1.5726734063977
mysql> SELECT COT(0);
-> NULL
```

- `CRC32(expr)` [1125]

Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is `NULL` if the argument is `NULL`. The argument is expected to be a string and (if possible) is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');
-> 3259397556
mysql> SELECT CRC32('mysql');
-> 2501908538
```

- `DEGREES(X)` [1125]

Returns the argument X , converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());
```

```

-> 180
mysql> SELECT DEGREES(PI() / 2);
-> 90

```

- [EXP\(X\) \[1126\]](#)

Returns the value of e (the base of natural logarithms) raised to the power of X . The inverse of this function is [LOG\(\)](#) [1126] (using a single argument only) or [LN\(\)](#) [1126].

```

mysql> SELECT EXP(2);
-> 7.3890560989307
mysql> SELECT EXP(-2);
-> 0.13533528323661
mysql> SELECT EXP(0);
-> 1

```

- [FLOOR\(X\) \[1126\]](#)

Returns the largest integer value not greater than X .

```

mysql> SELECT FLOOR(1.23);
-> 1
mysql> SELECT FLOOR(-1.23);
-> -2

```

For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- [FORMAT\(X,D\) \[1103\]](#)

Formats the number X to a format like ' $\#,###,###.##$ ', rounded to D decimal places, and returns the result as a string. For details, see [Section 12.5, "String Functions"](#).

- [HEX\(N_or_S\) \[1104\]](#)

This function can be used to obtain a hexadecimal representation of a decimal number or a string; the manner in which it does so varies according to the argument's type. See this function's description in [Section 12.5, "String Functions"](#), for details.

- [LN\(X\) \[1126\]](#)

Returns the natural logarithm of X ; that is, the base- e logarithm of X . If X is less than or equal to 0, then `NULL` is returned.

```

mysql> SELECT LN(2);
-> 0.69314718055995
mysql> SELECT LN(-2);
-> NULL

```

This function is synonymous with [LOG\(X\) \[1126\]](#). The inverse of this function is the [EXP\(\)](#) [1126] function.

- [LOG\(X\) \[1126\]](#), [LOG\(B,X\) \[1126\]](#)

If called with one parameter, this function returns the natural logarithm of X . If X is less than or equal to 0, then `NULL` is returned.

The inverse of this function (when called with a single argument) is the [EXP\(\)](#) [1126] function.

```

mysql> SELECT LOG(2);
-> 0.69314718055995
mysql> SELECT LOG(-2);

```

```
-> NULL
```

If called with two parameters, this function returns the logarithm of X to the base B . If X is less than or equal to 0, or if B is less than or equal to 1, then `NULL` is returned.

```
mysql> SELECT LOG(2,65536);
-> 16
mysql> SELECT LOG(10,100);
-> 2
mysql> SELECT LOG(1,100);
-> NULL
```

`LOG(B, X)` [1126] is equivalent to `LOG(X) / LOG(B)` [1126].

- `LOG2(X)` [1127]

Returns the base-2 logarithm of X .

```
mysql> SELECT LOG2(65536);
-> 16
mysql> SELECT LOG2(-100);
-> NULL
```

`LOG2()` [1127] is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression `LOG(X) / LOG(2)` [1126].

- `LOG10(X)` [1127]

Returns the base-10 logarithm of X .

```
mysql> SELECT LOG10(2);
-> 0.30102999566398
mysql> SELECT LOG10(100);
-> 2
mysql> SELECT LOG10(-100);
-> NULL
```

`LOG10(X)` [1127] is equivalent to `LOG(10, X)` [1126].

- `MOD(N, M)` [1127], `N % M` [1122], `N MOD M` [1122]

Modulo operation. Returns the remainder of N divided by M .

```
mysql> SELECT MOD(234, 10);
-> 4
mysql> SELECT 253 % 7;
-> 1
mysql> SELECT MOD(29, 9);
-> 2
mysql> SELECT 29 MOD 9;
-> 2
```

This function is safe to use with `BIGINT` values.

`MOD()` [1127] also works on values that have a fractional part and returns the exact remainder after division:

```
mysql> SELECT MOD(34.5, 3);
-> 1.5
```

`MOD(N, 0)` [1127] returns `NULL`.

- `PI()` [1127]

Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

```
mysql> SELECT PI();
-> 3.141593
mysql> SELECT PI()+0.00000000000000000000;
-> 3.141592653589793116
```

- [POW\(X,Y\) \[1128\]](#)

Returns the value of X raised to the power of Y .

```
mysql> SELECT POW(2,2);
-> 4
mysql> SELECT POW(2,-2);
-> 0.25
```

- [POWER\(X,Y\) \[1128\]](#)

This is a synonym for [POW\(\) \[1128\]](#).

- [RADIANS\(X\) \[1128\]](#)

Returns the argument X , converted from degrees to radians. (Note that π radians equals 180 degrees.)

```
mysql> SELECT RADIANS(90);
-> 1.5707963267949
```

- [RAND\(\) \[1128\]](#), [RAND\(N\) \[1128\]](#)

Returns a random floating-point value v in the range $0 \leq v < 1.0$. If a constant integer argument N is specified, it is used as the seed value, which produces a repeatable sequence of column values. In the following example, note that the sequences of values produced by [RAND\(3\)](#) is the same both places where it occurs.

```
mysql> CREATE TABLE t (i INT);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT i, RAND() FROM t;
+-----+-----+
| i    | RAND() |
+-----+-----+
| 1    | 0.61914388706828 |
| 2    | 0.93845168309142 |
| 3    | 0.83482678498591 |
+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+-----+-----+
| i    | RAND(3) |
+-----+-----+
| 1    | 0.90576975597606 |
| 2    | 0.37307905813035 |
| 3    | 0.14808605345719 |
+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> SELECT i, RAND() FROM t;
+-----+-----+
| i     | RAND() |
+-----+-----+
| 1     | 0.35877890638893 |
| 2     | 0.28941420772058 |
| 3     | 0.37073435016976 |
+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+-----+-----+
| i     | RAND(3) |
+-----+-----+
| 1     | 0.90576975597606 |
| 2     | 0.37307905813035 |
| 3     | 0.14808605345719 |
+-----+-----+
3 rows in set (0.01 sec)
```

With a constant initializer, the seed is initialized once when the statement is compiled, prior to execution. If a nonconstant initializer (such as a column name) is used as the argument, the seed is initialized with the value for each invocation of `RAND()` [1128]. (One implication of this is that for equal argument values, `RAND()` [1128] will return the same value each time.)

To obtain a random integer R in the range $i \leq R < j$, use the expression `FLOOR(i + RAND() * (j [1126] - i))`. For example, to obtain a random integer in the range the range $7 \leq R < 12$, you could use the following statement:

```
SELECT FLOOR(7 + (RAND() * 5));
```

`RAND()` [1128] in a `WHERE` clause is re-evaluated every time the `WHERE` is executed.

You cannot use a column with `RAND()` [1128] values in an `ORDER BY` clause, because `ORDER BY` would evaluate the column multiple times. However, you can retrieve rows in random order like this:

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

`ORDER BY RAND()` combined with `LIMIT` is useful for selecting a random sample from a set of rows:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d -> ORDER BY RAND() LIMIT 1000;
```

`RAND()` [1128] is not meant to be a perfect random generator. It is a fast way to generate random numbers on demand that is portable between platforms for the same MySQL version.

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.2, a warning is logged if you use this function when `binlog_format` [1899] is set to `STATEMENT`. (Bug #49222)

- `ROUND(X)` [1129], `ROUND(X,D)` [1129]

Rounds the argument X to D decimal places. The rounding algorithm depends on the data type of X . D defaults to 0 if not specified. D can be negative to cause D digits left of the decimal point of the value X to become zero.

```
mysql> SELECT ROUND(-1.23);
-> -1
mysql> SELECT ROUND(-1.58);
-> -2
mysql> SELECT ROUND(1.58);
-> 2
mysql> SELECT ROUND(1.298, 1);
-> 1.3
```

```
mysql> SELECT ROUND(1.298, 0);
-> 1
mysql> SELECT ROUND(23.298, -1);
-> 20
```

The return type is the same type as that of the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places):

```
mysql> SELECT ROUND(150.000,2), ROUND(150,2);
+-----+-----+
| ROUND(150.000,2) | ROUND(150,2) |
+-----+-----+
|          150.00 |          150 |
+-----+-----+
```

`ROUND()` [1129] uses the following rules depending on the type of the first argument:

- For exact-value numbers, `ROUND()` [1129] uses the “round half away from zero” or “round toward nearest” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` [1129] uses the “round to nearest even” rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3          | 2            |
+-----+-----+
```

For more information, see [Section 12.18, “Precision Math”](#).

- `SIGN(X)` [1130]

Returns the sign of the argument as `-1`, `0`, or `1`, depending on whether `X` is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);
-> -1
mysql> SELECT SIGN(0);
-> 0
mysql> SELECT SIGN(234);
-> 1
```

- `SIN(X)` [1130]

Returns the sine of `X`, where `X` is given in radians.

```
mysql> SELECT SIN(PI());
-> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
-> 0
```

- `SQRT(X)` [1130]

Returns the square root of a nonnegative number `X`.

```
mysql> SELECT SQRT(4);
-> 2
mysql> SELECT SQRT(20);
-> 4.4721359549996
mysql> SELECT SQRT(-16);
-> NULL
```

- [TAN\(*X*\) \[1131\]](#)

Returns the tangent of *X*, where *X* is given in radians.

```
mysql> SELECT TAN(PI());
-> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
-> 1.5574077246549
```

- [TRUNCATE\(*X*,*D*\) \[1131\]](#)

Returns the number *X*, truncated to *D* decimal places. If *D* is 0, the result has no decimal point or fractional part. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
-> 1.2
mysql> SELECT TRUNCATE(1.999,1);
-> 1.9
mysql> SELECT TRUNCATE(1.999,0);
-> 1
mysql> SELECT TRUNCATE(-1.999,1);
-> -1.9
mysql> SELECT TRUNCATE(122,-2);
-> 100
mysql> SELECT TRUNCATE(10.28*100,0);
-> 1028
```

All numbers are rounded toward zero.

12.7. Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See [Section 11.3, “Date and Time Types”](#), for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Table 12.13. Date/Time Functions

Name	Description
ADDDATE() [1134]	Add time values (intervals) to a date value
ADDTIME() [1134]	Add time
CONVERT_TZ() [1134]	Convert from one timezone to another
CURDATE() [1135]	Return the current date
CURRENT_DATE() , CURRENT_DATE [1135]	Synonyms for CURDATE()
CURRENT_TIME() , CURRENT_TIME [1135]	Synonyms for CURTIME()
CURRENT_TIMESTAMP() , CURRENT_TIMESTAMP [1135]	Synonyms for NOW()
CURTIME() [1135]	Return the current time

Name	Description
DATE_ADD() [1135]	Add time values (intervals) to a date value
DATE_FORMAT() [1138]	Format date as specified
DATE_SUB() [1139]	Subtract a time value (interval) from a date
DATE() [1135]	Extract the date part of a date or datetime expression
DATEDIFF() [1135]	Subtract two dates
DAY() [1139]	Synonym for DAYOFMONTH()
DAYNAME() [1140]	Return the name of the weekday
DAYOFMONTH() [1140]	Return the day of the month (0-31)
DAYOFWEEK() [1140]	Return the weekday index of the argument
DAYOFYEAR() [1140]	Return the day of the year (1-366)
EXTRACT() [1140]	Extract part of a date
FROM_DAYS() [1140]	Convert a day number to a date
FROM_UNIXTIME() [1140]	Format UNIX timestamp as a date
GET_FORMAT() [1141]	Return a date format string
HOUR() [1142]	Extract the hour
LAST_DAY [1142]	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME [1142]	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP() [1142]	Synonym for NOW()
MAKEDATE() [1142]	Create a date from the year and day of year
MAKETIME [1142]	MAKETIME()
MICROSECOND() [1142]	Return the microseconds from argument
MINUTE() [1142]	Return the minute from the argument
MONTH() [1143]	Return the month from the date passed
MONTHNAME() [1143]	Return the name of the month
NOW() [1143]	Return the current date and time
PERIOD_ADD() [1144]	Add a period to a year-month
PERIOD_DIFF() [1144]	Return the number of months between periods
QUARTER() [1144]	Return the quarter from a date argument
SEC_TO_TIME() [1144]	Converts seconds to 'HH:MM:SS' format
SECOND() [1144]	Return the second (0-59)
STR_TO_DATE() [1144]	Convert a string to a date
SUBDATE() [1145]	A synonym for DATE_SUB() when invoked with three arguments
SUBTIME() [1145]	Subtract times
SYSDATE() [1146]	Return the time at which the function executes
TIME_FORMAT() [1148]	Format as time
TIME_TO_SEC() [1148]	Return the argument converted to seconds

Name	Description
TIME() [1146]	Extract the time portion of the expression passed
TIMEDIFF() [1146]	Subtract time
TIMESTAMP() [1147]	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD() [1147]	Add an interval to a datetime expression
TIMESTAMPDIFF() [1147]	Subtract an interval from a datetime expression
TO_DAYS() [1148]	Return the date argument converted to days
TO_SECONDS() [1149]	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP() [1150]	Return a UNIX timestamp
UTC_DATE() [1150]	Return the current UTC date
UTC_TIME() [1150]	Return the current UTC time
UTC_TIMESTAMP() [1151]	Return the current UTC date and time
WEEK() [1151]	Return the week number
WEEKDAY() [1152]	Return the weekday index
WEEKOFYEAR() [1152]	Return the calendar week of the date (0-53)
YEAR() [1152]	Return the year
YEARWEEK() [1152]	Return the year and week

Here is an example that uses date functions. The following query selects all rows with a `date_col` value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
-> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

The query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as [NOW\(\) \[1143\]](#) within a single query always produce the same result. (For our purposes, a single query also includes a call to a stored program (stored routine, trigger, or event) and all subprograms called by that program.) This principle also applies to [CURDATE\(\) \[1135\]](#), [CURTIME\(\) \[1135\]](#), [UTC_DATE\(\) \[1150\]](#), [UTC_TIME\(\) \[1150\]](#), [UTC_TIMESTAMP\(\) \[1151\]](#), and to any of their synonyms.

The [CURRENT_TIMESTAMP\(\) \[1135\]](#), [CURRENT_TIME\(\) \[1135\]](#), [CURRENT_DATE\(\) \[1135\]](#), and [FROM_UNIXTIME\(\) \[1140\]](#) functions return values in the connection's current time zone, which is available as the value of the `time_zone` [567] system variable. In addition, [UNIX_TIMESTAMP\(\) \[1150\]](#) assumes that its argument is a datetime value in the current time zone. See [Section 10.6, "MySQL Server Time Zone Support"](#).

Some date functions can be used with "zero" dates or incomplete dates such as `'2001-11-00'`, whereas others cannot. Functions that extract parts of dates typically work with incomplete dates and thus can return 0 when you might otherwise expect a nonzero value. For example:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
-> 0, 0
```

Other functions expect complete dates and return `NULL` for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
-> NULL
mysql> SELECT DAYNAME('2006-05-00');
-> NULL
```



Note

From MySQL 5.5.16 to 5.5.20, a change in handling of a date-related assertion caused several functions to become more strict when passed a `DATE()` [1135] function value as their argument and reject incomplete dates with a day part of zero. These functions are affected: `CONVERT_TZ()` [1134], `DATE_ADD()` [1135], `DATE_SUB()` [1139], `DAYOFYEAR()` [1140], `LAST_DAY()` [1142], `TIMESTAMPDIFF()` [1147], `TO_DAYS()` [1148], `TO_SECONDS()` [1149], `WEEK()` [1151], `WEEKDAY()` [1152], `WEEKOFYEAR()` [1152], `YEARWEEK()` [1152]. Because this changes date-handling behavior in General Availability-status series MySQL 5.5, the change was reverted in 5.5.21.

- `ADDDATE(date,INTERVAL expr unit)` [1134], `ADDDATE(expr,days)` [1134]

When invoked with the `INTERVAL` form of the second argument, `ADDDATE()` [1134] is a synonym for `DATE_ADD()` [1135]. The related function `SUBDATE()` [1145] is a synonym for `DATE_SUB()` [1139]. For information on the `INTERVAL unit` argument, see the discussion for `DATE_ADD()` [1135].

```
mysql> SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
-> '2008-02-02'
mysql> SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
-> '2008-02-02'
```

When invoked with the `days` form of the second argument, MySQL treats it as an integer number of days to be added to `expr`.

```
mysql> SELECT ADDDATE('2008-01-02', 31);
-> '2008-02-02'
```

- `ADDTIME(expr1,expr2)` [1134]

`ADDTIME()` [1134] adds `expr2` to `expr1` and returns the result. `expr1` is a time or datetime expression, and `expr2` is a time expression.

```
mysql> SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
-> '2008-01-02 01:01:01.000001'
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
-> '03:00:01.999997'
```

- `CONVERT_TZ(dt,from_tz,to_tz)` [1134]

`CONVERT_TZ()` [1134] converts a datetime value `dt` from the time zone given by `from_tz` to the time zone given by `to_tz` and returns the resulting value. Time zones are specified as described in Section 10.6, “MySQL Server Time Zone Support”. This function returns `NULL` if the arguments are invalid.

If the value falls out of the supported range of the `TIMESTAMP` type when converted from `from_tz` to UTC, no conversion occurs. The `TIMESTAMP` range is described in Section 11.1.2, “Date and Time Type Overview”.

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
-> '2004-01-01 13:00:00'
```

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
-> '2004-01-01 22:00:00'
```

**Note**

To use named time zones such as 'MET' or 'Europe/Moscow', the time zone tables must be properly set up. See [Section 10.6, “MySQL Server Time Zone Support”](#), for instructions.

- [CURDATE\(\) \[1135\]](#)

Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
-> '2008-06-13'
mysql> SELECT CURDATE() + 0;
-> 20080613
```

- [CURRENT_DATE \[1135\]](#), [CURRENT_DATE\(\) \[1135\]](#)

[CURRENT_DATE \[1135\]](#) and [CURRENT_DATE\(\) \[1135\]](#) are synonyms for [CURDATE\(\) \[1135\]](#).

- [CURRENT_TIME \[1135\]](#), [CURRENT_TIME\(\) \[1135\]](#)

[CURRENT_TIME \[1135\]](#) and [CURRENT_TIME\(\) \[1135\]](#) are synonyms for [CURTIME\(\) \[1135\]](#).

- [CURRENT_TIMESTAMP \[1135\]](#), [CURRENT_TIMESTAMP\(\) \[1135\]](#)

[CURRENT_TIMESTAMP \[1135\]](#) and [CURRENT_TIMESTAMP\(\) \[1135\]](#) are synonyms for [NOW\(\) \[1143\]](#).

- [CURTIME\(\) \[1135\]](#)

Returns the current time as a value in 'HH:MM:SS' or HHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT CURTIME();
-> '23:50:26'
mysql> SELECT CURTIME() + 0;
-> 235026.000000
```

- [DATE\(expr\) \[1135\]](#)

Extracts the date part of the date or datetime expression *expr*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
-> '2003-12-31'
```

- [DATEDIFF\(expr1,expr2\) \[1135\]](#)

[DATEDIFF\(\) \[1135\]](#) returns *expr1* – *expr2* expressed as a value in days from one date to the other. *expr1* and *expr2* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
-> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
-> -31
```

- [DATE_ADD\(date,INTERVAL expr unit\) \[1135\]](#), [DATE_SUB\(date,INTERVAL expr unit\) \[1139\]](#)

These functions perform date arithmetic. The *date* argument specifies the starting date or datetime value. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a “-” for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted.

The `INTERVAL` keyword and the *unit* specifier are not case sensitive.

The following table shows the expected form of the *expr* argument for each *unit* value.

<i>unit</i> Value	Expected <i>expr</i> Format
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	' SECONDS . MICROSECONDS '
MINUTE_MICROSECOND	' MINUTES : SECONDS . MICROSECONDS '
MINUTE_SECOND	' MINUTES : SECONDS '
HOUR_MICROSECOND	' HOURS : MINUTES : SECONDS . MICROSECONDS '
HOUR_SECOND	' HOURS : MINUTES : SECONDS '
HOUR_MINUTE	' HOURS : MINUTES '
DAY_MICROSECOND	' DAYS HOURS : MINUTES : SECONDS . MICROSECONDS '
DAY_SECOND	' DAYS HOURS : MINUTES : SECONDS '
DAY_MINUTE	' DAYS HOURS : MINUTES '
DAY_HOUR	' DAYS HOURS '
YEAR_MONTH	' YEARS - MONTHS '

The return value depends on the arguments:

- `DATETIME` if the first argument is a `DATETIME` (or `TIMESTAMP`) value, or if the first argument is a `DATE` and the *unit* value uses `HOURS`, `MINUTES`, or `SECONDS`.
- String otherwise.

To ensure that the result is `DATETIME`, you can use `CAST ()` [1168] to convert the first argument to `DATETIME`.

MySQL permits any punctuation delimiter in the *expr* format. Those shown in the table are the suggested delimiters. If the *date* argument is a `DATE` value and your calculations involve only `YEAR`, `MONTH`, and `DAY` parts (that is, no time parts), the result is a `DATE` value. Otherwise, the result is a `DATETIME` value.

Date arithmetic also can be performed using `INTERVAL` together with the `+` [1121] or `-` [1122] operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

`INTERVAL expr unit` is permitted on either side of the `+` [1121] operator if the expression on the other side is a date or datetime value. For the `-` [1122] operator, `INTERVAL expr unit` is permitted only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```
mysql> SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;
-> '2009-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '2008-12-31';
-> '2009-01-01'
mysql> SELECT '2005-01-01' - INTERVAL 1 SECOND;
-> '2004-12-31 23:59:59'
mysql> SELECT DATE_ADD('2000-12-31 23:59:59',
-> INTERVAL 1 SECOND);
-> '2001-01-01 00:00:00'
mysql> SELECT DATE_ADD('2010-12-31 23:59:59',
-> INTERVAL 1 DAY);
-> '2011-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
-> INTERVAL '1:1' MINUTE_SECOND);
-> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2005-01-01 00:00:00',
-> INTERVAL '1 1:1:1' DAY_SECOND);
-> '2004-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
-> INTERVAL '-1 10' DAY_HOUR);
-> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
-> INTERVAL '1.999999' SECOND_MICROSECOND);
-> '1993-01-01 00:00:01.000001'
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the `unit` keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a `unit` of `DAY_SECOND`, the value of `expr` is expected to have days, hours, minutes, and seconds parts. If you specify a value like `'1:10'`, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `'1:10' DAY_SECOND` is interpreted in such a way that it is equivalent to `'1:10' MINUTE_SECOND`. This is analogous to the way that MySQL interprets `TIME` values as representing elapsed time rather than as a time of day.

Because `expr` is treated as a string, be careful if you specify a nonstring value with `INTERVAL`. For example, with an interval specifier of `HOURL_MINUTE`, `6/4` evaluates to `1.5000` and is treated as 1 hour, 5000 minutes:

```
mysql> SELECT 6/4;
-> 1.5000
mysql> SELECT DATE_ADD('2009-01-01', INTERVAL 6/4 HOUR_MINUTE);
-> '2009-01-04 12:20:00'
```

To ensure interpretation of the interval value as you expect, a `CAST()` [1168] operation may be used. To treat `6/4` as 1 hour, 5 minutes, cast it to a `DECIMAL` value with a single fractional digit:

```
mysql> SELECT CAST(6/4 AS DECIMAL(3,1));
-> 1.5
mysql> SELECT DATE_ADD('1970-01-01 12:00:00',
-> INTERVAL CAST(6/4 AS DECIMAL(3,1)) HOUR_MINUTE);
```

```
-> '1970-01-01 13:05:00'
```

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 DAY);
-> '2013-01-02'
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 HOUR);
-> '2013-01-01 01:00:00'
```

If you add `MONTH`, `YEAR_MONTH`, or `YEAR` and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('2009-01-30', INTERVAL 1 MONTH);
-> '2009-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as `'2006-07-00'` or badly malformed dates:

```
mysql> SELECT DATE_ADD('2006-07-00', INTERVAL 1 DAY);
-> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
-> NULL
```

- [DATE_FORMAT\(*date*, *format*\)](#) [1138]

Formats the *date* value according to the *format* string.

The following specifiers may be used in the *format* string. The “%” character is required before format specifier characters.

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)

Specifier	Description
<code>%S</code>	Seconds (00..59)
<code>%s</code>	Seconds (00..59)
<code>%T</code>	Time, 24-hour (hh:mm:ss)
<code>%U</code>	Week (00..53), where Sunday is the first day of the week
<code>%u</code>	Week (00..53), where Monday is the first day of the week
<code>%V</code>	Week (01..53), where Sunday is the first day of the week; used with <code>%X</code>
<code>%v</code>	Week (01..53), where Monday is the first day of the week; used with <code>%x</code>
<code>%W</code>	Weekday name (Sunday..Saturday)
<code>%w</code>	Day of the week (0=Sunday..6=Saturday)
<code>%X</code>	Year for the week where Sunday is the first day of the week, numeric, four digits; used with <code>%V</code>
<code>%x</code>	Year for the week, where Monday is the first day of the week, numeric, four digits; used with <code>%v</code>
<code>%Y</code>	Year, numeric, four digits
<code>%y</code>	Year, numeric (two digits)
<code>%%</code>	A literal “%” character
<code>%x</code>	<code>x</code> , for any “ <code>x</code> ” not listed above

Ranges for the month and day specifiers begin with zero due to the fact that MySQL permits the storing of incomplete dates such as '2014-00-00'.

The language used for day and month names and abbreviations is controlled by the value of the `lc_time_names` [512] system variable (Section 10.7, “MySQL Server Locale Support”).

`DATE_FORMAT()` [1138] returns a string with a character set and collation given by `character_set_connection` [487] and `collation_connection` [489] so that it can return month and weekday names containing non-ASCII characters.

```
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
-> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
-> '%D %y %a %d %m %b %j');
-> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
-> '%H %k %I %r %T %S %w');
-> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
-> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
-> '00'
```

- `DATE_SUB(date, INTERVAL expr unit)` [1139]

See the description for `DATE_ADD()` [1135].

- `DAY(date)` [1139]

`DAY()` [1139] is a synonym for `DAYOFMONTH()` [1140].

- `DAYNAME(date)` [1140]

Returns the name of the weekday for *date*. The language used for the name is controlled by the value of the `lc_time_names` [512] system variable (Section 10.7, “MySQL Server Locale Support”).

```
mysql> SELECT DAYNAME('2007-02-03');
-> 'Saturday'
```

- `DAYOFMONTH(date)` [1140]

Returns the day of the month for *date*, in the range 1 to 31, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero day part.

```
mysql> SELECT DAYOFMONTH('2007-02-03');
-> 3
```

- `DAYOFWEEK(date)` [1140]

Returns the weekday index for *date* (1 = Sunday, 2 = Monday, ..., 7 = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('2007-02-03');
-> 7
```

- `DAYOFYEAR(date)` [1140]

Returns the day of the year for *date*, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('2007-02-03');
-> 34
```

- `EXTRACT(unit FROM date)` [1140]

The `EXTRACT()` [1140] function uses the same kinds of unit specifiers as `DATE_ADD()` [1135] or `DATE_SUB()` [1139], but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '2009-07-02');
-> 2009
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
-> 200907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03');
-> 20102
mysql> SELECT EXTRACT(MICROSECOND
-> FROM '2003-01-02 10:30:00.000123');
-> 123
```

- `FROM_DAYS(N)` [1140]

Given a day number *N*, returns a `DATE` value.

```
mysql> SELECT FROM_DAYS(730669);
-> '2007-07-03'
```

Use `FROM_DAYS()` [1140] with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See Section 12.8, “What Calendar Is Used By MySQL?”.

- `FROM_UNIXTIME(unix_timestamp)` [1140],
`FROM_UNIXTIME(unix_timestamp,format)` [1140]

Returns a representation of the *unix_timestamp* argument as a value in 'YYYY-MM-DD HH:MM:SS' or 'YYYYMMDDHHMMSS.aaaaaa' format, depending on whether the function is used in a string or numeric

context. The value is expressed in the current time zone. `unix_timestamp` is an internal timestamp value such as is produced by the `UNIX_TIMESTAMP()` [1150] function.

If `format` is given, the result is formatted according to the `format` string, which is used the same way as listed in the entry for the `DATE_FORMAT()` [1138] function.

```
mysql> SELECT FROM_UNIXTIME(1196440219);
-> '2007-11-30 10:30:19'
mysql> SELECT FROM_UNIXTIME(1196440219) + 0;
-> 20071130103019.000000
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
->          '%Y %D %M %h:%i:%s %x');
-> '2007 30th November 10:30:59 2007'
```

Note: If you use `UNIX_TIMESTAMP()` [1150] and `FROM_UNIXTIME()` [1140] to convert between `TIMESTAMP` values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the `UNIX_TIMESTAMP()` [1150] function.

- `GET_FORMAT({DATE|TIME|DATETIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})` [1141]

Returns a format string. This function is useful in combination with the `DATE_FORMAT()` [1138] and the `STR_TO_DATE()` [1144] functions.

The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the `DATE_FORMAT()` [1138] function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
<code>GET_FORMAT(DATE, 'USA')</code> [1141]	'%m.%d.%Y'
<code>GET_FORMAT(DATE, 'JIS')</code> [1141]	'%Y-%m-%d'
<code>GET_FORMAT(DATE, 'ISO')</code> [1141]	'%Y-%m-%d'
<code>GET_FORMAT(DATE, 'EUR')</code> [1141]	'%d.%m.%Y'
<code>GET_FORMAT(DATE, 'INTERNAL')</code> [1141]	'%Y%m%d'
<code>GET_FORMAT(DATETIME, 'USA')</code> [1141]	'%Y-%m-%d %H.%i.%s'
<code>GET_FORMAT(DATETIME, 'JIS')</code> [1141]	'%Y-%m-%d %H:%i:%s'
<code>GET_FORMAT(DATETIME, 'ISO')</code> [1141]	'%Y-%m-%d %H:%i:%s'
<code>GET_FORMAT(DATETIME, 'EUR')</code> [1141]	'%Y-%m-%d %H.%i.%s'
<code>GET_FORMAT(DATETIME, 'INTERNAL')</code> [1141]	'%Y%m%d%H%i%s'
<code>GET_FORMAT(TIME, 'USA')</code> [1141]	'%h:%i:%s %p'
<code>GET_FORMAT(TIME, 'JIS')</code> [1141]	'%H:%i:%s'
<code>GET_FORMAT(TIME, 'ISO')</code> [1141]	'%H:%i:%s'
<code>GET_FORMAT(TIME, 'EUR')</code> [1141]	'%H.%i.%s'
<code>GET_FORMAT(TIME, 'INTERNAL')</code> [1141]	'%H%i%s'

`TIMESTAMP` can also be used as the first argument to `GET_FORMAT()` [1141], in which case the function returns the same values as for `DATETIME`.

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
-> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
-> '2003-10-31'
```

- [HOUR\(*time*\)](#) [1142]

Returns the hour for *time*. The range of the return value is 0 to 23 for time-of-day values. However, the range of `TIME` values actually is much larger, so `HOUR` can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');
-> 10
mysql> SELECT HOUR('272:59:59');
-> 272
```

- [LAST_DAY\(*date*\)](#) [1142]

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns `NULL` if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
-> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
-> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
-> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
-> NULL
```

- [LOCALTIME](#) [1142], [LOCALTIME\(\)](#) [1142]

`LOCALTIME` [1142] and `LOCALTIME()` [1142] are synonyms for `NOW()` [1143].

- [LOCALTIMESTAMP](#) [1142], [LOCALTIMESTAMP\(\)](#) [1142]

`LOCALTIMESTAMP` [1142] and `LOCALTIMESTAMP()` [1142] are synonyms for `NOW()` [1143].

- [MAKEDATE\(*year*,*dayofyear*\)](#) [1142]

Returns a date, given year and day-of-year values. *dayofyear* must be greater than 0 or the result is `NULL`.

```
mysql> SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
-> '2011-01-31', '2011-02-01'
mysql> SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
-> '2011-12-31', '2014-12-31'
mysql> SELECT MAKEDATE(2011,0);
-> NULL
```

- [MAKETIME\(*hour*,*minute*,*second*\)](#) [1142]

Returns a time value calculated from the *hour*, *minute*, and *second* arguments.

```
mysql> SELECT MAKETIME(12,15,30);
-> '12:15:30'
```

- [MICROSECOND\(*expr*\)](#) [1142]

Returns the microseconds from the time or datetime expression *expr* as a number in the range from 0 to 999999.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
-> 123456
mysql> SELECT MICROSECOND('2009-12-31 23:59:59.000010');
-> 10
```

- [MINUTE\(*time*\)](#) [1142]

Returns the minute for *time*, in the range 0 to 59.

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
-> 5
```

- [MONTH\(*date*\) \[1143\]](#)

Returns the month for *date*, in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part.

```
mysql> SELECT MONTH('2008-02-03');
-> 2
```

- [MONTHNAME\(*date*\) \[1143\]](#)

Returns the full name of the month for *date*. The language used for the name is controlled by the value of the `lc_time_names` [512] system variable (Section 10.7, “MySQL Server Locale Support”).

```
mysql> SELECT MONTHNAME('2008-02-03');
-> 'February'
```

- [NOW\(\) \[1143\]](#)

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT NOW();
-> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
-> 20071215235026.000000
```

[NOW\(\) \[1143\]](#) returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, [NOW\(\) \[1143\]](#) returns the time at which the function or triggering statement began to execute.) This differs from the behavior for [SYSDATE\(\) \[1146\]](#), which returns the exact time at which it executes.

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 |          0 | 2006-04-12 13:47:36 |
+-----+-----+-----+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 |          0 | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

In addition, the `SET TIMESTAMP` statement affects the value returned by [NOW\(\) \[1143\]](#) but not by [SYSDATE\(\) \[1146\]](#). This means that timestamp settings in the binary log have no effect on invocations of [SYSDATE\(\) \[1146\]](#). Setting the timestamp to a nonzero value causes each subsequent invocation of [NOW\(\) \[1143\]](#) to return that value. Setting the timestamp to zero cancels this effect so that [NOW\(\) \[1143\]](#) once again returns the current date and time.

See the description for [SYSDATE\(\) \[1146\]](#) for additional information about the differences between the two functions.

- [PERIOD_ADD\(*P*,*N*\)](#) [1144]

Adds *N* months to period *P* (in the format *YYMM* or *YYYYMM*). Returns a value in the format *YYYYMM*. Note that the period argument *P* is *not* a date value.

```
mysql> SELECT PERIOD_ADD(200801,2);
-> 200803
```

- [PERIOD_DIFF\(*P1*,*P2*\)](#) [1144]

Returns the number of months between periods *P1* and *P2*. *P1* and *P2* should be in the format *YYMM* or *YYYYMM*. Note that the period arguments *P1* and *P2* are *not* date values.

```
mysql> SELECT PERIOD_DIFF(200802,200703);
-> 11
```

- [QUARTER\(*date*\)](#) [1144]

Returns the quarter of the year for *date*, in the range 1 to 4.

```
mysql> SELECT QUARTER('2008-04-01');
-> 2
```

- [SECOND\(*time*\)](#) [1144]

Returns the second for *time*, in the range 0 to 59.

```
mysql> SELECT SECOND('10:05:03');
-> 3
```

- [SEC_TO_TIME\(*seconds*\)](#) [1144]

Returns the *seconds* argument, converted to hours, minutes, and seconds, as a *TIME* value. The range of the result is constrained to that of the *TIME* data type. A warning occurs if the argument corresponds to a value outside that range.

```
mysql> SELECT SEC_TO_TIME(2378);
-> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
-> 3938
```

- [STR_TO_DATE\(*str*,*format*\)](#) [1144]

This is the inverse of the [DATE_FORMAT\(\)](#) [1138] function. It takes a string *str* and a format string *format*. [STR_TO_DATE\(\)](#) [1144] returns a *DATETIME* value if the format string contains both date and time parts, or a *DATE* or *TIME* value if the string contains only date or time parts. If the date, time, or datetime value extracted from *str* is illegal, [STR_TO_DATE\(\)](#) [1144] returns *NULL* and produces a warning.

The server scans *str* attempting to match *format* to it. The format string can contain literal characters and format specifiers beginning with *%*. Literal characters in *format* must match literally in *str*. Format specifiers in *format* must match a date or time part in *str*. For the specifiers that can be used in *format*, see the [DATE_FORMAT\(\)](#) [1138] function description.

```
mysql> SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
-> '2013-05-01'
mysql> SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
-> '2013-05-01'
```

Scanning starts at the beginning of *str* and fails if *format* is found not to match. Extra characters at the end of *str* are ignored.

```
mysql> SELECT STR_TO_DATE('a09:30:17','a%h:%i:%s');
-> '09:30:17'
mysql> SELECT STR_TO_DATE('a09:30:17','%h:%i:%s');
-> NULL
mysql> SELECT STR_TO_DATE('09:30:17a','%h:%i:%s');
-> '09:30:17'
```

Unspecified date or time parts have a value of 0, so incompletely specified values in *str* produce a result with some or all parts set to 0:

```
mysql> SELECT STR_TO_DATE('abc','abc');
-> '0000-00-00'
mysql> SELECT STR_TO_DATE('9','%m');
-> '0000-09-00'
mysql> SELECT STR_TO_DATE('9','%s');
-> '00:00:09'
```

Range checking on the parts of date values is as described in [Section 11.3.1, “The DATE, DATETIME, and TIMESTAMP Types”](#). This means, for example, that “zero” dates or dates with part values of 0 are permitted unless the SQL mode is set to disallow such values.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
-> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
-> '2004-04-31'
```



Note

You cannot use format “%X%V” to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
-> '2004-10-18'
```

- [SUBDATE\(*date*, INTERVAL *expr* *unit*\)](#) [1145], [SUBDATE\(*expr*, *days*\)](#) [1145]

When invoked with the *INTERVAL* form of the second argument, [SUBDATE\(\)](#) [1145] is a synonym for [DATE_SUB\(\)](#) [1139]. For information on the *INTERVAL unit* argument, see the discussion for [DATE_ADD\(\)](#) [1135].

```
mysql> SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
-> '2007-12-02'
mysql> SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
-> '2007-12-02'
```

The second form enables the use of an integer value for *days*. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression *expr*.

```
mysql> SELECT SUBDATE('2008-01-02 12:00:00', 31);
-> '2007-12-02 12:00:00'
```

- [SUBTIME\(*expr1*, *expr2*\)](#) [1145]

[SUBTIME\(\)](#) [1145] returns *expr1* – *expr2* expressed as a value in the same format as *expr1*. *expr1* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT SUBTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
-> '2007-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
```

```
-> '-00:59:59.999999'
```

- [SYSDATE\(\) \[1146\]](#)

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context.

[SYSDATE\(\) \[1146\]](#) returns the time at which it executes. This differs from the behavior for [NOW\(\) \[1143\]](#), which returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, [NOW\(\) \[1143\]](#) returns the time at which the function or triggering statement began to execute.)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 |          0 | 2006-04-12 13:47:36 |
+-----+-----+-----+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 |          0 | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

In addition, the [SET TIMESTAMP](#) statement affects the value returned by [NOW\(\) \[1143\]](#) but not by [SYSDATE\(\) \[1146\]](#). This means that timestamp settings in the binary log have no effect on invocations of [SYSDATE\(\) \[1146\]](#).

Because [SYSDATE\(\) \[1146\]](#) can return different values even within the same statement, and is not affected by [SET TIMESTAMP](#), it is nondeterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging.

Alternatively, you can use the [--sysdate-is-now \[465\]](#) option to cause [SYSDATE\(\) \[1146\]](#) to be an alias for [NOW\(\) \[1143\]](#). This works if the option is used on both the master and the slave.

The nondeterministic nature of [SYSDATE\(\) \[1146\]](#) also means that indexes cannot be used for evaluating expressions that refer to it.

- [TIME\(expr\) \[1146\]](#)

Extracts the time part of the time or datetime expression *expr* and returns it as a string.

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.1, a warning is logged if you use this function when [binlog_format \[1899\]](#) is set to [STATEMENT](#). (Bug #47995)

```
mysql> SELECT TIME('2003-12-31 01:02:03');
-> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
-> '01:02:03.000123'
```

- [TIMEDIFF\(expr1,expr2\) \[1146\]](#)

[TIMEDIFF\(\) \[1146\]](#) returns *expr1* – *expr2* expressed as a time value. *expr1* and *expr2* are time or date-and-time expressions, but both must be of the same type.

The result returned by `TIMEDIFF()` is limited to the range allowed for `TIME` values. Alternatively, you can use either of the functions `TIMESTAMPDIFF()` [1147] and `UNIX_TIMESTAMP()` [1150], both of which return integers.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
-> '2000:01:01 00:00:00.000001');
-> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('2008-12-31 23:59:59.000001',
-> '2008-12-30 01:01:01.000002');
-> '46:58:57.999999'
```

- `TIMESTAMP(expr)` [1147], `TIMESTAMP(expr1,expr2)` [1147]

With a single argument, this function returns the date or datetime expression `expr` as a datetime value. With two arguments, it adds the time expression `expr2` to the date or datetime expression `expr1` and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
-> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
-> '2004-01-01 00:00:00'
```

- `TIMESTAMPADD(unit, interval, datetime_expr)` [1147]

Adds the integer expression `interval` to the date or datetime expression `datetime_expr`. The unit for `interval` is given by the `unit` argument, which should be one of the following values: `MICROSECOND` (microseconds), `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, or `YEAR`.

It is possible to use `FRAC_SECOND` in place of `MICROSECOND`, but `FRAC_SECOND` is deprecated. `FRAC_SECOND` was removed in MySQL 5.5.3.

The `unit` value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` and `SQL_TSI_DAY` both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
-> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
-> '2003-01-09'
```

- `TIMESTAMPDIFF(unit, datetime_expr1, datetime_expr2)` [1147]

Returns `datetime_expr2 - datetime_expr1`, where `datetime_expr1` and `datetime_expr2` are date or datetime expressions. One expression may be a date and the other a datetime; a date value is treated as a datetime having the time part `'00:00:00'` where necessary. The unit for the result (an integer) is given by the `unit` argument. The legal values for `unit` are the same as those listed in the description of the `TIMESTAMPADD()` [1147] function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
-> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
-> -1
mysql> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
-> 128885
```



Note

The order of the date or datetime arguments for this function is the opposite of that used with the `TIMESTAMP()` [1147] function when invoked with 2 arguments.

- [TIME_FORMAT\(*time*, *format*\)](#) [1148]

This is used like the [DATE_FORMAT\(\)](#) [1138] function, but the *format* string may contain format specifiers only for hours, minutes, seconds, and microseconds. Other specifiers produce a `NULL` value or 0.

If the *time* value contains an hour part that is greater than 23, the `%H` and `%k` hour format specifiers produce a value larger than the usual range of 0..23. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
-> '100 100 04 04 4'
```

- [TIME_TO_SEC\(*time*\)](#) [1148]

Returns the *time* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
-> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
-> 2378
```

- [TO_DAYS\(*date*\)](#) [1148]

Given a date *date*, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
-> 728779
mysql> SELECT TO_DAYS('2007-10-07');
-> 733321
```

[TO_DAYS\(\)](#) [1148] is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See [Section 12.8, “What Calendar Is Used By MySQL?”](#), for details.

Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in [Section 11.3, “Date and Time Types”](#). For example, '2008-10-07' and '08-10-07' are seen as identical dates:

```
mysql> SELECT TO_DAYS('2008-10-07'), TO_DAYS('08-10-07');
-> 733687, 733687
```

In MySQL, the zero date is defined as '0000-00-00', even though this date is itself considered invalid. This means that, for '0000-00-00' and '0000-01-01', [TO_DAYS\(\)](#) [1148] returns the values shown here:

```
mysql> SELECT TO_DAYS('0000-00-00');
+-----+
| to_days('0000-00-00') |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT TO_DAYS('0000-01-01');
+-----+
| to_days('0000-01-01') |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

This is true whether or not the `ALLOW_INVALID_DATES` [612] SQL server mode is enabled.

- `TO_SECONDS(expr)` [1149]

Given a date or datetime `expr`, returns a the number of seconds since the year 0. If `expr` is not a valid date or datetime value, returns `NULL`.

```
mysql> SELECT TO_SECONDS(950501);
-> 62966505600
mysql> SELECT TO_SECONDS('2009-11-29');
-> 63426672000
mysql> SELECT TO_SECONDS('2009-11-29 13:43:32');
-> 63426721412
mysql> SELECT TO_SECONDS(NOW());
-> 63426721458
```

Like `TO_DAYS()` [1148], `TO_SECONDS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See [Section 12.8, “What Calendar Is Used By MySQL?”](#), for details.

Like `TO_DAYS()` [1148], `TO_SECONDS()`, converts two-digit year values in dates to four-digit form using the rules in [Section 11.3, “Date and Time Types”](#).

`TO_SECONDS()` is available beginning with MySQL 5.5.0.

In MySQL, the zero date is defined as `'0000-00-00'`, even though this date is itself considered invalid. This means that, for `'0000-00-00'` and `'0000-01-01'`, `TO_SECONDS()` [1149] returns the values shown here:

```
mysql> SELECT TO_SECONDS('0000-00-00');
+-----+
| TO_SECONDS('0000-00-00') |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT TO_SECONDS('0000-01-01');
+-----+
| TO_SECONDS('0000-01-01') |
+-----+
| 86400 |
+-----+
```

```
1 row in set (0.00 sec)
```

This is true whether or not the `ALLOW_INVALID_DATES` [612] SQL server mode is enabled.

- `UNIX_TIMESTAMP()` [1150], `UNIX_TIMESTAMP(date)` [1150]

If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If `UNIX_TIMESTAMP()` [1150] is called with a `date` argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. `date` may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format `YYMMDD` or `YYYYMMDD`. The server interprets `date` as a value in the current time zone and converts it to an internal value in UTC. Clients can set their time zone as described in Section 10.6, "MySQL Server Time Zone Support".

```
mysql> SELECT UNIX_TIMESTAMP();
-> 1196440210
mysql> SELECT UNIX_TIMESTAMP('2007-11-30 10:30:19');
-> 1196440219
```

When `UNIX_TIMESTAMP()` [1150] is used on a `TIMESTAMP` column, the function returns the internal timestamp value directly, with no implicit "string-to-Unix-timestamp" conversion. If you pass an out-of-range date to `UNIX_TIMESTAMP()` [1150], it returns 0.

Note: If you use `UNIX_TIMESTAMP()` [1150] and `FROM_UNIXTIME()` [1140] to convert between `TIMESTAMP` values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For example, due to conventions for local time zone changes, it is possible for two `UNIX_TIMESTAMP()` [1150] to map two `TIMESTAMP` values to the same Unix timestamp value. `FROM_UNIXTIME()` [1140] will map that value back to only one of the original `TIMESTAMP` values. Here is an example, using `TIMESTAMP` values in the `CET` time zone:

```
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+-----+
| 1111885200 |
+-----+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+-----+
| 1111885200 |
+-----+
mysql> SELECT FROM_UNIXTIME(1111885200);
+-----+
| FROM_UNIXTIME(1111885200) |
+-----+
| 2005-03-27 03:00:00 |
+-----+
```

If you want to subtract `UNIX_TIMESTAMP()` [1150] columns, you might want to cast the result to signed integers. See Section 12.10, "Cast Functions and Operators".

- `UTC_DATE` [1150], `UTC_DATE()` [1150]

Returns the current UTC date as a value in 'YYYY-MM-DD' or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
-> '2003-08-14', 20030814
```

- `UTC_TIME` [1150], `UTC_TIME()` [1150]

Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
-> '18:07:53', 180753.000000
```

- [UTC_TIMESTAMP \[1151\]](#), [UTC_TIMESTAMP\(\) \[1151\]](#)

Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
-> '2003-08-14 18:08:04', 20030814180804.000000
```

- [WEEK\(date\[,mode\]\) \[1151\]](#)

This function returns the week number for *date*. The two-argument form of [WEEK\(\) \[1151\]](#) enables you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the *mode* argument is omitted, the value of the [default_week_format \[494\]](#) system variable is used. See [Section 5.1.4, “Server System Variables”](#).

The following table describes how the *mode* argument works.

Mode	First day of week	Range	Week 1 is the first week ...
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with more than 3 days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with more than 3 days this year
4	Sunday	0-53	with more than 3 days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with more than 3 days this year
7	Monday	1-53	with a Monday in this year

```
mysql> SELECT WEEK('2008-02-20');
-> 7
mysql> SELECT WEEK('2008-02-20',0);
-> 7
mysql> SELECT WEEK('2008-02-20',1);
-> 8
mysql> SELECT WEEK('2008-12-31',1);
-> 53
```

Note that if a date falls in the last week of the previous year, MySQL returns 0 if you do not use 2, 3, 6, or 7 as the optional *mode* argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
-> 2000, 0
```

One might argue that MySQL should return 52 for the [WEEK\(\) \[1151\]](#) function, because the given date actually occurs in the 52nd week of 1999. We decided to return 0 instead because we want the function to return “the week number in the given year.” This makes use of the [WEEK\(\) \[1151\]](#) function reliable when combined with other functions that extract a date part from a date.

If you would prefer the result to be evaluated with respect to the year that contains the first day of the week for the given date, use 0, 2, 5, or 7 as the optional *mode* argument.

```
mysql> SELECT WEEK('2000-01-01',2);
-> 52
```

Alternatively, use the `YEARWEEK()` [1152] function:

```
mysql> SELECT YEARWEEK('2000-01-01');
-> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
-> '52'
```

- `WEEKDAY(date)` [1152]

Returns the weekday index for *date* (0 = Monday, 1 = Tuesday, ... 6 = Sunday).

```
mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
-> 6
mysql> SELECT WEEKDAY('2007-11-06');
-> 1
```

- `WEEKOFYEAR(date)` [1152]

Returns the calendar week of the date as a number in the range from 1 to 53. `WEEKOFYEAR()` [1152] is a compatibility function that is equivalent to `WEEK(date, 3)` [1151].

```
mysql> SELECT WEEKOFYEAR('2008-02-20');
-> 8
```

- `YEAR(date)` [1152]

Returns the year for *date*, in the range 1000 to 9999, or 0 for the “zero” date.

```
mysql> SELECT YEAR('1987-01-01');
-> 1987
```

- `YEARWEEK(date)` [1152], `YEARWEEK(date,mode)` [1152]

Returns year and week for a date. The *mode* argument works exactly like the *mode* argument to `WEEK()` [1151]. The year in the result may be different from the year in the date argument for the first and the last week of the year.

```
mysql> SELECT YEARWEEK('1987-01-01');
-> 198653
```

Note that the week number is different from what the `WEEK()` [1151] function would return (0) for optional arguments 0 or 1, as `WEEK()` [1151] then returns the week in the context of the given year.

12.8. What Calendar Is Used By MySQL?

MySQL uses what is known as a *proleptic Gregorian calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred.

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	2	3	4	15	16	17

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
18	19	20	21	22	23	24
25	26	27	28	29	30	31

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are nonexistent.

A calendar applied to dates when it was not actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL `DATE` or `DATETIME` values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

12.9. Full-Text Search Functions

`MATCH (col1,col2,...) AGAINST (expr [search_modifier])` [1153]

```
search_modifier:
{
  IN NATURAL LANGUAGE MODE
  | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
  | IN BOOLEAN MODE
  | WITH QUERY EXPANSION
}
```

MySQL has support for full-text indexing and searching:

- A full-text index in MySQL is an index of type `FULLTEXT`.
- Full-text indexes can be used only with `MyISAM` tables, and can be created only for `CHAR`, `VARCHAR`, or `TEXT` columns.
- A `FULLTEXT` index definition can be given in the `CREATE TABLE` statement when a table is created, or added later using `ALTER TABLE` or `CREATE INDEX`.
- For large data sets, it is much faster to load your data into a table that has no `FULLTEXT` index and then create the index after that, than to load data into a table that has an existing `FULLTEXT` index.

Full-text searching is performed using `MATCH() ... AGAINST` [1153] syntax. `MATCH() [1153]` takes a comma-separated list that names the columns to be searched. `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.

There are three types of full-text searches:

- A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators. The stopword list applies. In addition, words that are present in 50% or more of the rows are considered common and do not match.

Full-text searches are natural language searches if the `IN NATURAL LANGUAGE MODE` modifier is given or if no modifier is given. For more information, see [Section 12.9.1, “Natural Language Full-Text Searches”](#).

- A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Common words such as “some” or “then” are stopwords and do not match if present in the search string. The `IN BOOLEAN MODE` modifier specifies a boolean search. For more information, see [Section 12.9.2, “Boolean Full-Text Searches”](#).
- A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` or `WITH QUERY EXPANSION` modifier specifies a query expansion search. For more information, see [Section 12.9.3, “Full-Text Searches with Query Expansion”](#).

Constraints on full-text searching are listed in [Section 12.9.5, “Full-Text Restrictions”](#).

The `mysam_ftdump` utility can be used to dump the contents of a full-text index. This may be helpful for debugging full-text queries. See [Section 4.6.2, “mysam_ftdump — Display Full-Text Index information”](#).

12.9.1. Natural Language Full-Text Searches

By default or with the `IN NATURAL LANGUAGE MODE` modifier, the `MATCH()` [1153] function performs a natural language search for a string against a *text collection*. A collection is a set of one or more columns included in a `FULLTEXT` index. The search string is given as the argument to `AGAINST()`. For each row in the table, `MATCH()` [1153] returns a relevance value; that is, a similarity measure between the search string and the text in that row in the columns named in the `MATCH()` [1153] list.

```
mysql> CREATE TABLE articles (
->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
->   title VARCHAR(200),
->   body TEXT,
->   FULLTEXT (title,body)
-> ) ENGINE=MyISAM;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles (title,body) VALUES
-> ('MySQL Tutorial','DBMS stands for DataBase ...'),
-> ('How To Use MySQL Well','After you went through a ...'),
-> ('Optimizing MySQL','In this tutorial we will show ...'),
-> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
-> ('MySQL vs. YourSQL','In the following database comparison ...'),
-> ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+-----+-----+
| id | title                | body                |
+-----+-----+-----+
| 5 | MySQL vs. YourSQL    | In the following database comparison ... |
| 1 | MySQL Tutorial      | DBMS stands for DataBase ... |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

By default, the search is performed in case-insensitive fashion. However, you can perform a case-sensitive full-text search by using a binary collation for the indexed columns. For example, a column that uses the `latin1` character set of can be assigned a collation of `latin1_bin` to make it case sensitive for full-text searches.

When `MATCH()` [1153] is used in a `WHERE` clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first. Relevance values are nonnegative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row, the number of unique words in that row, the total number of words in the collection, and the number of documents (rows) that contain a particular word.

To simply count matches, you could use a query like this:

```
mysql> SELECT COUNT(*) FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+
| COUNT(*) |
+-----+
|         2 |
+-----+
1 row in set (0.00 sec)
```

However, you might find it quicker to rewrite the query as follows:

```
mysql> SELECT
-> COUNT(IF(MATCH (title,body) AGAINST ('database' IN NATURAL LANGUAGE MODE), 1, NULL))
-> AS count
-> FROM articles;
+-----+
| count |
+-----+
|       2 |
+-----+
1 row in set (0.03 sec)
```

The first query sorts the results by relevance whereas the second does not. However, the second query performs a full table scan and the first does not. The first may be faster if the search matches few rows; otherwise, the second may be faster because it would read many rows anyway.

For natural-language full-text searches, it is a requirement that the columns named in the `MATCH()` [1153] function be the same columns included in some `FULLTEXT` index in your table. For the preceding query, note that the columns named in the `MATCH()` [1153] function (`title` and `body`) are the same as those named in the definition of the `article` table's `FULLTEXT` index. If you wanted to search the `title` or `body` separately, you would need to create separate `FULLTEXT` indexes for each column.

It is also possible to perform a boolean search or a search with query expansion. These search types are described in [Section 12.9.2, "Boolean Full-Text Searches"](#), and [Section 12.9.3, "Full-Text Searches with Query Expansion"](#).

A full-text search that uses an index can name columns only from a single table in the `MATCH()` [1153] clause because an index cannot span multiple tables. A boolean search can be done in the absence of an index (albeit more slowly), in which case it is possible to name columns from multiple tables.

The preceding example is a basic illustration that shows how to use the `MATCH()` [1153] function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the `SELECT` statement includes neither `WHERE` nor `ORDER BY` clauses:

```
mysql> SELECT id, MATCH (title,body)
-> AGAINST ('Tutorial' IN NATURAL LANGUAGE MODE) AS score
-> FROM articles;
+-----+-----+
| id | score |
+-----+-----+
| 1 | 0.65545833110809 |
+-----+-----+
```

```

| 2 | 0 |
| 3 | 0.66266459226608 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
+-----+
6 rows in set (0.00 sec)

```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, you should specify `MATCH () [1153]` twice: once in the `SELECT` list and once in the `WHERE` clause. This causes no additional overhead, because the MySQL optimizer notices that the two `MATCH () [1153]` calls are identical and invokes the full-text search code only once.

```

mysql> SELECT id, body, MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root'
-> IN NATURAL LANGUAGE MODE) AS score
-> FROM articles WHERE MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root'
-> IN NATURAL LANGUAGE MODE);
+-----+-----+-----+
| id | body | score |
+-----+-----+-----+
| 4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
| 6 | When configured properly, MySQL ... | 1.3114095926285 |
+-----+-----+-----+
2 rows in set (0.00 sec)

```

The MySQL `FULLTEXT` implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes (“'”), but not more than one in a row. This means that `aaa'bbb` is regarded as one word, but `aaa' 'bbb` is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the `FULLTEXT` parser; `'aaa'bbb'` would be parsed as `aaa'bbb`.

The `FULLTEXT` parser determines where words start and end by looking for certain delimiter characters; for example, “ ” (space), “,” (comma), and “.” (period). If words are not separated by delimiters (as in, for example, Chinese), the `FULLTEXT` parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a `FULLTEXT` index, you must preprocess them so that they are separated by some arbitrary delimiter such as “ ”.

In MySQL 5.5, it is possible to write a plugin that replaces the built-in full-text parser. For details, see [Section 23.2, “The MySQL Plugin API”](#). For example parser plugin source code, see the `plugin/fulltext` directory of a MySQL source distribution.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is four characters.
- Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overwritten by a user-defined list.

The default stopword list is given in [Section 12.9.4, “Full-Text Stopwords”](#). The default minimum word length and stopword list can be changed as described in [Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”](#).

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Consequently, a word that is present in many documents has a lower weight (and may even have a zero weight), because it has lower semantic value in this particular collection. Conversely,

if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row.

Such a technique works best with large collections (in fact, it was carefully tuned this way). For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results. For example, although the word “MySQL” is present in every row of the `articles` table shown earlier, a search for the word produces no results:

```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('MySQL' IN NATURAL LANGUAGE MODE);
Empty set (0.00 sec)
```

The search result is empty because the word “MySQL” is present in at least 50% of the rows. As such, it is effectively treated as a stopword. For large data sets, this is the most desirable behavior: A natural language query should not return every second row from a 1GB table. For small data sets, it may be less desirable.

A word that matches half of the rows in a table is less likely to locate relevant documents. In fact, it most likely finds plenty of irrelevant documents. We all know this happens far too often when we are trying to find something on the Internet with a search engine. It is with this reasoning that rows containing the word are assigned a low semantic value for *the particular data set in which they occur*. A given word may reach the 50% threshold in one data set but not another.

The 50% threshold has a significant implication when you first try full-text searching to see how it works: If you create a table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results. Be sure to insert at least three rows, and preferably many more. Users who need to bypass the 50% limitation can use the boolean search mode; see [Section 12.9.2, “Boolean Full-Text Searches”](#).

12.9.2. Boolean Full-Text Searches

MySQL can perform boolean full-text searches using the `IN BOOLEAN MODE` modifier. With this modifier, certain characters have special meaning at the beginning or end of words in the search string. In the following query, the `+` and `-` operators indicate that a word is required to be present or absent, respectively, for a match to occur. Thus, the query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL”:

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
-> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+-----+-----+-----+
| id | title | body |
+-----+-----+-----+
| 1 | MySQL Tutorial | DBMS stands for DataBase ... |
| 2 | How To Use MySQL Well | After you went through a ... |
| 3 | Optimizing MySQL | In this tutorial we will show ... |
| 4 | 1001 MySQL Tricks | 1. Never run mysqld as root. 2. ... |
| 6 | MySQL Security | When configured properly, MySQL ... |
+-----+-----+-----+
```



Note

In implementing this feature, MySQL uses what is sometimes referred to as *implied Boolean logic*, in which

- `+` stands for `AND`
- `-` stands for `NOT`

- `[no operator]` implies `OR`

Boolean full-text searches have these characteristics:

- They do not use the 50% threshold.
- They do not automatically sort rows in order of decreasing relevance. You can see this from the preceding query result: The row with the highest relevance is the one that contains “MySQL” twice, but it is listed last, not first.
- They can work even without a `FULLTEXT` index, although a search executed in this fashion would be quite slow.
- The minimum and maximum word length full-text parameters apply.
- The stopword list applies.

The boolean full-text search capability supports the following operators:

- `+`

A leading plus sign indicates that this word *must* be present in each row that is returned.

- `-`

A leading minus sign indicates that this word must *not* be present in any of the rows that are returned.

Note: The `-` operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by `-` returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

- (no operator)

By default (when neither `+` nor `-` is specified) the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() . . . AGAINST() [1153]` without the `IN BOOLEAN MODE` modifier.

- `>` `<`

These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The `>` operator increases the contribution and the `<` operator decreases it. See the example following this list.

- `()`

Parentheses group words into subexpressions. Parenthesized groups can be nested.

- `~`

A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the `-` operator.

- `*`

The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it should be *appended* to the word to be affected. Words match if they begin with the word preceding the `*` operator.

If a word is specified with the truncation operator, it is not stripped from a boolean query, even if it is too short (as determined from the `ft_min_word_len` [501] setting) or a stopword. This occurs because the word is not seen as too short or a stopword, but as a prefix that must be present in the document in the form of a word that begins with the prefix. Suppose that `ft_min_word_len=4`. Then a search for `'+word +the*'` will likely return fewer rows than a search for `'+word +the'`:

- The former query remains as is and requires both `word` and `the*` (a word starting with `the`) to be present in the document.
- The latter query is transformed to `+word` (requiring only `word` to be present). `the` is both too short and a stopword, and either condition is enough to cause it to be ignored.
- "

A phrase that is enclosed within double quote (“”) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words and performs a search in the `FULLTEXT` index for the words. Nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, `"test phrase"` matches `"test, phrase"`.

If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The following examples demonstrate some search strings that use boolean full-text operators:

- `'apple banana'`

Find rows that contain at least one of the two words.

- `'+apple +juice'`

Find rows that contain both words.

- `'+apple macintosh'`

Find rows that contain the word “apple”, but rank rows higher if they also contain “macintosh”.

- `'+apple -macintosh'`

Find rows that contain the word “apple” but not “macintosh”.

- `'+apple ~macintosh'`

Find rows that contain the word “apple”, but if the row also contains the word “macintosh”, rate it lower than if row does not. This is “softer” than a search for `'+apple -macintosh'`, for which the presence of “macintosh” causes the row not to be returned at all.

- `'+apple +(>turnover <strudel)'`

Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.

- `'apple*'`

Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

- `'"some words"'`

Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”). Note that the “” characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotation marks that enclose the search string itself.

12.9.3. Full-Text Searches with Query Expansion

Full-text search supports query expansion (and in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL”, “Oracle”, “DB2”, and “RDBMS” all are phrases that should match “databases” and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` or `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL”, the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database”. The following example shows this difference:

```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+-----+-----+
| id | title                | body                |
+-----+-----+-----+
| 5 | MySQL vs. YourSQL    | In the following database comparison ... |
| 1 | MySQL Tutorial       | DBMS stands for DataBase ...           |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' WITH QUERY EXPANSION);
+-----+-----+-----+
| id | title                | body                |
+-----+-----+-----+
| 1 | MySQL Tutorial       | DBMS stands for DataBase ...           |
| 5 | MySQL vs. YourSQL    | In the following database comparison ... |
| 3 | Optimizing MySQL     | In this tutorial we will show ...       |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret”. A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.



Note

Because blind query expansion tends to increase noise significantly by returning nonrelevant documents, it is meaningful to use only when a search phrase is rather short.

12.9.4. Full-Text Stopwords

The stopword list is loaded and searched for full-text queries using the server character set and collation (the values of the `character_set_server` and `collation_server` system variables). False hits or misses may occur for stopword lookups if the stopword file or columns used for full-text

indexing or searches have a character set or collation different from `character_set_server` or `collation_server`.

Case sensitivity of stopword lookups depends on the server collation. For example, lookups are case insensitive if the collation is `latin1_swedish_ci`, whereas lookups are case sensitive if the collation is `latin1_general_cs` or `latin1_bin`.

As of MySQL 5.5.6, the stopword file is loaded and searched using `latin1` if `character_set_server` is `ucs2`, `utf16`, or `utf32`. If any table was created with `FULLTEXT` indexes while the server character set was `ucs2`, `utf16`, or `utf32`, it should be repaired using this statement:

```
REPAIR TABLE tbl_name QUICK;
```

The following table shows the default list of full-text stopwords. In a MySQL source distribution, you can find this list in the `storage/myisam/ft_static.c` file.

a's	able	about	above	according
accordingly	across	actually	after	afterwards
again	against	ain't	all	allow
allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway
anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co
com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during
each	edu	eg	eight	either
else	elsewhere	enough	entirely	especially
et	etc	even	ever	every
everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few

Full-Text Stopwords

fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten
greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored
immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	known	knows	last	lately
later	latter	latterly	least	less
lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv

Full-Text Stopwords

rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems
seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends
th	than	thank	thanks	thanx
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think
third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly
try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who
who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
wouldn't	yes	yet	you	you'd

you'll	you're	you've	your	yours
yourself	yourselves	zero		

12.9.5. Full-Text Restrictions

- Full-text searches are supported for [MyISAM](#) tables only.
- Full-text searches are not supported for partitioned tables. See [Section 18.5, “Restrictions and Limitations on Partitioning”](#).
- Full-text searches can be used with most multi-byte character sets. The exception is that for Unicode, the `utf8` character set can be used, but not the `ucs2` character set. However, although `FULLTEXT` indexes on `ucs2` columns cannot be used, you can perform `IN BOOLEAN MODE` searches on a `ucs2` column that has no such index.

The remarks for `utf8` also apply to `utf8mb4`, and the remarks for `ucs2` also apply to `utf16` and `utf32`.

- Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the `FULLTEXT` parser *cannot determine where words begin and end in these and other such languages*. The implications of this and some workarounds for the problem are described in [Section 12.9, “Full-Text Search Functions”](#).
- Although the use of multiple character sets within a single table is supported, all columns in a `FULLTEXT` index must use the same character set and collation.
- The `MATCH() [1153]` column list must match exactly the column list in some `FULLTEXT` index definition for the table, unless this `MATCH() [1153]` is `IN BOOLEAN MODE`. Boolean-mode searches can be done on nonindexed columns, although they are likely to be slow.
- The argument to `AGAINST()` must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.
- Index hints are more limited for `FULLTEXT` searches than for non-`FULLTEXT` searches. See [Section 13.2.9.3, “Index Hint Syntax”](#).

12.9.6. Fine-Tuning MySQL Full-Text Search

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See [Section 2.10, “Installing MySQL from Source”](#).

Note that full-text search is carefully tuned for the most effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do not alter the MySQL sources unless you know what you are doing.*

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the `FULLTEXT` indexes in your tables. Instructions for doing so are given later in this section.

- The minimum and maximum lengths of words to be indexed are defined by the `ft_min_word_len [501]` and `ft_max_word_len [501]` system variables. (See [Section 5.1.4, “Server System Variables”](#).) The default minimum value is four characters; the default maximum is version dependent. If you change either value, you must rebuild your `FULLTEXT` indexes. For example, if you

want three-character words to be searchable, you can set the `ft_min_word_len` [501] variable by putting the following lines in an option file:

```
[mysqld]
ft_min_word_len=3
```

Then restart the server and rebuild your `FULLTEXT` indexes. Note particularly the remarks regarding `myisamchk` in the instructions following this list.

- To override the default stopword list, set the `ft_stopword_file` [502] system variable. (See [Section 5.1.4, “Server System Variables”](#).) The variable value should be the path name of the file containing the stopword list, or the empty string to disable stopword filtering. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. After changing the value of this variable or the contents of the stopword file, restart the server and rebuild your `FULLTEXT` indexes.

The stopword list is free-form. That is, you may use any nonalphanumeric character such as newline, space, or comma to separate stopwords. Exceptions are the underscore character (“_”) and a single apostrophe (“’”) which are treated as part of a word. The character set of the stopword list is the server’s default character set; see [Section 10.1.3.1, “Server Character Set and Collation”](#).

- The 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in `storage/myisam/ftdefs.h`:

```
#define GWS_IN_USE GWS_PROB
```

Change that line to this:

```
#define GWS_IN_USE GWS_FREQ
```

Then recompile MySQL. There is no need to rebuild the indexes in this case.



Note

By making this change, you *severely* decrease MySQL’s ability to provide adequate relevance values for the `MATCH()` [1153] function. If you really need to search for such common words, it would be better to search using `IN BOOLEAN MODE` instead, which does not observe the 50% threshold.

- To change the operators used for boolean full-text searches, set the `ft_boolean_syntax` [500] system variable. This variable can be changed while the server is running, but you must have the `SUPER` [697] privilege to do so. No rebuilding of indexes is necessary in this case. See [Section 5.1.4, “Server System Variables”](#), which describes the rules governing how to set this variable.
- If you want to change the set of characters that are considered word characters, you can do so in several ways, as described in the following list. After making the modification, you must rebuild the indexes for each table that contains any `FULLTEXT` indexes. Suppose that you want to treat the hyphen character (“-”) as a word character. Use one of these methods:
 - Modify the MySQL source: In `storage/myisam/ftdefs.h`, see the `true_word_char()` and `misc_word_char()` macros. Add “-” to one of those macros and recompile MySQL.
 - Modify a character set file: This requires no recompilation. The `true_word_char()` macro uses a “character type” table to distinguish letters and numbers from other characters. . You can edit the contents of the `<ctype><map>` array in one of the character set XML files to specify that “-” is a “letter.” Then use the given character set for your `FULLTEXT` indexes. For information about the `<ctype><map>` array format, see [Section 10.3.1, “Character Definition Arrays”](#).

- Add a new collation for the character set used by the indexed columns, and alter the columns to use that collation. For general information about adding collations, see [Section 10.4, “Adding a Collation to a Character Set”](#). For an example specific to full-text indexing, see [Section 12.9.7, “Adding a Collation for Full-Text Indexing”](#).

If you modify full-text variables that affect indexing (`ft_min_word_len` [501], `ft_max_word_len` [501], or `ft_stopword_file` [502]), or if you change the stopwords file itself, you must rebuild your `FULLTEXT` indexes after making the changes and restarting the server. To rebuild the indexes in this case, it is sufficient to do a `QUICK` repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Alternatively, use `ALTER TABLE` with the `DROP INDEX` and `ADD INDEX` options to drop and re-create each `FULLTEXT` index. In some cases, this may be faster than a repair operation.

Each table that contains any `FULLTEXT` index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table will cause the server to see the table as corrupt and in need of repair.

Note that if you use `myisamchk` to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopwords file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or stopwords file values used by the server, specify the same `ft_min_word_len` [501], `ft_max_word_len` [501], and `ft_stopword_file` [502] values for `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` for index modification is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE` statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

12.9.7. Adding a Collation for Full-Text Indexing

This section describes how to add a new collation for full-text searches. The sample collation is like `latin1_swedish_ci` but treats the `'-'` character as a letter rather than as a punctuation character so that it can be indexed as a word character. General information about adding collations is given in [Section 10.4, “Adding a Collation to a Character Set”](#); it is assumed that you have read it and are familiar with the files involved.

To add a collation for full-text indexing, use this procedure:

1. Add a collation to the `Index.xml` file. The collation ID must be unused, so choose a value different from 62 if that ID is already taken on your system.

```
<charset name="latin1">
...
<collation name="latin1_fulltext_ci" id="62"/>
</charset>
```

2. Declare the sort order for the collation in the `latin1.xml` file. In this case, the order can be copied from `latin1_swedish_ci`:

```
<collation name="latin1_fulltext_ci">
<map>
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5D D7 D8 55 55 55 59 59 DE DF
41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
44 4E 4F 4F 4F 5D F7 D8 55 55 55 59 59 DE FF
</map>
</collation>
```

3. Modify the `ctype` array in `latin1.xml`. Change the value corresponding to 0x2D (which is the code for the '-' character) from 10 (punctuation) to 01 (small letter). In the following array, this is the element in the fourth row down, third value from the end.

```
<ctype>
<map>
00
20 20 20 20 20 20 20 20 20 28 28 28 28 28 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
48 10 10 10 10 10 10 10 10 10 10 10 10 01 10 10
84 84 84 84 84 84 84 84 84 84 10 10 10 10 10 10
10 81 81 81 81 81 81 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 10 10 10 10
10 82 82 82 82 82 82 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02 02 02 02 02 10 10 10 20
10 00 10 02 10 10 10 10 10 10 01 10 01 00 01 00
00 10 10 10 10 10 10 10 10 10 02 10 02 00 02 01
48 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 10 01 01 01 01 01 01 01 02
02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 10 02 02 02 02 02 02 02 02
</map>
</ctype>
```

4. Restart the server.
5. To employ the new collation, include it in the definition of columns that are to use it:

```
mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE t1 (
-> a TEXT CHARACTER SET latin1 COLLATE latin1_fulltext_ci,
-> FULLTEXT INDEX(a)
```

```
-> ) ENGINE=MyISAM;
Query OK, 0 rows affected (0.47 sec)
```

6. Test the collation to verify that hyphen is considered as a word character:

```
mysql> INSERT INTO t1 VALUES ('----'),('....'),('abcd');
Query OK, 3 rows affected (0.22 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1 WHERE MATCH a AGAINST ('----' IN BOOLEAN MODE);
+-----+
| a      |
+-----+
| ----  |
+-----+
1 row in set (0.00 sec)
```

12.10. Cast Functions and Operators

Table 12.14. Cast Functions

Name	Description
BINARY [1168]	Cast a string to a binary string
CAST() [1168]	Cast a value as a certain type
CONVERT() [1168]	Cast a value as a certain type

- [BINARY \[1168\]](#)

The [BINARY \[1168\]](#) operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column is not defined as [BINARY](#) or [BLOB](#). [BINARY \[1168\]](#) also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
-> 1
mysql> SELECT BINARY 'a' = 'A';
-> 0
mysql> SELECT 'a' = 'a ';
-> 1
mysql> SELECT BINARY 'a' = 'a ';
-> 0
```

In a comparison, [BINARY \[1168\]](#) affects the entire operation; it can be given before either operand with the same result.

[BINARY str \[1168\]](#) is shorthand for [CAST\(str AS BINARY\) \[1168\]](#).

Note that in some contexts, if you cast an indexed column to [BINARY](#), MySQL is not able to use the index efficiently.

- [CAST\(expr AS type\) \[1168\]](#)

The [CAST\(\) \[1168\]](#) function takes an expression of any type and produces a result value of a specified type, similar to [CONVERT\(\) \[1168\]](#). See the description of [CONVERT\(\) \[1168\]](#) for more information.

- [CONVERT\(expr, type\) \[1168\]](#), [CONVERT\(expr USING transcoding_name\) \[1168\]](#)

The [CONVERT\(\) \[1168\]](#) and [CAST\(\) \[1168\]](#) functions take an expression of any type and produce a result value of a specified type.

The *type* for the result can be one of the following values:

- `BINARY[(N)]`
- `CHAR[(N)]`
- `DATE`
- `DATETIME`
- `DECIMAL[(M[,D])]`
- `SIGNED [INTEGER]`
- `TIME`
- `UNSIGNED [INTEGER]`

`BINARY` produces a string with the `BINARY` data type. See [Section 11.4.2, “The `BINARY` and `VARBINARY` Types”](#) for a description of how this affects comparisons. If the optional length *N* is given, `BINARY(N)` causes the cast to use no more than *N* bytes of the argument. Values shorter than *N* bytes are padded with `0x00` bytes to a length of *N*.

`CHAR(N)` [1101] causes the cast to use no more than *N* characters of the argument.

`CAST()` [1168] and `CONVERT(... USING ...)` [1168] are standard SQL syntax. The non-`USING` form of `CONVERT()` [1168] is ODBC syntax.

`CONVERT()` [1168] with `USING` is used to convert data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string `'abc'` in the default character set to the corresponding string in the `utf8` character set:

```
SELECT CONVERT('abc' USING utf8);
```

Normally, you cannot compare a `BLOB` value or other binary string in case-insensitive fashion because binary strings have no character set, and thus no concept of lettercase. To perform a case-insensitive comparison, use the `CONVERT()` [1168] function to convert the value to a nonbinary string. Comparisons of the result use the string collation. For example, if the character set of the result has a case-insensitive collation, a `LIKE` [1111] operation is not case sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

To use a different character set, substitute its name for `latin1` in the preceding statement. To specify a particular collation for the converted string, use a `COLLATE` clause following the `CONVERT()` [1168] call, as described in [Section 10.1.9.2, “`CONVERT\(\)` and `CAST\(\)`”](#). For example, to use `latin1_german1_ci`:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) COLLATE latin1_german1_ci
FROM tbl_name;
```

`CONVERT()` [1168] can be used more generally for comparing strings that are represented in different character sets.

`LOWER()` [1105] (and `UPPER()` [1110]) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
```

```
+-----+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-----+
| New York    | new york                            |
+-----+
```

The cast functions are useful when you want to create a column with a specific type in a `CREATE TABLE ... SELECT` statement:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

The functions also can be useful for sorting `ENUM` columns in lexical order. Normally, sorting of `ENUM` columns occurs using the internal numeric values. Casting the values to `CHAR` results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

`CAST(str AS BINARY)` [1168] is the same thing as `BINARY str` [1168]. `CAST(expr AS CHAR)` [1168] treats the expression as a string with the default character set.

`CAST()` [1168] also changes the result if you use it as part of a more complex expression such as `CONCAT('Date: ', CAST(NOW() AS DATE))` [1102].

You should not use `CAST()` [1168] to extract data in different formats but instead use string functions like `LEFT()` [1104] or `EXTRACT()` [1140]. See [Section 12.7, “Date and Time Functions”](#).

To cast a string to a numeric value in numeric context, you normally do not have to do anything other than to use the string value as though it were a number:

```
mysql> SELECT 1+'1';
-> 2
```

If you use a string in an arithmetic operation, it is converted to a floating-point number during expression evaluation.

If you use a number in string context, the number automatically is converted to a string:

```
mysql> SELECT CONCAT('hello you ',2);
-> 'hello you 2'
```

For information about implicit conversion of numbers to strings, see [Section 12.2, “Type Conversion in Expression Evaluation”](#).

MySQL supports arithmetic with both signed and unsigned 64-bit values. If you are using numeric operators (such as `+` [1121] or `-` [1122]) and one of the operands is an unsigned integer, the result is unsigned by default (see [Section 12.6.1, “Arithmetic Operators”](#)). You can override this by using the `SIGNED` or `UNSIGNED` cast operator to cast a value to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
-> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
-> -1
```

If either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, `DECIMAL` column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
-> -1.0
```

The SQL mode affects the result of conversion operations. Examples:

- If you convert a “zero” date string to a date, `CONVERT()` [1168] and `CAST()` [1168] return `NULL` and produce a warning when the `NO_ZERO_DATE` [615] SQL mode is enabled.

- For integer subtraction, if the `NO_UNSIGNED_SUBTRACTION` [614] SQL mode is enabled, the subtraction result is signed even if any operand is unsigned.

For more information, see [Section 5.1.7, “Server SQL Modes”](#).

12.11. XML Functions

Table 12.15. XML Functions

Name	Description
<code>ExtractValue()</code> [1173]	Extracts a value from an XML string using XPath notation
<code>UpdateXML()</code> [1174]	Return replaced XML fragment

This section discusses XML and related functionality in MySQL.



Note

It is possible to obtain XML-formatted output from MySQL in the `mysql` and `mysqldump` clients by invoking them with the `--xml` [282] option. See [Section 4.5.1, “mysql — The MySQL Command-Line Tool”](#), and [Section 4.5.4, “mysqldump — A Database Backup Program”](#).

Two functions providing basic XPath 1.0 (XML Path Language, version 1.0) capabilities are available. Some basic information about XPath syntax and usage is provided later in this section; however, an in-depth discussion of these topics is beyond the scope of this Manual, and you should refer to the [XML Path Language \(XPath\) 1.0 standard](#) for definitive information. A useful resource for those new to XPath or who desire a refresher in the basics is the [Zvon.org XPath Tutorial](#), which is available in several languages.



Note

These functions remain under development. We continue to improve these and other aspects of XML and XPath functionality in MySQL 5.5 and onwards. You may discuss these, ask questions about them, and obtain help from other users with them in the [MySQL XML User Forum](#).

XPath expressions used with these functions support user variables and local stored program variables. User variables are weakly checked; variables local to stored programs are strongly checked (see also Bug #26518):

- **User variables (weak checking).** Variables using the syntax ``${variable_name}` (that is, user variables) are not checked. No warnings or errors are issued by the server if a variable has the wrong type or has previously not been assigned a value. This also means the user is fully responsible for any typographical errors, since no warnings will be given if (for example) ``${myvairable}` is used where ``${myvariable}` was intended.

Example:

```
mysql> SET @xml = '<a><b>X</b><b>Y</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @i = 1, @j = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @i, ExtractValue(@xml, '//b[`${i}]');
+-----+-----+
| @i    | ExtractValue(@xml, '//b[`${i}]') |
+-----+-----+
| 1    | X                                |
+-----+-----+
```

```

1 row in set (0.00 sec)

mysql> SELECT @j, ExtractValue(@xml, '//b[@j]');
+-----+-----+
| @j | ExtractValue(@xml, '//b[@j]') |
+-----+-----+
| 2 | Y |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT @k, ExtractValue(@xml, '//b[@k]');
+-----+-----+
| @k | ExtractValue(@xml, '//b[@k]') |
+-----+-----+
| NULL | |
+-----+-----+
1 row in set (0.00 sec)

```

- **Variables in stored programs (strong checking).** Variables using the syntax `$variable_name` can be declared and used with these functions when they are called inside stored programs. Such variables are local to the stored program in which they are defined, and are strongly checked for type and value.

Example:

```

mysql> DELIMITER |
mysql> CREATE PROCEDURE myproc ()
-> BEGIN
-> DECLARE i INT DEFAULT 1;
-> DECLARE xml VARCHAR(25) DEFAULT '<a>X</a><a>Y</a><a>Z</a>';
->
-> WHILE i < 4 DO
-> SELECT xml, i, ExtractValue(xml, '//a[$i]');
-> SET i = i+1;
-> END WHILE;
-> END |
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> CALL myproc;
+-----+-----+-----+
| xml | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 1 | X |
+-----+-----+-----+
1 row in set (0.00 sec)

+-----+-----+-----+
| xml | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 2 | Y |
+-----+-----+-----+
1 row in set (0.01 sec)

+-----+-----+-----+
| xml | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 3 | Z |
+-----+-----+-----+
1 row in set (0.01 sec)

```

Parameters. Variables used in XPath expressions inside stored routines that are passed in as parameters are also subject to strong checking.

Expressions containing user variables or variables local to stored programs must otherwise (except for notation) conform to the rules for XPath expressions containing variables as given in the XPath 1.0 specification.



Note

Currently, a user variable used to store an XPath expression is treated as an empty string. Because of this, it is not possible to store an XPath expression as a user variable. (Bug #32911)

- `ExtractValue(xml_frag, xpath_expr)` [1173]

`ExtractValue()` [1173] takes two string arguments, a fragment of XML markup *xml_frag* and an XPath expression *xpath_expr* (also known as a *locator*); it returns the text (CDATA) of the first text node which is a child of the elements or elements matched by the XPath expression. In MySQL 5.5, the XPath expression can contain at most 127 characters. (This limitation is lifted in MySQL 5.6.)

Using this function is the equivalent of performing a match using the *xpath_expr* after appending `/text()`. In other words, `ExtractValue('<a>Sakila', '/a/b')` [1173] and `ExtractValue('<a>Sakila', '/a/b/text()')` [1173] produce the same result.

If multiple matches are found, the content of the first child text node of each matching element is returned (in the order matched) as a single, space-delimited string.

If no matching text node is found for the expression (including the implicit `/text()`)—for whatever reason, as long as *xpath_expr* is valid, and *xml_frag* consists of elements which are properly nested and closed—an empty string is returned. No distinction is made between a match on an empty element and no match at all. This is by design.

If you need to determine whether no matching element was found in *xml_frag* or such an element was found but contained no child text nodes, you should test the result of an expression that uses the XPath `count()` function. For example, both of these statements return an empty string, as shown here:

```
mysql> SELECT ExtractValue('<a><b/></a>', '/a/b');
+-----+
| ExtractValue('<a><b/></a>', '/a/b') |
+-----+
|                                     |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c/></a>', '/a/b');
+-----+
| ExtractValue('<a><c/></a>', '/a/b') |
+-----+
|                                     |
+-----+
1 row in set (0.00 sec)
```

However, you can determine whether there was actually a matching element using the following:

```
mysql> SELECT ExtractValue('<a><b/></a>', 'count(/a/b)');
+-----+
| ExtractValue('<a><b/></a>', 'count(/a/b)') |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c/></a>', 'count(/a/b)');
+-----+
|                                     |
+-----+
```

```
| ExtractValue('<a><c/></a>', 'count(/a/b)') |
+-----+
| 0 |
+-----+
1 row in set (0.01 sec)
```



Important

`ExtractValue()` [1173] returns only `CDATA`, and does not return any tags that might be contained within a matching tag, nor any of their content (see the result returned as `val1` in the following example).

```
mysql> SELECT
->   ExtractValue('<a>ccc<b>ddd</b></a>', '/a') AS val1,
->   ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b') AS val2,
->   ExtractValue('<a>ccc<b>ddd</b></a>', '//b') AS val3,
->   ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val4,
->   ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '//b') AS val5;

+-----+-----+-----+-----+-----+
| val1 | val2 | val3 | val4 | val5 |
+-----+-----+-----+-----+-----+
| ccc  | ddd  | ddd  |      | ddd eee |
+-----+-----+-----+-----+-----+
```

This function uses the current SQL collation for making comparisons with `contains()`, performing the same collation aggregation as other string functions (such as `CONCAT()` [1102]), in taking into account the collation coercibility of their arguments; see [Section 10.1.7.5, “Collation of Expressions”](#), for an explanation of the rules governing this behavior.

(Previously, binary—that is, case-sensitive—comparison was always used.)

`NULL` is returned if `xml_frag` contains elements which are not properly nested or closed, and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+-----+
| ExtractValue('<a>c</a><b', '//a') |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1523 | Incorrect XML value: 'parse error at line 1 pos 11: END-OF-INPUT unexpected ('>' wanted)' |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-----+
| ExtractValue('<a>c</a><b/>', '//a') |
+-----+
| c |
+-----+
1 row in set (0.00 sec)
```

- `UpdateXML(xml_target, xpath_expr, new_xml)` [1174]

This function replaces a single portion of a given fragment of XML markup `xml_target` with a new XML fragment `new_xml`, and then returns the changed XML. The portion of `xml_target` that is replaced

matches an XPath expression *xpath_expr* supplied by the user. In MySQL 5.5, the XPath expression can contain at most 127 characters. (This limitation is lifted in MySQL 5.6.)

If no expression matching *xpath_expr* is found, or if multiple matches are found, the function returns the original *xml_target* XML fragment. All three arguments should be strings.

```
mysql> SELECT
->   UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
->   UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
->   UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>fff</e>') AS val3,
->   UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
->   UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
-> \G

***** 1. row *****
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>
```



Note

A discussion in depth of XPath syntax and usage are beyond the scope of this Manual. Please see the [XML Path Language \(XPath\) 1.0 specification](#) for definitive information. A useful resource for those new to XPath or who are wishing a refresher in the basics is the [Zvon.org XPath Tutorial](#), which is available in several languages.

Descriptions and examples of some basic XPath expressions follow:

- */tag*

Matches *<tag/>* if and only if *<tag/>* is the root element.

Example: */a* has a match in *<a>* because it matches the outermost (root) tag. It does not match the inner *a* element in *<a>* because in this instance it is the child of another element.

- */tag1/tag2*

Matches *<tag2/>* if and only if it is a child of *<tag1/>*, and *<tag1/>* is the root element.

Example: */a/b* matches the *b* element in the XML fragment *<a>* because it is a child of the root element *a*. It does not have a match in *<a>* because in this case, *b* is the root element (and hence the child of no other element). Nor does the XPath expression have a match in *<a><c></c>*; here, *b* is a descendant of *a*, but not actually a child of *a*.

This construct is extendable to three or more elements. For example, the XPath expression */a/b/c* matches the *c* element in the fragment *<a><c></c>*.

- *//tag*

Matches any instance of *<tag>*.

Example: *//a* matches the *a* element in any of the following: *<a><c></c>*; *<c><a></c>*; *<c><a></c>*.

// can be combined with *.* For example, *//a/b* matches the *b* element in either of the fragments *<a>* or *<a><c></c>*

**Note**

`//tag` is the equivalent of `/descendant-or-self::*/*tag`. A common error is to confuse this with `/descendant-or-self::tag`, although the latter expression can actually lead to very different results, as can be seen here:

```
mysql> SET @xml = '<a><b><c>w</c><b>x</b><d>y</d>z</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @xml;
+-----+
| @xml                                     |
+-----+
| <a><b><c>w</c><b>x</b><d>y</d>z</b></a>      |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//b[1]');
+-----+
| ExtractValue(@xml, '//b[1]')           |
+-----+
| x z                                     |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//b[2]');
+-----+
| ExtractValue(@xml, '//b[2]')           |
+-----+
|                                         |
+-----+
1 row in set (0.01 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::*b[1]');
+-----+
| ExtractValue(@xml, '/descendant-or-self::*b[1]') |
+-----+
| x z                                             |
+-----+
1 row in set (0.06 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::*b[2]');
+-----+
| ExtractValue(@xml, '/descendant-or-self::*b[2]') |
+-----+
|                                         |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::b[1]');
+-----+
| ExtractValue(@xml, '/descendant-or-self::b[1]') |
+-----+
| z                                             |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::b[2]');
+-----+
| ExtractValue(@xml, '/descendant-or-self::b[2]') |
+-----+
| x                                             |
+-----+
1 row in set (0.00 sec)
```

- The `*` operator acts as a “wildcard” that matches any element. For example, the expression `*/b` matches the `b` element in either of the XML fragments `<a>` or `<c></c>`. However, the expression does not produce a match in the fragment `<a/>` because `b` must be a child of some other element. The wildcard may be used in any position: The expression `*/b/*` will match any child of a `b` element that is itself not the root element.
- You can match any of several locators using the `|` (UNION) operator. For example, the expression `//b|//c` matches all `b` and `c` elements in the XML target.
- It is also possible to match an element based on the value of one or more of its attributes. This done using the syntax `tag[@attribute="value"]`. For example, the expression `//b[@id="idB"]` matches the second `b` element in the fragment `<a><b id="idA"/><c/><b id="idB"/>`. To match against *any* element having `attribute="value"`, use the XPath expression `//*[@attribute="value"]`.

To filter multiple attribute values, simply use multiple attribute-comparison clauses in succession. For example, the expression `//b[@c="x"][@d="y"]` matches the element `<b c="x" d="y"/>` occurring anywhere in a given XML fragment.

To find elements for which the same attribute matches any of several values, you can use multiple locators joined by the `|` operator. For example, to match all `b` elements whose `c` attributes have either of the values 23 or 17, use the expression `//b[@c="23"]|//b[@c="17"]`. You can also use the logical `or` operator for this purpose: `//b[@c="23" or @c="17"]`.



Note

The difference between `or` and `|` is that `or` joins conditions, while `|` joins result sets.

XPath Limitations. The XPath syntax supported by these functions is currently subject to the following limitations:

- Nodeset-to-nodeset comparison (such as `'/a/b[@c=@d]'`) is not supported.
- All of the standard XPath comparison operators are supported. (Bug #22823)
- Relative locator expressions are resolved in the context of the root node. For example, consider the following query and result:

```
mysql> SELECT ExtractValue(
->   '<a><b c="1">X</b><b c="2">Y</b></a>',
->   'a/b'
-> ) AS result;
+-----+
| result |
+-----+
| X Y    |
+-----+
1 row in set (0.03 sec)
```

In this case, the locator `a/b` resolves to `/a/b`.

Relative locators are also supported within predicates. In the following example, `d[../@c="1"]` is resolved as `/a/b[@c="1"]/d`:

```
mysql> SELECT ExtractValue(
->   '<a>
->     <b c="1"><d>X</d></b>
->     <b c="2"><d>X</d></b>
->   </a>',
```

```

->      'a/b/d[../@c="1"]')
-> AS result;
+-----+
| result |
+-----+
| X      |
+-----+
1 row in set (0.00 sec)

```

- Locators prefixed with expressions that evaluate as scalar values—including variable references, literals, numbers, and scalar function calls—are not permitted, and their use results in an error.
- The `::` operator is not supported in combination with node types such as the following:
 - `axis::comment()`
 - `axis::text()`
 - `axis::processing-instructions()`
 - `axis::node()`

However, name tests (such as `axis::name` and `axis::*`) are supported, as shown in these examples:

```

mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::b');
+-----+
| ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::b') |
+-----+
| x                                             |
+-----+
1 row in set (0.02 sec)

mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::*');
+-----+
| ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::*') |
+-----+
| x y                                           |
+-----+
1 row in set (0.01 sec)

```

- “Up-and-down” navigation is not supported in cases where the path would lead “above” the root element. That is, you cannot use expressions which match on descendants of ancestors of a given element, where one or more of the ancestors of the current element is also an ancestor of the root element (see Bug #16321).
- The following XPath functions are not supported, or have known issues as indicated:
 - `id()`
 - `lang()`
 - `local-name()`
 - `name()`
 - `namespace-uri()`
 - `normalize-space()`
 - `starts-with()`
 - `string()`

- `substring-after()`
- `substring-before()`
- `translate()`
- The following axes are not supported:
 - `following-sibling`
 - `following`
 - `preceding-sibling`
 - `preceding`

XPath expressions passed as arguments to `ExtractValue()` [1173] and `UpdateXML()` [1174] may contain the colon character (":") in element selectors, which enables their use with markup employing XML namespaces notation. For example:

```
mysql> SET @xml = '<a>111<b:c>222<d>333</d><e:f>444</e:f></b:c></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//e:f');
+-----+
| ExtractValue(@xml, '//e:f') |
+-----+
| 444                          |
+-----+
1 row in set (0.00 sec)

mysql> SELECT UpdateXML(@xml, '//b:c', '<g:h>555</g:h>');
+-----+
| UpdateXML(@xml, '//b:c', '<g:h>555</g:h>') |
+-----+
| <a>111<g:h>555</g:h></a>                  |
+-----+
1 row in set (0.00 sec)
```

This is similar in some respects to what is permitted by [Apache Xalan](#) and some other parsers, and is much simpler than requiring namespace declarations or the use of the `namespace-uri()` and `local-name()` functions.

Error handling. For both `ExtractValue()` [1173] and `UpdateXML()` [1174], the XPath locator used must be valid and the XML to be searched must consist of elements which are properly nested and closed. If the locator is invalid, an error is generated:

```
mysql> SELECT ExtractValue('<a>c</a><b/>', '&a');
ERROR 1105 (HY000): XPATH syntax error: '&a'
```

If `xml_frag` does not consist of elements which are properly nested and closed, `NULL` is returned and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+-----+
| ExtractValue('<a>c</a><b', '//a') |
+-----+
| NULL                          |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
```

```

+-----+-----+-----+
| Level   | Code | Message
+-----+-----+-----+
| Warning | 1523 | Incorrect XML value: 'parse error at line 1 pos 11: END-OF-INPUT unexpected ('>' wanted)'
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-----+-----+
| ExtractValue('<a>c</a><b/>', '//a') |
+-----+-----+
| c
+-----+-----+
1 row in set (0.00 sec)

```



Important

The replacement XML used as the third argument to `UpdateXML()` [1174] is *not* checked to determine whether it consists solely of elements which are properly nested and closed.

XPath Injection. *code injection* occurs when malicious code is introduced into the system to gain unauthorized access to privileges and data. It is based on exploiting assumptions made by developers about the type and content of data input from users. XPath is no exception in this regard.

A common scenario in which this can happen is the case of application which handles authorization by matching the combination of a login name and password with those found in an XML file, using an XPath expression like this one:

```
//user[login/text()='neapolitan' and password/text()='1c3cr34m']/attribute::id
```

This is the XPath equivalent of an SQL statement like this one:

```
SELECT id FROM users WHERE login='neapolitan' AND password='1c3cr34m';
```

A PHP application employing XPath might handle the login process like this:

```

<?php

$file      =   "users.xml";

$login     =   $_POST["login"];
$password  =   $_POST["password"];

$xmlpath = "//user[login/text()=$login and password/text()=$password]/attribute::id";

if( file_exists($file) )
{
    $xml = simplexml_load_file($file);

    if($result = $xml->xpath($xmlpath))
        echo "You are now logged in as user $result[0].";
    else
        echo "Invalid login name or password.";
}
else
    exit("Failed to open $file.");

?>

```

No checks are performed on the input. This means that a malevolent user can “short-circuit” the test by entering ' or 1=1 for both the login name and password, resulting in `$xmlpath` being evaluated as shown here:

```
//user[login/text()=' ' or 1=1 and password/text()=' ' or 1=1]/attribute::id
```

Since the expression inside the square brackets always evaluates as `true`, it is effectively the same as this one, which matches the `id` attribute of every `user` element in the XML document:

```
//user/attribute::id
```

One way in which this particular attack can be circumvented is simply by quoting the variable names to be interpolated in the definition of `$xpath`, forcing the values passed from a Web form to be converted to strings:

```
$xpath = "//user[login/text()='&#39;$login' and password/text()='&#39;$password']/attribute::id";
```

This is the same strategy that is often recommended for preventing SQL injection attacks. In general, the practices you should follow for preventing XPath injection attacks are the same as for preventing SQL injection:

- Never accepted untested data from users in your application.
- Check all user-submitted data for type; reject or convert data that is of the wrong type
- Test numeric data for out of range values; truncate, round, or reject values that are out of range. Test strings for illegal characters and either strip them out or reject input containing them.
- Do not output explicit error messages that might provide an unauthorized user with clues that could be used to compromise the system; log these to a file or database table instead.

Just as SQL injection attacks can be used to obtain information about database schemas, so can XPath injection be used to traverse XML files to uncover their structure, as discussed in Amit Klein's paper [Blind XPath Injection](#) (PDF file, 46KB).

It is also important to check the output being sent back to the client. Consider what can happen when we use the MySQL `ExtractValue()` [1173] function:

```
mysql> SELECT ExtractValue(
->   LOAD_FILE('users.xml'),
->   '//user[login/text()='&#39; or 1=1 and password/text()='&#39; or 1=1]/attribute::id'
-> ) AS id;
+-----+
| id      |
+-----+
| 00327 13579 02403 42354 28570 |
+-----+
1 row in set (0.01 sec)
```

Because `ExtractValue()` [1173] returns multiple matches as a single space-delimited string, this injection attack provides every valid ID contained within `users.xml` to the user as a single row of output. As an extra safeguard, you should also test output before returning it to the user. Here is a simple example:

```
mysql> SELECT @id = ExtractValue(
->   LOAD_FILE('users.xml'),
->   '//user[login/text()='&#39; or 1=1 and password/text()='&#39; or 1=1]/attribute::id'
-> );
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT IF(
->   INSTR(@id, ' ') = 0,
->   @id,
->   'Unable to retrieve user ID')
-> AS singleID;
+-----+
| singleID |
+-----+
| Unable to retrieve user ID |
+-----+
```

```
+-----+
1 row in set (0.00 sec)
```

In general, the guidelines for returning data to users securely are the same as for accepting user input. These can be summed up as:

- Always test outgoing data for type and permissible values.
- Never permit unauthorized users to view error messages that might provide information about the application that could be used to exploit it.

12.12. Bit Functions

Table 12.16. Bitwise Functions

Name	Description
<code>BIT_COUNT()</code> [1183]	Return the number of bits that are set
<code>&</code> [1182]	Bitwise AND
<code>~</code> [1183]	Invert bits
<code> </code> [1182]	Bitwise OR
<code>^</code> [1182]	Bitwise XOR
<code><<</code> [1183]	Left shift
<code>>></code> [1183]	Right shift

MySQL uses `BIGINT` (64-bit) arithmetic for bit operations, so these operators have a maximum range of 64 bits.

- `|` [1182]

Bitwise OR:

```
mysql> SELECT 29 | 15;
-> 31
```

The result is an unsigned 64-bit integer.

- `&` [1182]

Bitwise AND:

```
mysql> SELECT 29 & 15;
-> 13
```

The result is an unsigned 64-bit integer.

- `^` [1182]

Bitwise XOR:

```
mysql> SELECT 1 ^ 1;
-> 0
mysql> SELECT 1 ^ 0;
-> 1
mysql> SELECT 11 ^ 3;
-> 8
```

The result is an unsigned 64-bit integer.

- [<< \[1183\]](#)

Shifts a longlong (`BIGINT`) number to the left.

```
mysql> SELECT 1 << 2;
-> 4
```

The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

- [>> \[1183\]](#)

Shifts a longlong (`BIGINT`) number to the right.

```
mysql> SELECT 4 >> 2;
-> 1
```

The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

- [~ \[1183\]](#)

Invert all bits.

```
mysql> SELECT 5 & ~1;
-> 4
```

The result is an unsigned 64-bit integer.

- [BIT_COUNT\(N\) \[1183\]](#)

Returns the number of bits that are set in the argument *N*.

```
mysql> SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
-> 4, 3
```

12.13. Encryption and Compression Functions

Table 12.17. Encryption Functions

Name	Description
AES_DECRYPT() [1185]	Decrypt using AES
AES_ENCRYPT() [1185]	Encrypt using AES
COMPRESS() [1185]	Return result as a binary string
DECODE() [1186]	Decodes a string encrypted using ENCODE()
DES_DECRYPT() [1186]	Decrypt a string
DES_ENCRYPT() [1186]	Encrypt a string
ENCODE() [1187]	Encode a string
ENCRYPT() [1187]	Encrypt a string
MD5() [1187]	Calculate MD5 checksum
OLD_PASSWORD() [1187]	Return the value of the pre-4.1 implementation of PASSWORD
PASSWORD() [1188]	Calculate and return a password string
SHA1(), SHA() [1189]	Calculate an SHA-1 160-bit checksum
SHA2() [1189]	Calculate an SHA-2 checksum

Name	Description
UNCOMPRESS() [1189]	Uncompress a string compressed
UNCOMPRESSED_LENGTH() [1189]	Return the length of a string before compression

Many encryption and compression functions return strings for which the result might contain arbitrary byte values. If you want to store these results, use a column with a [VARBINARY](#) or [BLOB](#) binary string data type. This will avoid potential problems with trailing space removal or character set conversion that would change data values, such as may occur if you use a nonbinary string data type ([CHAR](#), [VARCHAR](#), [TEXT](#)).

Some encryption functions return strings of ASCII characters: [MD5\(\) \[1187\]](#), [OLD_PASSWORD\(\) \[1187\]](#), [PASSWORD\(\) \[1188\]](#), [SHA\(\) \[1189\]](#), [SHA1\(\) \[1189\]](#). As of MySQL 5.5.3, their return value is a nonbinary string that has a character set and collation determined by the [character_set_connection \[487\]](#) and [collation_connection \[489\]](#) system variables. Before 5.5.3, these functions return binary strings. The same change was made for [SHA2\(\) \[1189\]](#) in MySQL 5.5.6.

For versions in which functions such as [MD5\(\)](#) or [SHA1\(\)](#) return a string of hex digits as a binary string, the return value cannot be converted to uppercase or compared in case-insensitive fashion as is. You must convert the value to a nonbinary string. See the discussion of binary string conversion in [Section 12.10, “Cast Functions and Operators”](#).

If an application stores values from a function such as [MD5\(\) \[1187\]](#) or [SHA1\(\) \[1189\]](#) that returns a string of hex digits, more efficient storage and comparisons can be obtained by converting the hex representation to binary using [UNHEX\(\) \[1110\]](#) and storing the result in a [BINARY\(N\)](#) column. Each pair of hex digits requires one byte in binary form, so the value of *N* depends on the length of the hex string. *N* is 16 for an [MD5\(\) \[1187\]](#) value and 20 for a [SHA1\(\) \[1189\]](#) value. For [SHA2\(\) \[1189\]](#), *N* ranges from 28 to 32 depending on the argument specifying the desired bit length of the result.

The size penalty for storing the hex string in a [CHAR](#) column is at least two times, up to eight times if the value is stored in a column that uses the [utf8](#) character set (where each character uses 4 bytes). Storing the string also results in slower comparisons because of the larger values and the need to take character set collation rules into account.

Suppose that an application stores [MD5\(\) \[1187\]](#) string values in a [CHAR\(32\)](#) column:

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

To convert hex strings to more compact form, modify the application to use [UNHEX\(\) \[1110\]](#) and [BINARY\(16\)](#) instead as follows:

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

Applications should be prepared to handle the very rare case that a hashing function produces the same value for two different input values. One way to make collisions detectable is to make the hash column a primary key.



Note

Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using one of the other encryption functions described in this section instead, such as [SHA2\(\) \[1189\]](#).



Caution

Passwords or other sensitive values supplied as arguments to encryption functions are sent in plaintext to the MySQL server unless an SSL connection is used. Also,

such values will appear in any MySQL logs to which they are written. To avoid these types of exposure, applications can encrypt sensitive values on the client side before sending them to the server. The same considerations apply to encryption keys. To avoid exposing these, applications can use stored procedures to encrypt and decrypt values on the server side.

- [AES_DECRYPT\(*crypt_str*,*key_str*\)](#) [1185]

This function decrypts data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of [AES_ENCRYPT\(\)](#) [1185].

- [AES_ENCRYPT\(*str*,*key_str*\)](#) [1185]

[AES_ENCRYPT\(\)](#) [1185] and [AES_DECRYPT\(\)](#) [1185] enable encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” Encoding with a 128-bit key length is used, but you can extend it up to 256 bits by modifying the source. We chose 128 bits because it is much faster and it is secure enough for most purposes.

[AES_ENCRYPT\(\)](#) [1185] encrypts a string and returns a binary string. [AES_DECRYPT\(\)](#) [1185] decrypts the encrypted string and returns the original string. The input arguments may be any length. If either argument is `NULL`, the result of this function is also `NULL`.

Because AES is a block-level algorithm, padding is used to encode uneven length strings and so the result string length may be calculated using this formula:

```
16 * (trunc(string_length / 16) + 1)
```

If [AES_DECRYPT\(\)](#) [1185] detects invalid data or incorrect padding, it returns `NULL`. However, it is possible for [AES_DECRYPT\(\)](#) [1185] to return a non-`NULL` value (possibly garbage) if the input data or the key is invalid.

You can use the AES functions to store data in an encrypted form by modifying your queries:

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

[AES_ENCRYPT\(\)](#) [1185] and [AES_DECRYPT\(\)](#) [1185] can be considered the most cryptographically secure encryption functions currently available in MySQL.

- [COMPRESS\(*string_to_compress*\)](#) [1185]

Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`. The compressed string can be uncompressed with [UNCOMPRESS\(\)](#) [1189].

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
-> 21
mysql> SELECT LENGTH(COMPRESS(''));
-> 0
mysql> SELECT LENGTH(COMPRESS('a'));
-> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
-> 15
```

The compressed string contents are stored the following way:

- Empty strings are stored as empty strings.
- Nonempty strings are stored as a 4-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra “.” character is added to avoid problems

with endspace trimming should the result be stored in a [CHAR](#) or [VARCHAR](#) column. (However, use of nonbinary string data types such as [CHAR](#) or [VARCHAR](#) to store compressed strings is not recommended anyway because character set conversion may occur. Use a [VARBINARY](#) or [BLOB](#) binary string column instead.)

- [DECODE\(*crypt_str*,*pass_str*\)](#) [1186]

Decrypts the encrypted string *crypt_str* using *pass_str* as the password. *crypt_str* should be a string returned from [ENCODE\(\)](#) [1187].

- [DES_DECRYPT\(*crypt_str*\[,*key_str*\]\)](#) [1186]

Decrypts a string encrypted with [DES_ENCRYPT\(\)](#) [1186]. If an error occurs, this function returns [NULL](#).

This function works only if MySQL has been configured with SSL support. See [Section 6.3.8, “Using SSL for Secure Connections”](#).

If no *key_str* argument is given, [DES_DECRYPT\(\)](#) [1186] examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the [SUPER](#) [697] privilege. The key file can be specified with the [--des-key-file](#) [442] server option.

If you pass this function a *key_str* argument, that string is used as the key for decrypting the message.

If the *crypt_str* argument does not appear to be an encrypted string, MySQL returns the given *crypt_str*.

- [DES_ENCRYPT\(*str*\[,{*key_num*|*key_str*}\] \)](#) [1186]

Encrypts the string with the given key using the Triple-DES algorithm.

This function works only if MySQL has been configured with SSL support. See [Section 6.3.8, “Using SSL for Secure Connections”](#).

The encryption key to use is chosen based on the second argument to [DES_ENCRYPT\(\)](#) [1186], if one was given. With no argument, the first key from the DES key file is used. With a *key_num* argument, the given key number (0 to 9) from the DES key file is used. With a *key_str* argument, the given key string is used to encrypt *str*.

The key file can be specified with the [--des-key-file](#) [442] server option.

The return string is a binary string where the first character is [CHAR\(128 | *key_num*\)](#) [1101]. If an error occurs, [DES_ENCRYPT\(\)](#) [1186] returns [NULL](#).

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *key_num* is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each *key_num* value must be a number in the range from 0 to 9. Lines in the file may be in any order. *des_key_str* is the string that is used to encrypt the message. There should be at least one space

between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()` [1186].

You can tell MySQL to read new key values from the key file with the `FLUSH DES_KEY_FILE` statement. This requires the `RELOAD` [696] privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table
       > WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

- `ENCODE(str, pass_str)` [1187]

Encrypt `str` using `pass_str` as the password. To decrypt the result, use `DECODE()` [1186].

The result is a binary string of the same length as `str`.

The strength of the encryption is based on how good the random generator is. It should suffice for short strings.

- `ENCRYPT(str[, salt])` [1187]

Encrypts `str` using the Unix `crypt()` system call and returns a binary string. The `salt` argument must be a string with at least two characters or the result will be `NULL`. If no `salt` argument is given, a random value is used.

```
mysql> SELECT ENCRYPT('hello');
       -> 'VxuFAJXVARROc'
```

`ENCRYPT()` [1187] ignores all but the first eight characters of `str`, at least on some systems. This behavior is determined by the implementation of the underlying `crypt()` system call.

The use of `ENCRYPT()` [1187] with the `ucs2`, `utf16`, or `utf32` multi-byte character sets is not recommended because the system call expects a string terminated by a zero byte.

If `crypt()` is not available on your system (as is the case with Windows), `ENCRYPT()` [1187] always returns `NULL`.

- `MD5(str)` [1187]

Calculates an MD5 128-bit checksum for the string. The value is returned as a string of 32 hex digits, or `NULL` if the argument was `NULL`. The return value can, for example, be used as a hash key. See the notes at the beginning of this section about storing hash values efficiently.

As of MySQL 5.5.3, the return value is a nonbinary string in the connection character set. Before 5.5.3, the return value is a binary string; see the notes at the beginning of this section about using the value as a nonbinary string.

```
mysql> SELECT MD5('testing');
       -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

This is the “RSA Data Security, Inc. MD5 Message-Digest Algorithm.”

See the note regarding the MD5 algorithm at the beginning this section.

- `OLD_PASSWORD(str)` [1187]

`OLD_PASSWORD()` [1187] was added when the implementation of `PASSWORD()` [1188] was changed in MySQL 4.1 to improve security. `OLD_PASSWORD()` [1187] returns the value of the pre-4.1 implementation of `PASSWORD()` [1188] as a string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to your version 5.5 MySQL server without locking them out. See [Section 6.1.2.4, “Password Hashing in MySQL”](#).

As of MySQL 5.5.3, the return value is a nonbinary string in the connection character set. Before 5.5.3, the return value is a binary string.

- `PASSWORD(str)` [1188]

Calculates and returns a hashed password string from the plaintext password `str` and returns a nonbinary string in the connection character set (a binary string before MySQL 5.5.3), or `NULL` if the argument is `NULL`. This function is the SQL interface to the algorithm used by the server to encrypt MySQL passwords for storage in the `mysql.user` grant table.

The password hashing method used by `PASSWORD()` [1188] depends on the value of the `old_passwords` [534] system variable:

```
mysql> SET old_passwords = 0;
mysql> SELECT PASSWORD('mypass');
+-----+
| PASSWORD('mypass') |
+-----+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-----+

mysql> SET old_passwords = 1;
mysql> SELECT PASSWORD('mypass');
+-----+
| PASSWORD('mypass') |
+-----+
| 6f8c114b58f2ce9e |
+-----+
```

If `old_passwords=1`, `PASSWORD('str')` [1188] returns the same value as `OLD_PASSWORD('str')` [1187].

For descriptions of the permitted values of `old_passwords` [534], see [Section 5.1.4, “Server System Variables”](#).

Encryption performed by `PASSWORD()` [1188] is one-way (not reversible). It is not the same type of encryption as used for Unix passwords; for that, use `ENCRYPT()` [1187].



Note

The `PASSWORD()` [1188] function is used by the authentication system in MySQL Server; you should *not* use it in your own applications. For that purpose, consider `MD5()` [1187] or `SHA2()` [1189] instead. Also see [RFC 2195, section 2 \(Challenge-Response Authentication Mechanism \(CRAM\)\)](#), for more information about handling passwords and authentication securely in your applications.



Important

Statements that invoke `PASSWORD()` [1188] may be recorded in server logs or in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. See [Section 6.1.2, “Keeping Passwords Secure”](#).

- [SHA1\(*str*\) \[1189\]](#), [SHA\(*str*\) \[1189\]](#)

Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a string of 40 hex digits, or `NULL` if the argument was `NULL`. One of the possible uses for this function is as a hash key. See the notes at the beginning of this section about storing hash values efficiently. You can also use [SHA1\(\) \[1189\]](#) as a cryptographic function for storing passwords. [SHA\(\) \[1189\]](#) is synonymous with [SHA1\(\) \[1189\]](#).

As of MySQL 5.5.3, the return value is a nonbinary string in the connection character set. Before 5.5.3, the return value is a binary string; see the notes at the beginning of this section about using the value as a nonbinary string.

```
mysql> SELECT SHA1('abc');
-> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

[SHA1\(\) \[1189\]](#) can be considered a cryptographically more secure equivalent of [MD5\(\) \[1187\]](#). However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

- [SHA2\(*str*, *hash_length*\) \[1189\]](#)

Calculates the SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512). The first argument is the cleartext string to be hashed. The second argument indicates the desired bit length of the result, which must have a value of 224, 256, 384, 512, or 0 (which is equivalent to 256). If either argument is `NULL` or the hash length is not one of the permitted values, the return value is `NULL`. Otherwise, the function result is a hash value containing the desired number of bits. See the notes at the beginning of this section about storing hash values efficiently.

As of MySQL 5.5.6, the return value is a nonbinary string in the connection character set. Before 5.5.6, the return value is a binary string; see the notes at the beginning of this section about using the value as a nonbinary string.

```
mysql> SELECT SHA2('abc', 224);
-> '23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7'
```

This function works only if MySQL has been configured with SSL support. See [Section 6.3.8, “Using SSL for Secure Connections”](#).

[SHA2\(\) \[1189\]](#) can be considered cryptographically more secure than [MD5\(\) \[1187\]](#) or [SHA1\(\) \[1189\]](#).

[SHA2\(\)](#) was added in MySQL 5.5.5.

- [UNCOMPRESS\(*string_to_uncompress*\) \[1189\]](#)

Uncompresses a string compressed by the [COMPRESS\(\) \[1185\]](#) function. If the argument is not a compressed value, the result is `NULL`. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
-> 'any string'
mysql> SELECT UNCOMPRESS('any string');
-> NULL
```

- [UNCOMPRESSED_LENGTH\(*compressed_string*\) \[1189\]](#)

Returns the length that the compressed string had before being compressed.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
-> 30
```

12.14. Information Functions

Table 12.18. Information Functions

Name	Description
BENCHMARK() [1190]	Repeatedly execute an expression
CHARSET() [1191]	Return the character set of the argument
COERCIBILITY() [1191]	Return the collation coercibility value of the string argument
COLLATION() [1191]	Return the collation of the string argument
CONNECTION_ID() [1191]	Return the connection ID (thread ID) for the connection
CURRENT_USER() , CURRENT_USER [1192]	The authenticated user name and host name
DATABASE() [1193]	Return the default (current) database name
FOUND_ROWS() [1193]	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
LAST_INSERT_ID() [1194]	Value of the AUTOINCREMENT column for the last INSERT
ROW_COUNT() [1196]	The number of rows updated
SCHEMA() [1197]	A synonym for DATABASE()
SESSION_USER() [1197]	Synonym for USER()
SYSTEM_USER() [1197]	Synonym for USER()
USER() [1197]	The user name and host name provided by the client
VERSION() [1198]	Returns a string that indicates the MySQL server version

- [BENCHMARK\(count,expr\) \[1190\]](#)

The [BENCHMARK\(\) \[1190\]](#) function executes the expression *expr* repeatedly *count* times. It may be used to time how quickly MySQL processes the expression. The result value is always 0. The intended use is from within the `mysql` client, which reports query execution times:

```
mysql> SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+-----+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+-----+
|                                             0 |
+-----+
1 row in set (4.74 sec)
```

The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute [BENCHMARK\(\) \[1190\]](#) several times, and to interpret the result with regard to how heavily loaded the server machine is.

[BENCHMARK\(\) \[1190\]](#) is intended for measuring the runtime performance of scalar expressions, which has some significant implications for the way that you use it and interpret the results:

- Only scalar expressions can be used. Although the expression can be a subquery, it must return a single column and at most a single row. For example, [BENCHMARK\(10, \(SELECT * FROM t\)\) \[1190\]](#) will fail if the table *t* has more than one column or more than one row.
- Executing a `SELECT expr` statement *N* times differs from executing `SELECT BENCHMARK(N, expr)` in terms of the amount of overhead involved. The two have very different execution profiles

and you should not expect them to take the same amount of time. The former involves the parser, optimizer, table locking, and runtime evaluation *N* times each. The latter involves only runtime evaluation *N* times, and all the other components just once. Memory structures already allocated are reused, and runtime optimizations such as local caching of results already evaluated for aggregate functions can alter the results. Use of `BENCHMARK()` [1190] thus measures performance of the runtime component by giving more weight to that component and removing the “noise” introduced by the network, parser, optimizer, and so forth.

- `CHARSET(str)` [1191]

Returns the character set of the string argument.

```
mysql> SELECT CHARSET('abc');
      -> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
      -> 'utf8'
mysql> SELECT CHARSET(USER());
      -> 'utf8'
```

- `COERCIBILITY(str)` [1191]

Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
      -> 0
mysql> SELECT COERCIBILITY(USER());
      -> 3
mysql> SELECT COERCIBILITY('abc');
      -> 4
```

The return values have the meanings shown in the following table. Lower values have higher precedence.

Coercibility	Meaning	Example
0	Explicit collation	Value with <code>COLLATE</code> clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value, stored routine parameter or local variable
3	System constant	<code>USER()</code> [1197] return value
4	Coercible	Literal string
5	Ignorable	<code>NULL</code> or an expression derived from <code>NULL</code>

- `COLLATION(str)` [1191]

Returns the collation of the string argument.

```
mysql> SELECT COLLATION('abc');
      -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
      -> 'utf8_general_ci'
```

- `CONNECTION_ID()` [1191]

Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

```
mysql> SELECT CONNECTION_ID();
-> 23786
```

- [CURRENT_USER \[1192\]](#), [CURRENT_USER\(\) \[1192\]](#)

Returns the user name and host name combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges. The return value is a string in the `utf8` character set.

The value of [CURRENT_USER\(\) \[1192\]](#) can differ from the value of [USER\(\) \[1197\]](#).

```
mysql> SELECT USER();
-> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
-> '@localhost'
```

The example illustrates that although the client specified a user name of `davida` (as indicated by the value of the [USER\(\) \[1197\]](#) function), the server authenticated the client using an anonymous user account (as seen by the empty user name part of the [CURRENT_USER\(\) \[1192\]](#) value). One way this might occur is that there is no account listed in the grant tables for `davida`.

Within a stored program or view, [CURRENT_USER\(\) \[1192\]](#) returns the account for the user who defined the object (as given by its `DEFINER` value). For stored procedures and functions and views defined with the `SQL SECURITY INVOKER` characteristic, [CURRENT_USER\(\) \[1192\]](#) returns the object's invoker.

The following statements support use of the [CURRENT_USER\(\) \[1192\]](#) function to take the place of the name of (and, possibly, a host for) an affected user or a definer; in such cases, [CURRENT_USER\(\) \[1192\]](#) is expanded where and as needed:

- `DROP USER`
- `RENAME USER`
- `GRANT`
- `REVOKE`
- `CREATE FUNCTION`
- `CREATE PROCEDURE`
- `CREATE TRIGGER`
- `CREATE EVENT`
- `CREATE VIEW`
- `ALTER EVENT`
- `ALTER VIEW`
- `SET PASSWORD`

For information about the implications that this expansion of [CURRENT_USER\(\) \[1192\]](#) has for replication in different releases of MySQL 5.5, see [Section 16.4.1.6, “Replication of `CURRENT_USER\(\)`”](#).

- [DATABASE\(\) \[1193\]](#)

Returns the default (current) database name as a string in the `utf8` character set. If there is no default database, [DATABASE\(\) \[1193\]](#) returns `NULL`. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

```
mysql> SELECT DATABASE();
-> 'test'
```

If there is no default database, [DATABASE\(\) \[1193\]](#) returns `NULL`.

- [FOUND_ROWS\(\) \[1193\]](#)

A `SELECT` statement may include a `LIMIT` clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the `LIMIT`, but without running the statement again. To obtain this row count, include a `SQL_CALC_FOUND_ROWS` option in the `SELECT` statement, and then invoke [FOUND_ROWS\(\) \[1193\]](#) afterward:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
-> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

The second `SELECT` returns a number indicating how many rows the first `SELECT` would have returned had it been written without the `LIMIT` clause.

In the absence of the `SQL_CALC_FOUND_ROWS` option in the most recent successful `SELECT` statement, [FOUND_ROWS\(\) \[1193\]](#) returns the number of rows in the result set returned by that statement. If the statement includes a `LIMIT` clause, [FOUND_ROWS\(\) \[1193\]](#) returns the number of rows up to the limit. For example, [FOUND_ROWS\(\) \[1193\]](#) returns 10 or 60, respectively, if the statement includes `LIMIT 10` or `LIMIT 50, 10`.

The row count available through [FOUND_ROWS\(\) \[1193\]](#) is transient and not intended to be available past the statement following the `SELECT SQL_CALC_FOUND_ROWS` statement. If you need to refer to the value later, save it:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

If you are using `SELECT SQL_CALC_FOUND_ROWS`, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without `LIMIT`, because the result set need not be sent to the client.

`SQL_CALC_FOUND_ROWS` and [FOUND_ROWS\(\) \[1193\]](#) can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display containing links to the pages that show other sections of a search result. Using [FOUND_ROWS\(\) \[1193\]](#) enables you to determine how many other pages are needed for the rest of the result.

The use of `SQL_CALC_FOUND_ROWS` and [FOUND_ROWS\(\) \[1193\]](#) is more complex for `UNION` statements than for simple `SELECT` statements, because `LIMIT` may occur at multiple places in a `UNION`. It may be applied to individual `SELECT` statements in the `UNION`, or global to the `UNION` result as a whole.

The intent of `SQL_CALC_FOUND_ROWS` for `UNION` is that it should return the row count that would be returned without a global `LIMIT`. The conditions for use of `SQL_CALC_FOUND_ROWS` with `UNION` are:

- The `SQL_CALC_FOUND_ROWS` keyword must appear in the first `SELECT` of the `UNION`.
- The value of `FOUND_ROWS()` [1193] is exact only if `UNION ALL` is used. If `UNION` without `ALL` is used, duplicate removal occurs and the value of `FOUND_ROWS()` [1193] is only approximate.
- If no `LIMIT` is present in the `UNION`, `SQL_CALC_FOUND_ROWS` is ignored and returns the number of rows in the temporary table that is created to process the `UNION`.

Beyond the cases described here, the behavior of `FOUND_ROWS()` [1193] is undefined (for example, its value following a `SELECT` statement that fails with an error).



Important

`FOUND_ROWS()` [1193] is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

- `LAST_INSERT_ID()` [1194], `LAST_INSERT_ID(expr)` [1194]

`LAST_INSERT_ID()` [1194] (with no argument) returns a `BIGINT` (64-bit) value representing the first automatically generated value successfully inserted for an `AUTO_INCREMENT` column as a result of the most recently executed `INSERT` statement. The value of `LAST_INSERT_ID()` [1194] remains unchanged if no rows are successfully inserted.

For example, after inserting a row that generates an `AUTO_INCREMENT` value, you can get the value like this:

```
mysql> SELECT LAST_INSERT_ID();
-> 195
```

The currently executing statement does not affect the value of `LAST_INSERT_ID()` [1194]. Suppose that you generate an `AUTO_INCREMENT` value with one statement, and then refer to `LAST_INSERT_ID()` [1194] in a multiple-row `INSERT` statement that inserts rows into a table with its own `AUTO_INCREMENT` column. The value of `LAST_INSERT_ID()` [1194] will remain stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to `LAST_INSERT_ID()` [1194] and `LAST_INSERT_ID(expr)` [1194], the effect is undefined.)

If the previous statement returned an error, the value of `LAST_INSERT_ID()` [1194] is undefined. For transactional tables, if the statement is rolled back due to an error, the value of `LAST_INSERT_ID()` [1194] is left undefined. For manual `ROLLBACK`, the value of `LAST_INSERT_ID()` [1194] is not restored to that before the transaction; it remains as it was at the point of the `ROLLBACK`.

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` [1194] changes the same way as for statements executed outside the body of these kinds of objects. The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` [1194] that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()` [1194], the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements will not see a changed value.

The ID that was generated is maintained in the server on a *per-connection* basis. This means that the value returned by the function to a given client is the first `AUTO_INCREMENT` value generated for most

recent statement affecting an `AUTO_INCREMENT` column *by that client*. This value cannot be affected by other clients, even if they generate `AUTO_INCREMENT` values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

The value of `LAST_INSERT_ID()` [1194] is not changed if you set the `AUTO_INCREMENT` column of a row to a non-“magic” value (that is, a value that is not `NULL` and not 0).



Important

If you insert multiple rows using a single `INSERT` statement, `LAST_INSERT_ID()` [1194] returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same `INSERT` statement against some other server.

For example:

```
mysql> USE test;
Database changed
mysql> CREATE TABLE t (
  ->   id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  ->   name VARCHAR(10) NOT NULL
  -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t VALUES (NULL, 'Bob');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t;
+----+-----+
| id | name |
+----+-----+
|  1 | Bob  |
+----+-----+
1 row in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|                1 |
+-----+
1 row in set (0.00 sec)

mysql> INSERT INTO t VALUES
  -> (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t;
+----+-----+
| id | name |
+----+-----+
|  1 | Bob  |
|  2 | Mary |
|  3 | Jane |
|  4 | Lisa |
+----+-----+
4 rows in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
```

```
|          2 |
+-----+
1 row in set (0.00 sec)
```

Although the second `INSERT` statement inserted three new rows into `t`, the ID generated for the first of these rows was 2, and it is this value that is returned by `LAST_INSERT_ID()` [1194] for the following `SELECT` statement.

If you use `INSERT IGNORE` and the row is ignored, the `AUTO_INCREMENT` counter is not incremented and `LAST_INSERT_ID()` [1194] returns 0, which reflects that no row was inserted.

If `expr` is given as an argument to `LAST_INSERT_ID()` [1194], the value of the argument is returned by the function and is remembered as the next value to be returned by `LAST_INSERT_ID()` [1194]. This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

2. Use the table to generate sequence numbers like this:

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

The `UPDATE` statement increments the sequence counter and causes the next call to `LAST_INSERT_ID()` [1194] to return the updated value. The `SELECT` statement retrieves that value. The `mysql_insert_id()` C API function can also be used to get the value. See [Section 22.8.3.37](#), “`mysql_insert_id()`”.

You can generate sequences without calling `LAST_INSERT_ID()` [1194], but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the `UPDATE` statement and get their own sequence value with the `SELECT` statement (or `mysql_insert_id()`), without affecting or being affected by other clients that generate their own sequence values.

Note that `mysql_insert_id()` is only updated after `INSERT` and `UPDATE` statements, so you cannot use the C API function to retrieve the value for `LAST_INSERT_ID(expr)` [1194] after executing other SQL statements like `SELECT` or `SET`.

- [ROW_COUNT\(\)](#) [1196]

Before MySQL 5.5.5, `ROW_COUNT()` [1196] returns the number of rows changed, deleted, or inserted by the last statement if it was an `UPDATE`, `DELETE`, or `INSERT`. For other statements, the value may not be meaningful.

As of MySQL 5.5.5, `ROW_COUNT()` returns a value as follows:

- DDL statements: 0. This applies to statements such as `CREATE TABLE` or `DROP TABLE`.
- DML statements other than `SELECT`: The number of affected rows. This applies to statements such as `UPDATE`, `INSERT`, or `DELETE` (as before), but now also to statements such as `ALTER TABLE` and `LOAD DATA INFILE`.
- `SELECT`: -1 if the statement returns a result set, or the number of rows “affected” if it does not. For example, for `SELECT * FROM t1`, `ROW_COUNT()` [1196] returns -1. For `SELECT * FROM t1 INTO OUTFILE 'file_name'`, `ROW_COUNT()` [1196] returns the number of rows written to the file.

- `SIGNAL` statements: 0.

For `UPDATE` statements, the affected-rows value by default is the number of rows actually changed. If you specify the `CLIENT_FOUND_ROWS` flag to `mysql_real_connect()` when connecting to `mysqld`, the affected-rows value is the number of rows “found”; that is, matched by the `WHERE` clause.

For `REPLACE` statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For `INSERT ... ON DUPLICATE KEY UPDATE` statements, the affected-rows value is 1 if the row is inserted as a new row and 2 if an existing row is updated.

The `ROW_COUNT()` [1196] value is similar to the value from the `mysql_affected_rows()` C API function and the row count that the `mysql` client displays following statement execution.

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|          3 |
+-----+
1 row in set (0.00 sec)

mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|          2 |
+-----+
1 row in set (0.00 sec)
```



Important

`ROW_COUNT()` [1196] is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

- `SCHEMA()` [1197]

This function is a synonym for `DATABASE()` [1193].

- `SESSION_USER()` [1197]

`SESSION_USER()` [1197] is a synonym for `USER()` [1197].

- `SYSTEM_USER()` [1197]

`SYSTEM_USER()` [1197] is a synonym for `USER()` [1197].

- `USER()` [1197]

Returns the current MySQL user name and host name as a string in the `utf8` character set.

```
mysql> SELECT USER();
-> 'davida@localhost'
```

The value indicates the user name you specified when connecting to the server, and the client host from which you connected. The value can be different from that of `CURRENT_USER()` [1192].

You can extract only the user name part like this:

```
mysql> SELECT SUBSTRING_INDEX(USER(), '@', 1);
-> 'davida'
```

- `VERSION()` [1198]

Returns a string that indicates the MySQL server version. The string uses the `utf8` character set. The value might have a suffix in addition to the version number. See the description of the `version` [572] system variable in [Section 5.1.4, “Server System Variables”](#).

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.1, a warning is logged if you use this function when `binlog_format` [1899] is set to `STATEMENT`. (Bug #47995)

```
mysql> SELECT VERSION();
-> '5.5.31-standard'
```

12.15. Miscellaneous Functions

Table 12.19. Miscellaneous Functions

Name	Description
<code>DEFAULT()</code> [1198]	Return the default value for a table column
<code>GET_LOCK()</code> [1199]	Get a named lock
<code>INET_ATON()</code> [1199]	Return the numeric value of an IP address
<code>INET_NTOA()</code> [1200]	Return the IP address from a numeric value
<code>IS_FREE_LOCK()</code> [1200]	Checks whether the named lock is free
<code>IS_USED_LOCK()</code> [1200]	Checks whether the named lock is in use. Return connection identifier if true.
<code>MASTER_POS_WAIT()</code> [1200]	Block until the slave has read and applied all updates up to the specified position
<code>NAME_CONST()</code> [1201]	Causes the column to have the given name
<code>RAND()</code> [1128]	Return a random floating-point value
<code>RELEASE_LOCK()</code> [1201]	Releases the named lock
<code>SLEEP()</code> [1201]	Sleep for a number of seconds
<code>UUID_SHORT()</code> [1202]	Return an integer-valued universal identifier
<code>UUID()</code> [1201]	Return a Universal Unique Identifier (UUID)
<code>VALUES()</code> [1202]	Defines the values to be used during an INSERT

- `DEFAULT(col_name)` [1198]

Returns the default value for a table column. An error results if the column has no default value.

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

- `FORMAT(X,D)` [1103]

Formats the number *X* to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string. For details, see [Section 12.5, “String Functions”](#).

- `GET_LOCK(str,timeout)` [1199]

Tries to obtain a lock with a name given by the string *str*, using a timeout of *timeout* seconds. Returns 1 if the lock was obtained successfully, 0 if the attempt timed out (for example, because another client has previously locked the name), or NULL if an error occurred (such as running out of memory or the thread was killed with `mysqladmin kill`). If you have a lock obtained with `GET_LOCK()` [1199], it is released when you execute `RELEASE_LOCK()` [1201], execute a new `GET_LOCK()` [1199], or your connection terminates (either normally or abnormally). Locks obtained with `GET_LOCK()` [1199] do not interact with transactions. That is, committing a transaction does not release any such locks obtained during the transaction.

This function can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked by one client, `GET_LOCK()` [1199] blocks any request by another client for a lock with the same name. This enables clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also enables a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form `db_name.str` or `app_name.str`.

```
mysql> SELECT GET_LOCK('lock1',10);
-> 1
mysql> SELECT IS_FREE_LOCK('lock2');
-> 1
mysql> SELECT GET_LOCK('lock2',10);
-> 1
mysql> SELECT RELEASE_LOCK('lock2');
-> 1
mysql> SELECT RELEASE_LOCK('lock1');
-> NULL
```

The second `RELEASE_LOCK()` [1201] call returns NULL because the lock 'lock1' was automatically released by the second `GET_LOCK()` [1199] call.

If multiple clients are waiting for a lock, the order in which they will acquire it is undefined and depends on factors such as the thread library in use. In particular, applications should not assume that clients will acquire the lock in the same order that they issued the lock requests.



Note

Before MySQL 5.5.3, if a client attempts to acquire a lock that is already held by another client, it blocks according to the *timeout* argument. If the blocked client terminates, its thread does not die until the lock request times out.

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.1, a warning is logged if you use this function when `binlog_format` [1899] is set to `STATEMENT`. (Bug #47995)

- `INET_ATON(expr)` [1199]

Given the dotted-quad representation of an IPv4 network address as a string, returns an integer that represents the numeric value of the address in network byte order (big endian). `INET_ATON()` [1199] returns NULL if it does not understand its argument.

```
mysql> SELECT INET_ATON('10.0.5.9');
```

```
-> 167773449
```

For this example, the return value is calculated as $10 \times 256^3 + 0 \times 256^2 + 5 \times 256 + 9$.

`INET_ATON()` [1199] may or may not return a non-NULL result for short-form IP addresses (such as '127.1' as a representation of '127.0.0.1'). Because of this, `INET_ATON()` [1199] should not be used for such addresses.



Note

To store values generated by `INET_ATON()` [1199], use an `INT UNSIGNED` column rather than `INT`, which is signed. If you use a signed column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See [Section 11.2.6, “Out-of-Range and Overflow Handling”](#).

- `INET_NTOA(expr)` [1200]

Given a numeric IPv4 network address in network byte order, returns the dotted-quad representation of the address as a string. `INET_NTOA()` [1200] returns `NULL` if it does not understand its argument.

As of MySQL 5.5.3, the return value is a nonbinary string in the connection character set. Before 5.5.3, the return value is a binary string.

```
mysql> SELECT INET_NTOA(167773449);
-> '10.0.5.9'
```

- `IS_FREE_LOCK(str)` [1200]

Checks whether the lock named `str` is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument).

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.1, a warning is logged if you use this function when `binlog_format` [1899] is set to `STATEMENT`. (Bug #47995)

- `IS_USED_LOCK(str)` [1200]

Checks whether the lock named `str` is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns `NULL`.

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.1, a warning is logged if you use this function when `binlog_format` [1899] is set to `STATEMENT`. (Bug #47995)

- `MASTER_POS_WAIT(log_name, log_pos[, timeout])` [1200]

This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events the slave had to wait for to advance to the specified position. The function returns `NULL` if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns `-1` if the timeout has been exceeded. If the slave SQL thread stops while `MASTER_POS_WAIT()` [1200] is waiting, the function returns `NULL`. If the slave is past the specified position, the function returns immediately.

If a `timeout` value is specified, `MASTER_POS_WAIT()` [1200] stops waiting when `timeout` seconds have elapsed. `timeout` must be greater than 0; a zero or negative `timeout` means no timeout.

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.1, a warning is logged if you use this function when `binlog_format` [1899] is set to `STATEMENT`. (Bug #47995)

- `NAME_CONST(name, value)` [1201]

Returns the given value. When used to produce a result set column, `NAME_CONST()` [1201] causes the column to have the given name. The arguments should be constants.

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|      14 |
+-----+
```

This function is for internal use only. The server uses it when writing statements from stored programs that contain references to local program variables, as described in [Section 19.7, “Binary Logging of Stored Programs”](#). You might see this function in the output from `mysqlbinlog`.

- `RELEASE_LOCK(str)` [1201]

Releases the lock named by the string `str` that was obtained with `GET_LOCK()` [1199]. Returns 1 if the lock was released, 0 if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` [1199] or if it has previously been released.

The `DO` statement is convenient to use with `RELEASE_LOCK()` [1201]. See [Section 13.2.3, “DO Syntax”](#).

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.1, a warning is logged if you use this function when `binlog_format` [1899] is set to `STATEMENT`. (Bug #47995)

- `SLEEP(duration)` [1201]

Sleeps (pauses) for the number of seconds given by the `duration` argument, then returns 0. If `SLEEP()` [1201] is interrupted, it returns 1. The duration may have a fractional part given in microseconds.

This function is unsafe for statement-based replication. Beginning with MySQL 5.5.1, a warning is logged if you use this function when `binlog_format` [1899] is set to `STATEMENT`. (Bug #47995)

- `UUID()` [1201]

Returns a Universal Unique Identifier (UUID) generated according to “DCE 1.1: Remote Procedure Call” (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in October 1997 (Document Number C706, <http://www.opengroup.org/public/pubs/catalog/c706.htm>).

A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` [1201] are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

A UUID is a 128-bit number represented by a `utf8` string of five hexadecimal numbers in `aaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

- The first three numbers are generated from a timestamp.
- The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
- The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system).

In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have very low probability.

Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MySQL uses a randomly generated 48-bit number.

```
mysql> SELECT UUID();
-> '6ccd780c-baba-1026-9564-0040f4311e29'
```



Warning

Although `UUID()` [1201] values are intended to be unique, they are not necessarily unguessable or unpredictable. If unpredictability is required, UUID values should be generated some other way.



Note

`UUID()` [1201] does not work with statement-based replication.

- `UUID_SHORT()` [1202]

Returns a “short” universal identifier as a 64-bit unsigned integer (rather than a string-form 128-bit identifier as returned by the `UUID()` [1201] function).

The value of `UUID_SHORT()` [1202] is guaranteed to be unique if the following conditions hold:

- The `server_id` [551] of the current host is unique among your set of master and slave servers
- `server_id` [551] is between 0 and 255
- You do not set back your system time for your server between `mysqld` restarts
- You do not invoke `UUID_SHORT()` [1202] on average more than 16 million times per second between `mysqld` restarts

The `UUID_SHORT()` [1202] return value is constructed this way:

```
(server_id & 255) << 56
+ (server_startup_time_in_seconds << 24)
+ incremented_variable++;
```

```
mysql> SELECT UUID_SHORT();
-> 92395783831158784
```

Note that `UUID_SHORT()` [1202] does not work with statement-based replication.

- `VALUES(col_name)` [1202]

In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` [1202] function is meaningful only in the `ON DUPLICATE KEY UPDATE` clause of `INSERT` statements and returns `NULL` otherwise. See [Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
-> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

12.16. Functions and Modifiers for Use with `GROUP BY` Clauses

12.16.1. `GROUP BY` (Aggregate) Functions

Table 12.20. Aggregate (`GROUP BY`) Functions

Name	Description
<code>AVG()</code> [1203]	Return the average value of the argument
<code>BIT_AND()</code> [1204]	Return bitwise and
<code>BIT_OR()</code> [1204]	Return bitwise or
<code>BIT_XOR()</code> [1204]	Return bitwise xor
<code>COUNT(DISTINCT)</code> [1204]	Return the count of a number of different values
<code>COUNT()</code> [1204]	Return a count of the number of rows returned
<code>GROUP_CONCAT()</code> [1205]	Return a concatenated string
<code>MAX()</code> [1205]	Return the maximum value
<code>MIN()</code> [1206]	Return the minimum value
<code>STD()</code> [1206]	Return the population standard deviation
<code>STDDEV_POP()</code> [1206]	Return the population standard deviation
<code>STDDEV_SAMP()</code> [1206]	Return the sample standard deviation
<code>STDDEV()</code> [1206]	Return the population standard deviation
<code>SUM()</code> [1206]	Return the sum
<code>VAR_POP()</code> [1207]	Return the population standard variance
<code>VAR_SAMP()</code> [1207]	Return the sample variance
<code>VARIANCE()</code> [1207]	Return the population standard variance

This section describes group (aggregate) functions that operate on sets of values. Unless otherwise stated, group functions ignore `NULL` values.

If you use a group function in a statement containing no `GROUP BY` clause, it is equivalent to grouping on all rows. For more information, see [Section 12.16.3, “MySQL Extensions to `GROUP BY`”](#).

For numeric arguments, the variance and standard deviation functions return a `DOUBLE` value. The `SUM()` [1206] and `AVG()` [1203] functions return a `DECIMAL` value for exact-value arguments (integer or `DECIMAL`), and a `DOUBLE` value for approximate-value arguments (`FLOAT` or `DOUBLE`).

The `SUM()` [1206] and `AVG()` [1203] aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Functions such as `SUM()` [1206] or `AVG()` [1203] that expect a numeric argument cast the argument to a number if necessary. For `SET` or `ENUM` values, the cast operation causes the underlying numeric value to be used.

- `AVG([DISTINCT] expr)` [1203]

Returns the average value of *expr*. The `DISTINCT` option can be used to return the average of the distinct values of *expr*.

`AVG ()` [1203] returns `NULL` if there were no matching rows.

```
mysql> SELECT student_name, AVG(test_score)
->      FROM student
->      GROUP BY student_name;
```

- `BIT_AND(expr)` [1204]

Returns the bitwise `AND` of all bits in *expr*. The calculation is performed with 64-bit (`BIGINT`) precision.

This function returns `18446744073709551615` if there were no matching rows. (This is the value of an unsigned `BIGINT` value with all bits set to 1.)

- `BIT_OR(expr)` [1204]

Returns the bitwise `OR` of all bits in *expr*. The calculation is performed with 64-bit (`BIGINT`) precision.

This function returns `0` if there were no matching rows.

- `BIT_XOR(expr)` [1204]

Returns the bitwise `XOR` [1095] of all bits in *expr*. The calculation is performed with 64-bit (`BIGINT`) precision.

This function returns `0` if there were no matching rows.

- `COUNT(expr)` [1204]

Returns a count of the number of non-`NULL` values of *expr* in the rows retrieved by a `SELECT` statement. The result is a `BIGINT` value.

`COUNT ()` [1204] returns `0` if there were no matching rows.

```
mysql> SELECT student.student_name,COUNT(*)
->      FROM student,course
->      WHERE student.student_id=course.student_id
->      GROUP BY student_name;
```

`COUNT (*)` [1204] is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain `NULL` values.

`COUNT (*)` [1204] is optimized to return very quickly if the `SELECT` retrieves from one table, no other columns are retrieved, and there is no `WHERE` clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```

This optimization applies only to `MyISAM` tables only, because an exact row count is stored for this storage engine and can be accessed very quickly. For transactional storage engines such as `InnoDB`, storing an exact row count is more problematic because multiple transactions may be occurring, each of which may affect the count.

- `COUNT(DISTINCT expr, [expr...])` [1204]

Returns a count of the number of rows with different non-`NULL` *expr* values.

`COUNT(DISTINCT)` [1204] returns `0` if there were no matching rows.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

In MySQL, you can obtain the number of distinct expression combinations that do not contain `NULL` by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside `COUNT(DISTINCT ...)` [1204].

- `GROUP_CONCAT(expr)` [1205]

This function returns a string result with the concatenated non-`NULL` values from a group. It returns `NULL` if there are no non-`NULL` values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
              [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
->     GROUP_CONCAT(test_score)
->     FROM student
->     GROUP BY student_name;
```

Or:

```
mysql> SELECT student_name,
->     GROUP_CONCAT(DISTINCT test_score
->                   ORDER BY test_score DESC SEPARATOR ' ')
->     FROM student
->     GROUP BY student_name;
```

In MySQL, you can get the concatenated values of expression combinations. To eliminate duplicate values, use the `DISTINCT` clause. To sort values in the result, use the `ORDER BY` clause. To sort in reverse order, add the `DESC` (descending) keyword to the name of the column you are sorting by in the `ORDER BY` clause. The default is ascending order; this may be specified explicitly using the `ASC` keyword. The default separator between values in a group is comma (","). To specify a separator explicitly, use `SEPARATOR` followed by the string literal value that should be inserted between group values. To eliminate the separator altogether, specify `SEPARATOR ''`.

The result is truncated to the maximum length that is given by the `group_concat_max_len` [503] system variable, which has a default value of 1024. The value can be set higher, although the effective maximum length of the return value is constrained by the value of `max_allowed_packet` [519]. The syntax to change the value of `group_concat_max_len` [503] at runtime is as follows, where `val` is an unsigned integer:

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

The return value is a nonbinary or binary string, depending on whether the arguments are nonbinary or binary strings. The result type is `TEXT` or `BLOB` unless `group_concat_max_len` [503] is less than or equal to 512, in which case the result type is `VARCHAR` or `VARBINARY`.

See also `CONCAT()` [1102] and `CONCAT_WS()` [1102]: Section 12.5, "String Functions".

- `MAX([DISTINCT] expr)` [1205]

Returns the maximum value of `expr`. `MAX()` [1205] may take a string argument; in such cases, it returns the maximum string value. See Section 8.3.1, "How MySQL Uses Indexes". The `DISTINCT` keyword can be used to find the maximum of the distinct values of `expr`, however, this produces the same result as omitting `DISTINCT`.

`MAX()` [1205] returns `NULL` if there were no matching rows.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
->     FROM student
->     GROUP BY student_name;
```

For `MAX()` [1205], MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set. This differs from how `ORDER BY` compares them. This is expected to be rectified in a future MySQL release.

- `MIN([DISTINCT] expr)` [1206]

Returns the minimum value of `expr`. `MIN()` [1206] may take a string argument; in such cases, it returns the minimum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The `DISTINCT` keyword can be used to find the minimum of the distinct values of `expr`, however, this produces the same result as omitting `DISTINCT`.

`MIN()` [1206] returns `NULL` if there were no matching rows.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
->     FROM student
->     GROUP BY student_name;
```

For `MIN()` [1206], MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set. This differs from how `ORDER BY` compares them. This is expected to be rectified in a future MySQL release.

- `STD(expr)` [1206]

Returns the population standard deviation of `expr`. This is an extension to standard SQL. The standard SQL function `STDDEV_POP()` [1206] can be used instead.

This function returns `NULL` if there were no matching rows.

- `STDDEV(expr)` [1206]

Returns the population standard deviation of `expr`. This function is provided for compatibility with Oracle. The standard SQL function `STDDEV_POP()` [1206] can be used instead.

This function returns `NULL` if there were no matching rows.

- `STDDEV_POP(expr)` [1206]

Returns the population standard deviation of `expr` (the square root of `VAR_POP()` [1207]). You can also use `STD()` [1206] or `STDDEV()` [1206], which are equivalent but not standard SQL.

`STDDEV_POP()` [1206] returns `NULL` if there were no matching rows.

- `STDDEV_SAMP(expr)` [1206]

Returns the sample standard deviation of `expr` (the square root of `VAR_SAMP()` [1207]).

`STDDEV_SAMP()` [1206] returns `NULL` if there were no matching rows.

- `SUM([DISTINCT] expr)` [1206]

Returns the sum of `expr`. If the return set has no rows, `SUM()` [1206] returns `NULL`. The `DISTINCT` keyword can be used to sum only the distinct values of `expr`.

`SUM()` [1206] returns `NULL` if there were no matching rows.

- `VAR_POP(expr)` [1207]

Returns the population standard variance of `expr`. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. You can also use `VARIANCE()` [1207], which is equivalent but is not standard SQL.

`VAR_POP()` [1207] returns `NULL` if there were no matching rows.

- `VAR_SAMP(expr)` [1207]

Returns the sample variance of `expr`. That is, the denominator is the number of rows minus one.

`VAR_SAMP()` [1207] returns `NULL` if there were no matching rows.

- `VARIANCE(expr)` [1207]

Returns the population standard variance of `expr`. This is an extension to standard SQL. The standard SQL function `VAR_POP()` [1207] can be used instead.

`VARIANCE()` [1207] returns `NULL` if there were no matching rows.

12.16.2. GROUP BY Modifiers

The `GROUP BY` clause permits a `WITH ROLLUP` modifier that causes extra rows to be added to the summary output. These rows represent higher-level (or super-aggregate) summary operations. `ROLLUP` thus enables you to answer questions at multiple levels of analysis with a single query. It can be used, for example, to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a table named `sales` has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales
(
  year      INT NOT NULL,
  country   VARCHAR(20) NOT NULL,
  product   VARCHAR(32) NOT NULL,
  profit    INT
);
```

The table's contents can be summarized per year with a simple `GROUP BY` like this:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+-----+-----+
| year | SUM(profit) |
+-----+-----+
| 2000 |          4525 |
| 2001 |          3010 |
+-----+-----+
```

This output shows the total profit for each year, but if you also want to determine the total profit summed over all years, you must add up the individual values yourself or run an additional query.

Or you can use `ROLLUP`, which provides both levels of analysis with a single query. Adding a `WITH ROLLUP` modifier to the `GROUP BY` clause causes the query to produce another row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+-----+-----+
| year | SUM(profit) |
+-----+-----+
| 2000 |          4525 |
```

2001	3010
NULL	7535

The grand total super-aggregate line is identified by the value `NULL` in the `year` column.

`ROLLUP` has a more complex effect when there are multiple `GROUP BY` columns. In this case, each time there is a “break” (change in value) in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without `ROLLUP`, a summary on the `sales` table based on `year`, `country`, and `product` might look like this:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	India	Calculator	150
2000	India	Computer	1200
2000	USA	Calculator	75
2000	USA	Computer	1500
2001	Finland	Phone	10
2001	USA	Calculator	50
2001	USA	Computer	2700
2001	USA	TV	250

The output indicates summary values only at the year/country/product level of analysis. When `ROLLUP` is added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	Finland	NULL	1600
2000	India	Calculator	150
2000	India	Computer	1200
2000	India	NULL	1350
2000	USA	Calculator	75
2000	USA	Computer	1500
2000	USA	NULL	1575
2000	NULL	NULL	4525
2001	Finland	Phone	10
2001	Finland	NULL	10
2001	USA	Calculator	50
2001	USA	Computer	2700
2001	USA	TV	250
2001	USA	NULL	3000
2001	NULL	NULL	3010
NULL	NULL	NULL	7535

For this query, adding `ROLLUP` causes the output to include summary information at four levels of analysis, not just one. Here is how to interpret the `ROLLUP` output:

- Following each set of product rows for a given year and country, an extra summary row is produced showing the total for all products. These rows have the `product` column set to `NULL`.

- Following each set of rows for a given year, an extra summary row is produced showing the total for all countries and products. These rows have the `country` and `products` columns set to `NULL`.
- Finally, following all other rows, an extra summary row is produced showing the grand total for all years, countries, and products. This row has the `year`, `country`, and `products` columns set to `NULL`.

Other Considerations When using `ROLLUP`

The following items list some behaviors specific to the MySQL implementation of `ROLLUP`:

When you use `ROLLUP`, you cannot also use an `ORDER BY` clause to sort the results. In other words, `ROLLUP` and `ORDER BY` are mutually exclusive. However, you still have some control over sort order. `GROUP BY` in MySQL sorts results, and you can use explicit `ASC` and `DESC` keywords with columns named in the `GROUP BY` list to specify sort order for individual columns. (The higher-level summary rows added by `ROLLUP` still appear after the rows from which they are calculated, regardless of the sort order.)

`LIMIT` can be used to restrict the number of rows returned to the client. `LIMIT` is applied after `ROLLUP`, so the limit applies against the extra rows added by `ROLLUP`. For example:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP
-> LIMIT 5;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	Finland	NULL	1600
2000	India	Calculator	150
2000	India	Computer	1200

Using `LIMIT` with `ROLLUP` may produce results that are more difficult to interpret, because you have less context for understanding the super-aggregate rows.

The `NULL` indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the `GROUP BY` clause following the leftmost one that has changed value. For any column in the result set with a name that is a lexical match to any of those names, its value is set to `NULL`. (If you specify grouping columns by column number, the server identifies which columns to set to `NULL` by number.)

Because the `NULL` values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you cannot test them as `NULL` values within the query itself. For example, you cannot add `HAVING product IS NULL` to the query to eliminate from the output all but the super-aggregate rows.

On the other hand, the `NULL` values do appear as `NULL` on the client side and can be tested as such using any MySQL client programming interface.

12.16.3. MySQL Extensions to `GROUP BY`

In standard SQL, a query that includes a `GROUP BY` clause cannot refer to nonaggregated columns in the select list that are not named in the `GROUP BY` clause. For example, this query is illegal in standard SQL because the `name` column in the select list does not appear in the `GROUP BY`:

```
SELECT o.custid, c.name, MAX(o.payment)
FROM orders AS o, customers AS c
WHERE o.custid = c.custid
```

```
GROUP BY o.custid;
```

For the query to be legal, the `name` column must be omitted from the select list or named in the `GROUP BY` clause.

MySQL extends the use of `GROUP BY` so that the select list can refer to nonaggregated columns not named in the `GROUP BY` clause. This means that the preceding query is legal in MySQL. You can use this feature to get better performance by avoiding unnecessary column sorting and grouping. However, this is useful primarily when all values in each nonaggregated column not named in the `GROUP BY` are the same for each group. The server is free to choose any value from each group, so unless they are the same, the values chosen are indeterminate. Furthermore, the selection of values from each group cannot be influenced by adding an `ORDER BY` clause. Sorting of the result set occurs after values have been chosen, and `ORDER BY` does not affect which values the server chooses.

A similar MySQL extension applies to the `HAVING` clause. In standard SQL, a query that includes a `GROUP BY` clause cannot refer to nonaggregated columns in the `HAVING` clause that are not named in the `GROUP BY` clause. A MySQL extension permits references to such columns to simplify calculations. This extension assumes that the nongrouped columns will have the same group-wise values. Otherwise, the result is indeterminate.

To disable the MySQL `GROUP BY` extension, enable the `ONLY_FULL_GROUP_BY [615]` SQL mode. This enables standard SQL behavior: Columns not named in the `GROUP BY` clause cannot be used in the select list or `HAVING` clause unless enclosed in an aggregate function.

`ONLY_FULL_GROUP_BY [615]` also affects use of aliases in the `HAVING` clauses. For example, the following query returns `name` values that occur only once in table `orders`:

```
SELECT name, COUNT(name) FROM orders
GROUP BY name
HAVING COUNT(name) = 1;
```

MySQL extends this behavior to permit the use of an alias in the `HAVING` clause for the aggregated column:

```
SELECT name, COUNT(name) AS c FROM orders
GROUP BY name
HAVING c = 1;
```

Enabling `ONLY_FULL_GROUP_BY [615]` disables this MySQL extension and a `non-grouping field 'c' is used in HAVING clause` error occurs because the column `c` in the `HAVING` clause is not enclosed in an aggregate function (instead, it *is* an aggregate function).

The select list extension also applies to `ORDER BY`. That is, you can refer to nonaggregated columns in the `ORDER BY` clause that do not appear in the `GROUP BY` clause. (However, as mentioned previously, `ORDER BY` does not affect which values are chosen from nonaggregated columns; it only sorts them after they have been chosen.) This extension does not apply if the `ONLY_FULL_GROUP_BY [615]` SQL mode is enabled.

In some cases, you can use `MIN()` [1206] and `MAX()` [1205] to obtain a specific column value even if it is not unique. If the `sort` column contains integers no larger than 6 digits, the following query gives the value of `column` from the row containing the smallest `sort` value:

```
SUBSTR(MIN(CONCAT(LPAD(sort,6,'0'),column)),7)
```

See [Section 3.6.4, “The Rows Holding the Group-wise Maximum of a Certain Column”](#).

If you are trying to follow standard SQL, you cannot use expressions in `GROUP BY` clauses. As a workaround, use an alias for the expression:

```
SELECT id, FLOOR(value/100) AS val
FROM tbl_name
GROUP BY id, val;
```

MySQL permits expressions in `GROUP BY` clauses, so the alias is unnecessary:

```
SELECT id, FLOOR(value/100)
FROM tbl_name
GROUP BY id, FLOOR(value/100);
```

12.17. Spatial Extensions

MySQL supports spatial extensions to enable the generation, storage, and analysis of geographic features. These features are available for `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` tables.

For spatial columns, `MyISAM` supports both `SPATIAL` and non-`SPATIAL` indexes. Other storage engines support non-`SPATIAL` indexes, as described in [Section 13.1.13, “CREATE INDEX Syntax”](#).

This chapter covers the following topics:

- The basis of these spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

Additional Resources

- The Open Geospatial Consortium publishes the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at <http://www.opengis.org/docs/99-049.pdf>.
- If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: <http://forums.mysql.com/list.php?23>.

12.17.1. Introduction to MySQL Spatial Support

MySQL implements spatial extensions following the specification of the Open Geospatial Consortium (OGC). This is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. The OGC maintains a Web site at <http://www.opengis.org/>.

In 1997, the Open Geospatial Consortium published the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at <http://www.opengis.org/docs/99-049.pdf>. It contains additional information relevant to this chapter.

MySQL implements a subset of the **SQL with Geometry Types** environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describe a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, town district, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

This chapter uses all of these terms synonymously: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. Here, the term most commonly used is **geometry**, defined as *a point or an aggregate of points representing anything in the world that has a location*.

12.17.2. The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

12.17.2.1. The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- `Geometry` (noninstantiable)
 - `Point` (instantiable)
 - `Curve` (noninstantiable)
 - `LineString` (instantiable)
 - `Line`
 - `LinearRing`
 - `Surface` (noninstantiable)
 - `Polygon` (instantiable)
 - `GeometryCollection` (instantiable)
 - `MultiPoint` (instantiable)
 - `MultiCurve` (noninstantiable)
 - `MultiLineString` (instantiable)
 - `MultiSurface` (noninstantiable)
 - `MultiPolygon` (instantiable)

It is not possible to create objects in noninstantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

`Geometry` is the base class. It is an abstract class. The instantiable subclasses of `Geometry` are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base `Geometry` class has subclasses for `Point`, `Curve`, `Surface`, and `GeometryCollection`:

- `Point` represents zero-dimensional objects.
- `Curve` represents one-dimensional objects, and has subclass `LineString`, with sub-subclasses `Line` and `LinearRing`.
- `Surface` is designed for two-dimensional objects and has subclass `Polygon`.
- `GeometryCollection` has specialized zero-, one-, and two-dimensional collection classes named `MultiPoint`, `MultiLineString`, and `MultiPolygon` for modeling geometries corresponding to collections of `Points`, `LineStrings`, and `Polygons`, respectively. `MultiCurve` and `MultiSurface` are introduced as abstract superclasses that generalize the collection interfaces to handle `Curves` and `Surfaces`.

`Geometry`, `Curve`, `Surface`, `MultiCurve`, and `MultiSurface` are defined as noninstantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

`Point`, `LineString`, `Polygon`, `GeometryCollection`, `MultiPoint`, `MultiLineString`, and `MultiPolygon` are instantiable classes.

12.17.2.2. Class `Geometry`

`Geometry` is the root class of the hierarchy. It is a noninstantiable class but has a number of properties that are common to all geometry values created from any of the `Geometry` subclasses. These properties are described in the following list. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its **SRID**, or Spatial Reference Identifier. This value identifies the geometry's associated Spatial Reference System that describes the coordinate space in which the geometry object is defined.

In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

- Its **coordinates** in its Spatial Reference System, represented as double-precision (8-byte) numbers. All nonempty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geocentric** system (coordinates on the Earth's surface) are different things.

- Its **interior**, **boundary**, and **exterior**.

Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.

- Its **MBR** (Minimum Bounding Rectangle), or Envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

```
((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

- Whether the value is **simple** or **nonsimple**. Geometry values of types ([LineString](#), [MultiPoint](#), [MultiLineString](#)) are either simple or nonsimple. Each type determines its own assertions for being simple or nonsimple.
- Whether the value is **closed** or **not closed**. Geometry values of types ([LineString](#), [MultiString](#)) are either closed or not closed. Each type determines its own assertions for being closed or not closed.
- Whether the value is **empty** or **nonempty** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a [NULL](#) value). An empty geometry is defined to be always simple and has an area of 0.
- Its **dimension**. A geometry can have a dimension of -1, 0, 1, or 2:
 - -1 for an empty geometry.
 - 0 for a geometry with no length and no area.
 - 1 for a geometry with nonzero length and zero area.
 - 2 for a geometry with nonzero area.

[Point](#) objects have a dimension of zero. [LineString](#) objects have a dimension of 1. [Polygon](#) objects have a dimension of 2. The dimensions of [MultiPoint](#), [MultiLineString](#), and [MultiPolygon](#) objects are the same as the dimensions of the elements they consist of.

12.17.2.3. Class [Point](#)

A [Point](#) is a geometry that represents a single location in coordinate space.

[Point](#) Examples

- Imagine a large-scale map of the world with many cities. A [Point](#) object could represent each city.
- On a city map, a [Point](#) object could represent a bus stop.

[Point](#) Properties

- X-coordinate value.
- Y-coordinate value.
- [Point](#) is defined as a zero-dimensional geometry.
- The boundary of a [Point](#) is the empty set.

12.17.2.4. Class [Curve](#)

A [Curve](#) is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of [Curve](#) define the type of interpolation between points. [Curve](#) is a noninstantiable class.

Curve Properties

- A [Curve](#) has the coordinates of its points.
- A [Curve](#) is defined as a one-dimensional geometry.
- A [Curve](#) is simple if it does not pass through the same point twice.
- A [Curve](#) is closed if its start point is equal to its endpoint.
- The boundary of a closed [Curve](#) is empty.
- The boundary of a nonclosed [Curve](#) consists of its two endpoints.
- A [Curve](#) that is simple and closed is a [LinearRing](#).

12.17.2.5. Class [LineString](#)

A [LineString](#) is a [Curve](#) with linear interpolation between points.

[LineString](#) Examples

- On a world map, [LineString](#) objects could represent rivers.
- In a city map, [LineString](#) objects could represent streets.

[LineString](#) Properties

- A [LineString](#) has coordinates of segments, defined by each consecutive pair of points.
- A [LineString](#) is a [Line](#) if it consists of exactly two points.
- A [LineString](#) is a [LinearRing](#) if it is both closed and simple.

12.17.2.6. Class [Surface](#)

A [Surface](#) is a two-dimensional geometry. It is a noninstantiable class. Its only instantiable subclass is [Polygon](#).

[Surface](#) Properties

- A [Surface](#) is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple [Surface](#) as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple [Surface](#) is the set of closed curves corresponding to its exterior and interior boundaries.

12.17.2.7. Class [Polygon](#)

A [Polygon](#) is a planar [Surface](#) representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the [Polygon](#).

[Polygon](#) Examples

- On a region map, [Polygon](#) objects could represent forests, districts, and so on.

Polygon Assertions

- The boundary of a `Polygon` consists of a set of `LinearRing` objects (that is, `LineString` objects that are both simple and closed) that make up its exterior and interior boundaries.
- A `Polygon` has no rings that cross. The rings in the boundary of a `Polygon` may intersect at a `Point`, but only as a tangent.
- A `Polygon` has no lines, spikes, or punctures.
- A `Polygon` has an interior that is a connected point set.
- A `Polygon` may have holes. The exterior of a `Polygon` with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a `Polygon` a simple geometry.

12.17.2.8. Class `GeometryCollection`

A `GeometryCollection` is a geometry that is a collection of one or more geometries of any class.

All the elements in a `GeometryCollection` must be in the same Spatial Reference System (that is, in the same coordinate system). There are no other constraints on the elements of a `GeometryCollection`, although the subclasses of `GeometryCollection` described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a `MultiPoint` may contain only `Point` elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

12.17.2.9. Class `MultiPoint`

A `MultiPoint` is a geometry collection composed of `Point` elements. The points are not connected or ordered in any way.

`MultiPoint` Examples

- On a world map, a `MultiPoint` could represent a chain of small islands.
- On a city map, a `MultiPoint` could represent the outlets for a ticket office.

`MultiPoint` Properties

- A `MultiPoint` is a zero-dimensional geometry.
- A `MultiPoint` is simple if no two of its `Point` values are equal (have identical coordinate values).
- The boundary of a `MultiPoint` is the empty set.

12.17.2.10. Class `MultiCurve`

A `MultiCurve` is a geometry collection composed of `Curve` elements. `MultiCurve` is a noninstantiable class.

`MultiCurve` Properties

- A `MultiCurve` is a one-dimensional geometry.
- A `MultiCurve` is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A `MultiCurve` boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a `MultiCurve` if it is in the boundaries of an odd number of `MultiCurve` elements.
- A `MultiCurve` is closed if all of its elements are closed.
- The boundary of a closed `MultiCurve` is always empty.

12.17.2.11. Class `MultiLineString`

A `MultiLineString` is a `MultiCurve` geometry collection composed of `LineString` elements.

`MultiLineString` Examples

- On a region map, a `MultiLineString` could represent a river system or a highway system.

12.17.2.12. Class `MultiSurface`

A `MultiSurface` is a geometry collection composed of surface elements. `MultiSurface` is a noninstantiable class. Its only instantiable subclass is `MultiPolygon`.

`MultiSurface` Assertions

- Two `MultiSurface` surfaces have no interiors that intersect.
- Two `MultiSurface` elements have boundaries that intersect at most at a finite number of points.

12.17.2.13. Class `MultiPolygon`

A `MultiPolygon` is a `MultiSurface` object composed of `Polygon` elements.

`MultiPolygon` Examples

- On a region map, a `MultiPolygon` could represent a system of lakes.

`MultiPolygon` Assertions

- A `MultiPolygon` has no two `Polygon` elements with interiors that intersect.
- A `MultiPolygon` has no two `Polygon` elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A `MultiPolygon` may not have cut lines, spikes, or punctures. A `MultiPolygon` is a regular, closed point set.
- A `MultiPolygon` that has more than one `Polygon` has an interior that is not connected. The number of connected components of the interior of a `MultiPolygon` is equal to the number of `Polygon` values in the `MultiPolygon`.

`MultiPolygon` Properties

- A `MultiPolygon` is a two-dimensional geometry.

- A `MultiPolygon` boundary is a set of closed curves (`LineString` values) corresponding to the boundaries of its `Polygon` elements.
- Each `Curve` in the boundary of the `MultiPolygon` is in the boundary of exactly one `Polygon` element.
- Every `Curve` in the boundary of an `Polygon` element is in the boundary of the `MultiPolygon`.

12.17.3. Supported Spatial Data Formats

This section describes the standard spatial data formats that are used to represent geometry objects in queries. They are:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format.

12.17.3.1. Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form. For a Backus-Naur grammar that specifies the formal production rules for writing WKT values, see the OpenGIS specification document referenced in [Section 12.17, “Spatial Extensions”](#).

Examples of WKT representations of geometry objects:

- A `Point`:

```
POINT(15 20)
```

Note that point coordinates are specified with no separating comma. This differs from the syntax for the SQL `POINT()` [1223] function, which requires a comma between the coordinates. Take care to use the syntax appropriate to the context of a given spatial operation. For example, the following statements both extract the X-coordinate from a `Point` object. The first produces the object directly using the `POINT()` [1223] function. The second uses a WKT representation converted to a `Point` with `GeomFromText()`.

```
mysql> SELECT X(POINT(15, 20));
+-----+
| X(POINT(15, 20)) |
+-----+
|                15 |
+-----+

mysql> SELECT X(GeomFromText('POINT(15 20)'));
+-----+
| X(GeomFromText('POINT(15 20)')) |
+-----+
|                15 |
+-----+
```

- A `LineString` with four points:

```
LINestring(0 0, 10 10, 20 25, 50 60)
```

Note that point coordinate pairs are separated by commas.

- A `Polygon` with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A `MultiPoint` with three `Point` values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- A `MultiLineString` with two `LineString` values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- A `MultiPolygon` with two `Polygon` values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- A `GeometryCollection` consisting of two `Point` values and one `LineString`:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

12.17.3.2. Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation for geometric values is defined by the OpenGIS specification. It is also defined in the ISO *SQL/MM Part 3: Spatial* standard.

WKB is used to exchange geometry data as binary streams represented by `BLOB` values containing geometric WKB information.

WKB uses 1-byte unsigned integers, 4-byte unsigned integers, and 8-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to `POINT(1 1)` consists of this sequence of 21 bytes (each represented here by two hex digits):

```
010100000000000000000000F03F000000000000F03F
```

The sequence may be broken down into these components:

```
Byte order : 01
WKB type   : 01000000
X          : 00000000000000F03F
Y          : 00000000000000F03F
```

Component representation is as follows:

- The byte order may be either 1 or 0 to indicate little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.
- The WKB type is a code that indicates the geometry type. Values from 1 through 7 indicate `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`.
- A `Point` value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values are represented by more complex data structures, as detailed in the OpenGIS specification.

12.17.4. Creating a Spatially Enabled MySQL Database

This section describes the data types you can use for representing spatial data in MySQL, and the functions available for creating and retrieving spatial values.

12.17.4.1. MySQL Spatial Data Types

MySQL has data types that correspond to OpenGIS classes. Some of these types hold single geometry values:

- `GEOMETRY`
- `POINT`
- `LINestring`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types (`POINT`, `LINestring`, and `POLYGON`) restrict their values to a particular geometry type.

The other data types hold collections of values:

- `MULTIPOINT`
- `MULTILINestring`
- `MULTIPOLYGON`
- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` can store a collection of objects of any type. The other collection types (`MULTIPOINT`, `MULTILINestring`, `MULTIPOLYGON`, and `GEOMETRYCOLLECTION`) restrict collection members to those having a particular geometry type.

12.17.4.2. Creating Spatial Values

This section describes how to create spatial values using Well-Known Text and Well-Known Binary functions that are defined in the OpenGIS standard, and using MySQL-specific functions.

12.17.4.2.1. Creating Geometry Values Using WKT Functions

MySQL provides a number of functions that take as arguments a Well-Known Text representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromText()` [1220] accepts a WKT of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromText(wkt[,srid])` [1220],
`GeometryCollectionFromText(wkt[,srid])` [1220]

Constructs a `GEOMETRYCOLLECTION` value using its WKT representation and SRID.

- `GeomFromText(wkt[,srid])` [1220], `GeometryFromText(wkt[,srid])` [1220]

Constructs a geometry value of any type using its WKT representation and SRID.

- `LineFromText(wkt[,srid])` [1220], `LineStringFromText(wkt[,srid])` [1220]

Constructs a `LINestring` value using its WKT representation and SRID.

- `MLineFromText(wkt[,srid])` [1220], `MultiLineStringFromText(wkt[,srid])` [1220]

Constructs a `MULTILINESTRING` value using its WKT representation and SRID.

- `MPointFromText(wkt[,srid])` [1221], `MultiPointFromText(wkt[,srid])` [1221]

Constructs a `MULTIPOINT` value using its WKT representation and SRID.

- `MPolyFromText(wkt[,srid])` [1221], `MultiPolygonFromText(wkt[,srid])` [1221]

Constructs a `MULTIPOLYGON` value using its WKT representation and SRID.

- `PointFromText(wkt[,srid])` [1221]

Constructs a `POINT` value using its WKT representation and SRID.

- `PolyFromText(wkt[,srid])` [1221], `PolygonFromText(wkt[,srid])` [1221]

Constructs a `POLYGON` value using its WKT representation and SRID.

The OpenGIS specification also defines the following optional functions, which MySQL does not implement. These functions construct `Polygon` or `MultiPolygon` values based on the WKT representation of a collection of rings or closed `LineString` values. These values may intersect.

- `BdMPolyFromText(wkt,srid)` [1221]

Constructs a `MultiPolygon` value from a `MultiLineString` value in WKT format containing an arbitrary collection of closed `LineString` values.

- `BdPolyFromText(wkt,srid)` [1221]

Constructs a `Polygon` value from a `MultiLineString` value in WKT format containing an arbitrary collection of closed `LineString` values.

12.17.4.2.2. Creating Geometry Values Using WKB Functions

MySQL provides a number of functions that take as arguments a `BLOB` containing a Well-Known Binary representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

These functions also accept geometry objects for compatibility with the return value of the functions in [Section 12.17.4.2.3, “Creating Geometry Values Using MySQL-Specific Functions”](#). Thus, those functions may be used to provide the first argument to the functions in this section.

- `GeomCollFromWKB(wkb[,srid])` [1221], `GeometryCollectionFromWKB(wkb[,srid])` [1221]

Constructs a `GEOMETRYCOLLECTION` value using its WKB representation and SRID.

- `GeomFromWKB(wkb[,srid])` [1221], `GeometryFromWKB(wkb[,srid])` [1221]

Constructs a geometry value of any type using its WKB representation and SRID.

- `LineFromWKB(wkb[,srid])` [1221], `LineStringFromWKB(wkb[,srid])` [1221]

Constructs a `LINestring` value using its WKB representation and SRID.

- `MLineFromWKB(wkb[,srid])` [1221], `MultiLineStringFromWKB(wkb[,srid])` [1221]

Constructs a `MULTILINESTRING` value using its WKB representation and SRID.

- `MPointFromWKB(wkb[,srid])` [1222], `MultiPointFromWKB(wkb[,srid])` [1222]
Constructs a `MULTIPOINT` value using its WKB representation and SRID.
- `MPolyFromWKB(wkb[,srid])` [1222], `MultiPolygonFromWKB(wkb[,srid])` [1222]
Constructs a `MULTIPOLYGON` value using its WKB representation and SRID.
- `PointFromWKB(wkb[,srid])` [1222]
Constructs a `POINT` value using its WKB representation and SRID.
- `PolyFromWKB(wkb[,srid])` [1222], `PolygonFromWKB(wkb[,srid])` [1222]
Constructs a `POLYGON` value using its WKB representation and SRID.

The OpenGIS specification also describes optional functions for constructing `Polygon` or `MultiPolygon` values based on the WKB representation of a collection of rings or closed `LineString` values. These values may intersect. MySQL does not implement these functions:

- `BdMPolyFromWKB(wkb,srid)` [1222]
Constructs a `MultiPolygon` value from a `MultiLineString` value in WKB format containing an arbitrary collection of closed `LineString` values.
- `BdPolyFromWKB(wkb,srid)` [1222]
Constructs a `Polygon` value from a `MultiLineString` value in WKB format containing an arbitrary collection of closed `LineString` values.

12.17.4.2.3. Creating Geometry Values Using MySQL-Specific Functions

MySQL provides a set of useful nonstandard functions for creating geometry values. The functions described in this section are MySQL extensions to the OpenGIS specification.

These functions produce geometry objects from either WKB values or geometry objects as arguments. If any argument is not a proper WKB or geometry representation of the proper object type, the return value is `NULL`.

For example, you can insert the geometry return value from `Point()` [1223] directly into a `Point` column:

```
INSERT INTO t1 (pt_col) VALUES(Point(1,2));
```

- `GeometryCollection(g1,g2,...)` [1222]
Constructs a `GeometryCollection`.
- `LineString(pt1,pt2,...)` [1222]
Constructs a `LineString` value from a number of `Point` or WKB `Point` arguments. If the number of arguments is less than two, the return value is `NULL`.
- `MultiLineString(ls1,ls2,...)` [1222]
Constructs a `MultiLineString` value using `LineString` or WKB `LineString` arguments.
- `MultiPoint(pt1,pt2,...)` [1222]
Constructs a `MultiPoint` value using `Point` or WKB `Point` arguments.

- `MultiPolygon(poly1,poly2,...)` [1223]

Constructs a `MultiPolygon` value from a set of `Polygon` or WKB `Polygon` arguments.

- `Point(x,y)` [1223]

Constructs a `Point` using its coordinates.

- `Polygon(ls1,ls2,...)` [1223]

Constructs a `Polygon` value from a number of `LineString` or WKB `LineString` arguments. If any argument does not represent a `LinearRing` (that is, not a closed and simple `LineString`), the return value is `NULL`.

12.17.4.3. Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Currently, spatial columns are supported for `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` tables. See also the annotations about spatial indexes under [Section 12.17.6.1, “Creating Spatial Indexes”](#).

- Use the `CREATE TABLE` statement to create a table with a spatial column:

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the `ALTER TABLE` statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;
```

12.17.4.4. Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values into internal geometry format:

- Perform the conversion directly in the `INSERT` statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Perform the conversion prior to the `INSERT`:

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
```

```
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples all use `GeomFromText()` [1220] to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (PolygonFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

Note that if a client application program wants to use WKB representations of geometry values, it is responsible for sending correctly formed WKB in queries to the server. However, there are several ways of satisfying this requirement. For example:

- Inserting a `POINT(1 1)` value with hex literal syntax:

```
mysql> INSERT INTO geom VALUES
-> (GeomFromWKB(0x010100000000000000000000F03F0000000000F03F));
```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of `BLOB` type:

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See [Section 22.8.3.53](#), “`mysql_real_escape_string()`”.

12.17.4.5. Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them into WKT or WKB format.

- Fetching spatial data in internal format:

Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

- Fetching spatial data in WKT format:

The `AsText()` [1225] function converts a geometry from internal format into a WKT string.

```
SELECT AsText(g) FROM geom;
```

- Fetching spatial data in WKB format:

The `AsBinary()` [1225] function converts a geometry from internal format into a `BLOB` containing the WKB value.

```
SELECT AsBinary(g) FROM geom;
```

12.17.5. Spatial Analysis Functions

After populating spatial columns with values, you are ready to query and analyze them. MySQL provides a set of functions to perform various operations on spatial data. These functions can be grouped into four major categories according to the type of operation they perform:

- Functions that convert geometries between various formats
- Functions that provide access to qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

Spatial analysis functions can be used in many contexts, such as:

- Any interactive SQL program, such as `mysql`.
- Application programs written in any language that supports a MySQL client API

12.17.5.1. Geometry Format Conversion Functions

MySQL supports the following functions for converting geometry values between internal format and either WKT or WKB format:

- [AsBinary\(*g*\) \[1225\]](#), [AsWKB\(*g*\) \[1225\]](#)

Converts a value in internal geometry format to its WKB representation and returns the binary result.

```
SELECT AsBinary(g) FROM geom;
```

- [AsText\(*g*\) \[1225\]](#), [AsWKT\(*g*\) \[1225\]](#)

Converts a value in internal geometry format to its WKT representation and returns the string result.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+-----+
| AsText(GeomFromText(@g)) |
+-----+
| LINESTRING(1 1,2 2,3 3) |
+-----+
```

- [GeomFromText\(*wkt*\[, *srid*\]\) \[1220\]](#)

Converts a string value from its WKT representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as [PointFromText\(\) \[1221\]](#) and [LineFromText\(\) \[1220\]](#). See [Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”](#).

- [GeomFromWKB\(*wkb*\[, *srid*\]\) \[1221\]](#)

Converts a binary value from its WKB representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as [PointFromWKB\(\) \[1222\]](#) and [LineFromWKB\(\) \[1221\]](#). See [Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”](#).

12.17.5.2. Geometry Property Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such

functions return `NULL` if the argument is of an incorrect geometry type. For example, `Area()` [1229] returns `NULL` if the object type is neither `Polygon` nor `MultiPolygon`.

12.17.5.2.1. General Geometry Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- `Dimension(g)` [1226]

Returns the inherent dimension of the geometry value *g*. The result can be `-1`, `0`, `1`, or `2`. The meaning of these values is given in [Section 12.17.2.2, “Class Geometry”](#).

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| 1 |
+-----+
```

- `Envelope(g)` [1226]

Returns the Minimum Bounding Rectangle (MBR) for the geometry value *g*. The result is returned as a `Polygon` value.

The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-----+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| POLYGON((1 1,2 1,2 2,1 2,1 1)) |
+-----+
```

- `GeometryType(g)` [1226]

Returns as a binary string the name of the geometry type of which the geometry instance *g* is a member. The name corresponds to one of the instantiable `Geometry` subclasses.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----+
| POINT |
+-----+
```

- `SRID(g)` [1226]

Returns an integer indicating the Spatial Reference System ID for the geometry value *g*.

In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----+
| 101 |
+-----+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- [Boundary\(*g*\)](#) [1227]

Returns a geometry that is the closure of the combinatorial boundary of the geometry value *g*.

- [IsEmpty\(*g*\)](#) [1227]

This function is a placeholder that returns 0 for any valid geometry value, 1 for any invalid geometry value or `NULL`.

MySQL does not support GIS `EMPTY` values such as `POINT EMPTY`.

- [IsSimple\(*g*\)](#) [1227]

In MySQL 5.5, this function is a placeholder that always returns 0.

The description of each instantiable geometric class given earlier in the chapter includes the specific conditions that cause an instance of that class to be classified as not simple. (See [Section 12.17.2.1](#), “The Geometry Class Hierarchy”.)

12.17.5.2.2. Point Functions

A `Point` consists of X and Y coordinates, which may be obtained using the following functions:

- [X\(*p*\)](#) [1227]

Returns the X-coordinate value for the `Point` object *p* as a double-precision number.

```
mysql> SELECT X(POINT(56.7, 53.34));
+-----+
| X(POINT(56.7, 53.34)) |
+-----+
|                    56.7 |
+-----+
```

- [Y\(*p*\)](#) [1227]

Returns the Y-coordinate value for the `Point` object *p* as a double-precision number.

```
mysql> SELECT Y(POINT(56.7, 53.34));
+-----+
| Y(POINT(56.7, 53.34)) |
+-----+
|                    53.34 |
+-----+
```

12.17.5.2.3. LineString Functions

A `LineString` consists of `Point` values. You can extract particular points of a `LineString`, count the number of points that it contains, or obtain its length.

- [EndPoint\(*ls*\)](#) [1227]

Returns the `Point` that is the endpoint of the `LineString` value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-----+
| AsText(EndPoint(GeomFromText(@ls))) |
+-----+
| POINT(3 3) |
+-----+
```

- [GLength\(*ls*\)](#) [1228]

Returns as a double-precision number the length of the `LineString` value *ls* in its associated spatial reference.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+-----+
| GLength(GeomFromText(@ls)) |
+-----+
|                2.8284271247462 |
+-----+
```

`GLength()` [1228] is a nonstandard name. It corresponds to the OpenGIS `Length()` [1104] function.

- [NumPoints\(*ls*\)](#) [1228]

Returns the number of `Point` objects in the `LineString` value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+-----+
| NumPoints(GeomFromText(@ls)) |
+-----+
|                3 |
+-----+
```

- [PointN\(*ls*,*N*\)](#) [1228]

Returns the *N*-th `Point` in the `LineString` value *ls*. Points are numbered beginning with 1.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
+-----+
| AsText(PointN(GeomFromText(@ls),2)) |
+-----+
| POINT(2 2) |
+-----+
```

- [StartPoint\(*ls*\)](#) [1228]

Returns the `Point` that is the start point of the `LineString` value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+-----+
| AsText(StartPoint(GeomFromText(@ls))) |
+-----+
| POINT(1 1) |
+-----+
```

The OpenGIS specification also defines the following function, which MySQL does not implement:

- [IsRing\(*ls*\)](#) [1228]

Returns 1 if the `LineString` value *ls* is closed (that is, its `StartPoint()` [1228] and `EndPoint()` [1227] values are the same) and is simple (does not pass through the same point more than once). Returns 0 if *ls* is not a ring, and -1 if it is `NULL`.

12.17.5.2.4. MultiLineString Functions

These functions return properties of `MultiLineString` values.

- [GLength\(*m*ls\)](#) [1228]

Returns as a double-precision number the length of the `MultiLineString` value *m*ls. The length of *m*ls is equal to the sum of the lengths of its elements.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+-----+
| GLength(GeomFromText(@mls)) |
+-----+
|                4.2426406871193 |
+-----+
```

`GLength()` [1228] is a nonstandard name. It corresponds to the OpenGIS `Length()` [1104] function.

- [IsClosed\(*m*ls\)](#) [1229]

Returns 1 if the `MultiLineString` value *m*ls is closed (that is, the `StartPoint()` [1228] and `EndPoint()` [1227] values are the same for each `LineString` in *m*ls). Returns 0 if *m*ls is not closed, and -1 if it is NULL.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+-----+
| IsClosed(GeomFromText(@mls)) |
+-----+
|                0 |
+-----+
```

12.17.5.2.5. Polygon Functions

These functions return properties of `Polygon` values.

- [Area\(*poly*\)](#) [1229]

Returns as a double-precision number the area of the `Polygon` value *poly*, as measured in its spatial reference system.

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
mysql> SELECT Area(GeomFromText(@poly));
+-----+
| Area(GeomFromText(@poly)) |
+-----+
|                4 |
+-----+
```

- [ExteriorRing\(*poly*\)](#) [1229]

Returns the exterior ring of the `Polygon` value *poly* as a `LineString`.

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-----+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-----+
| LINESTRING(0 0,0 3,3 3,3 0,0 0) |
+-----+
```

- [InteriorRingN\(*poly*,*N*\)](#) [1229]

Returns the *N*-th interior ring for the `Polygon` value *poly* as a `LineString`. Rings are numbered beginning with 1.

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+-----+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+-----+
| LINESTRING(1 1,1 2,2 2,2 1,1 1) |
+-----+
```

- [NumInteriorRings\(*poly*\) \[1230\]](#)

Returns the number of interior rings in the [Polygon](#) value *poly*.

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+-----+
| NumInteriorRings(GeomFromText(@poly)) |
+-----+
| 1 |
+-----+
```

12.17.5.2.6. MultiPolygon Functions

These functions return properties of [MultiPolygon](#) values.

- [Area\(*mpoly*\) \[1229\]](#)

Returns as a double-precision number the area of the [MultiPolygon](#) value *mpoly*, as measured in its spatial reference system.

```
mysql> SET @mpoly =
-> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+-----+
| Area(GeomFromText(@mpoly)) |
+-----+
| 8 |
+-----+
```

- [Centroid\(*mpoly*\) \[1230\]](#)

Returns the mathematical centroid for the [MultiPolygon](#) value *mpoly* as a [Point](#). The result is not guaranteed to be on the [MultiPolygon](#).

```
mysql> SET @poly =
-> 'GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7,5 5)))';
mysql> SELECT GeometryType(@poly),AsText(Centroid(@poly));
+-----+
| GeometryType(@poly) | AsText(Centroid(@poly)) |
+-----+
| POLYGON | POINT(4.958333333333333 4.958333333333333) |
+-----+
```

The OpenGIS specification also defines the following function, which MySQL does not implement:

- [PointOnSurface\(*mpoly*\)](#)

Returns a [Point](#) value that is guaranteed to be on the [MultiPolygon](#) value *mpoly*.

12.17.5.2.7. GeometryCollection Functions

These functions return properties of [GeometryCollection](#) values.

- [GeometryN\(*gc*,*N*\) \[1231\]](#)

Returns the *N*-th geometry in the [GeometryCollection](#) value *gc*. Geometries are numbered beginning with 1.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+-----+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+-----+
| POINT(1 1) |
+-----+
```

- [NumGeometries\(*gc*\) \[1231\]](#)

Returns the number of geometries in the [GeometryCollection](#) value *gc*.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+-----+
| NumGeometries(GeomFromText(@gc)) |
+-----+
| 2 |
+-----+
```

12.17.5.3. Functions That Create New Geometries from Existing Ones

The following sections describe functions that take geometry values as arguments and return new geometry values.

12.17.5.3.1. Geometry Functions That Produce New Geometries

[Section 12.17.5.2, "Geometry Property Functions"](#), discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- [Envelope\(*g*\) \[1226\]](#)
- [StartPoint\(*ls*\) \[1228\]](#)
- [EndPoint\(*ls*\) \[1227\]](#)
- [PointN\(*ls*,*N*\) \[1228\]](#)
- [ExteriorRing\(*poly*\) \[1229\]](#)
- [InteriorRingN\(*poly*,*N*\) \[1229\]](#)
- [GeometryN\(*gc*,*N*\) \[1231\]](#)

12.17.5.3.2. Spatial Operators

OpenGIS proposes a number of other functions that can produce geometries. They are designed to implement spatial operators.

These functions are not implemented in MySQL.

- [Buffer\(*g*,*d*\) \[1231\]](#)

Returns a geometry that represents all points whose distance from the geometry value *g* is less than or equal to a distance of *d*.

- [ConvexHull\(*g*\)](#) [1232]

Returns a geometry that represents the convex hull of the geometry value *g*.

- [Difference\(*g1,g2*\)](#) [1232]

Returns a geometry that represents the point set difference of the geometry value *g1* with *g2*.

- [Intersection\(*g1,g2*\)](#) [1232]

Returns a geometry that represents the point set intersection of the geometry values *g1* with *g2*.

- [SymDifference\(*g1,g2*\)](#) [1232]

Returns a geometry that represents the point set symmetric difference of the geometry value *g1* with *g2*.

- [Union\(*g1,g2*\)](#) [1232]

Returns a geometry that represents the point set union of the geometry values *g1* and *g2*.

12.17.5.4. Functions for Testing Spatial Relations Between Geometric Objects

The functions described in these sections take two geometries as input parameters and return a qualitative or quantitative relation between them.

12.17.5.4.1. Relations on Geometry Minimal Bounding Rectangles (MBRs)

MySQL provides several functions that test relations between minimal bounding rectangles of two geometries *g1* and *g2*. The return values 1 and 0 indicate true and false, respectively.

- [MBRContains\(*g1,g2*\)](#) [1232]

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* contains the Minimum Bounding Rectangle of *g2*. This tests the opposite relationship as [MBRWithin\(\)](#) [1233].

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
+-----+-----+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

- [MBRDisjoint\(*g1,g2*\)](#) [1232]

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

- [MBREqual\(*g1,g2*\)](#) [1232]

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are the same.

- [MBRIntersects\(*g1,g2*\)](#) [1232]

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* intersect.

- [MBROverlaps\(*g1,g2*\)](#) [1232]

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* overlap. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `MBRTouches(g1,g2)` [1233]

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* touch. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `MBRWithin(g1,g2)` [1233]

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* is within the Minimum Bounding Rectangle of *g2*. This tests the opposite relationship as `MBRContains()` [1232].

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-----+-----+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

12.17.5.4.2. Functions That Test Spatial Relationships Between Geometries

The OpenGIS specification defines the following functions. They test the relationship between two geometry values *g1* and *g2*.

The return values 1 and 0 indicate true and false, respectively.



Note

Currently, MySQL does not implement these functions according to the specification. Those that are implemented return the same result as the corresponding MBR-based functions.

- `Contains(g1,g2)` [1233]

Returns 1 or 0 to indicate whether *g1* completely contains *g2*. This tests the opposite relationship as `Within()` [1234].

- `Crosses(g1,g2)` [1233]

Returns 1 if *g1* spatially crosses *g2*. Returns `NULL` if *g1* is a `Polygon` or a `MultiPolygon`, or if *g2* is a `Point` or a `MultiPoint`. Otherwise, returns 0.

The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

- The two geometries intersect
- Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
- Their intersection is not equal to either of the two given geometries

- `Disjoint(g1,g2)` [1233]

Returns 1 or 0 to indicate whether *g1* is spatially disjoint from (does not intersect) *g2*.

- `Equals(g1,g2)` [1234]

Returns 1 or 0 to indicate whether *g1* is spatially equal to *g2*.

- `Intersects(g1,g2)` [1234]

Returns 1 or 0 to indicate whether *g1* spatially intersects *g2*.

- `Overlaps(g1,g2)` [1234]

Returns 1 or 0 to indicate whether *g1* spatially overlaps *g2*. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `Touches(g1,g2)` [1234]

Returns 1 or 0 to indicate whether *g1* spatially touches *g2*. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `Within(g1,g2)` [1234]

Returns 1 or 0 to indicate whether *g1* is spatially within *g2*. This tests the opposite relationship as `Contains()` [1233].

12.17.6. Optimizing Spatial Analysis

For `MyISAM` tables, Search operations in nonspatial databases can be optimized using `SPATIAL` indexes. This is true for spatial databases as well. With the help of a great variety of multi-dimensional indexing methods that have previously been designed, it is possible to optimize spatial searches. The most typical of these are:

- Point queries that search for all objects that contain a given point
- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** for `SPATIAL` indexes on spatial columns. A `SPATIAL` index is built using the MBR of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. In a non-`SPATIAL` index, you must declare a prefix for any spatial column except for `POINT` columns.

`MyISAM` supports both `SPATIAL` and non-`SPATIAL` indexes. Other storage engines support non-`SPATIAL` indexes, as described in [Section 13.1.13, “CREATE INDEX Syntax”](#).

12.17.6.1. Creating Spatial Indexes

For `MyISAM` tables, MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but extended with the `SPATIAL` keyword. Currently, columns in spatial indexes must be declared `NOT NULL`. The following examples demonstrate how to create spatial indexes:

- With `CREATE TABLE`:

```
CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g)) ENGINE=MyISAM;
```

- With `ALTER TABLE`:

```
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- With `CREATE INDEX`:

```
CREATE SPATIAL INDEX sp_index ON geom (g);
```

For `MyISAM` tables, `SPATIAL INDEX` creates an R-tree index. For storage engines that support nonspatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values will be useful for exact-value lookups, but not for range scans.

For more information on indexing spatial columns, see [Section 13.1.13, “CREATE INDEX Syntax”](#).

To drop spatial indexes, use `ALTER TABLE` or `DROP INDEX`:

- With `ALTER TABLE`:

```
ALTER TABLE geom DROP INDEX g;
```

- With `DROP INDEX`:

```
DROP INDEX sp_index ON geom;
```

Example: Suppose that a table `geom` contains more than 32,000 geometries, which are stored in the column `g` of type `GEOMETRY`. The table also has an `AUTO_INCREMENT` column `fid` for storing object ID values.

```
mysql> DESCRIBE geom;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| fid   | int(11)   |      | PRI | NULL    | auto_increment |
| g     | geometry  |      |     |         |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+-----+
| count(*) |
+-----+
|      32376 |
+-----+
1 row in set (0.00 sec)
```

To add a spatial index on the column `g`, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376 Duplicates: 0 Warnings: 0
```

12.17.6.2. Using a Spatial Index

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as `MBRContains()` [1232] or `MBRWithin()` [1233] in the `WHERE` clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
           31000 15000,
           31000 16000,
           30000 16000,
           30000 15000))';
```

```
mysql> SELECT fid,AsText(g) FROM geom WHERE
-> MBRContains(GeomFromText(@poly),g);
+-----+-----+
| fid | AsText(g) |
+-----+-----+
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+-----+-----+
20 rows in set (0.00 sec)
```

Use [EXPLAIN](#) to check the way this query is executed:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
                31000 15000,
                31000 16000,
                30000 16000,
                30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
-> MBRContains(GeomFromText(@poly),g)\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: geom
         type: range
possible_keys: g
         key: g
        key_len: 32
         ref: NULL
         rows: 50
      Extra: Using where
1 row in set (0.00 sec)
```

Check what would happen without a spatial index:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
                31000 15000,
                31000 16000,
                30000 16000,
                30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
-> MBRContains(GeomFromText(@poly),g)\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: geom
         type: ALL
possible_keys: NULL
```

```

    key: NULL
  key_len: NULL
    ref: NULL
   rows: 32376
  Extra: Using where
1 row in set (0.00 sec)

```

Executing the `SELECT` statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```

mysql> SET @poly =
      -> 'Polygon((30000 15000,
                31000 15000,
                31000 16000,
                30000 16000,
                30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
      -> MBRContains(GeomFromText(@poly),g);
+-----+
| fid | AsText(g)
+-----+
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ...
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ...
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ...
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ...
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ...
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ...
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ...
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ...
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ...
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ...
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ...
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ...
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ...
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ...
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ...
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ...
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ...
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ...
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ...
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ...
+-----+
20 rows in set (0.46 sec)

```

12.17.7. MySQL Conformance and Compatibility

MySQL does not yet implement the following GIS features:

- Additional Metadata Views

OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function `Length()` [1104] on `LineString` and `MultiLineString` currently should be called in MySQL as `GLength()` [1228]

The problem is that there is an existing SQL function `Length()` [1104] that calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context. We need either to solve this somehow, or decide on another function name.

12.18. Precision Math

MySQL 5.5 provides support for precision math: numeric value handling that results in extremely accurate results and a high degree control over invalid values. Precision math is based on these two features:

- SQL modes that control how strict the server is about accepting or rejecting invalid data.
- The MySQL library for fixed-point arithmetic.

These features have several implications for numeric operations and provide a high degree of compliance with standard SQL:

- **Precise calculations:** For exact-value numbers, calculations do not introduce floating-point errors. Instead, exact precision is used. For example, MySQL treats a number such as `.0001` as an exact value rather than as an approximation, and summing it 10,000 times produces a result of exactly `1`, not a value that is merely “close” to 1.
- **Well-defined rounding behavior:** For exact-value numbers, the result of `ROUND()` [1129] depends on its argument, not on environmental factors such as how the underlying C library works.
- **Platform independence:** Operations on exact numeric values are the same across different platforms such as Windows and Unix.
- **Control over handling of invalid values:** Overflow and division by zero are detectable and can be treated as errors. For example, you can treat a value that is too large for a column as an error rather than having the value truncated to lie within the range of the column's data type. Similarly, you can treat division by zero as an error rather than as an operation that produces a result of `NULL`. The choice of which approach to take is determined by the setting of the server SQL mode.

The following discussion covers several aspects of how precision math works, including possible incompatibilities with older applications. At the end, some examples are given that demonstrate how MySQL 5.5 handles numeric operations precisely. For information about controlling the SQL mode, see [Section 5.1.7, “Server SQL Modes”](#).

12.18.1. Types of Numeric Values

The scope of precision math for exact-value operations includes the exact-value data types (`DECIMAL` and integer types) and exact-value numeric literals. Approximate-value data types and numeric literals are handled as floating-point numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

12.18.2. `DECIMAL` Data Type Changes

This section discusses the characteristics of the `DECIMAL` data type (and its synonyms) in MySQL 5.5, with particular regard to the following topics:

- Maximum number of digits
- Storage format
- Storage requirements
- The nonstandard MySQL extension to the upper range of `DECIMAL` columns

Possible incompatibilities with applications that are written for older versions of MySQL (prior to 5.0.3) are noted throughout this section.

The declaration syntax for a `DECIMAL` column is `DECIMAL(M,D)`. The ranges of values for the arguments in MySQL 5.5 are as follows:

- *M* is the maximum number of digits (the precision). It has a range of 1 to 65. (Older versions of MySQL permitted a range of 1 to 254.)
- *D* is the number of digits to the right of the decimal point (the scale). It has a range of 0 to 30 and must be no larger than *M*.

The maximum value of 65 for *M* means that calculations on `DECIMAL` values are accurate up to 65 digits. This limit of 65 digits of precision also applies to exact-value numeric literals, so the maximum range of such literals differs from before. (In older versions of MySQL, decimal values could have up to 254 digits. However, calculations were done using floating-point and thus were approximate, not exact.)

Values for `DECIMAL` columns in MySQL 5.5 are stored using a binary format that packs nine decimal digits into 4 bytes. The storage requirements for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires 4 bytes, and any remaining digits left over require some fraction of 4 bytes. The storage required for remaining digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1–2	1
3–4	2
5–6	3
7–9	4

For example, a `DECIMAL(18,9)` column has nine digits on either side of the decimal point, so the integer part and the fractional part each require 4 bytes. A `DECIMAL(20,6)` column has fourteen integer digits and six fractional digits. The integer digits require four bytes for nine of the digits and 3 bytes for the remaining five digits. The six fractional digits require 3 bytes.

Unlike some older versions of MySQL, `DECIMAL` columns in MySQL 5.5 do not store a leading `+` character or `-` character or leading `0` digits. If you insert `+0003.1` into a `DECIMAL(5,1)` column, it is stored as `3.1`. For negative numbers, a literal `-` character is not stored. Applications that rely on the older behavior must be modified to account for this change.

`DECIMAL` columns in MySQL 5.5 do not permit values larger than the range implied by the column definition. For example, a `DECIMAL(3,0)` column supports a range of `-999` to `999`. A `DECIMAL(M,D)` column permits at most $M - D$ digits to the left of the decimal point. This is not compatible with applications relying on older versions of MySQL that permitted storing an extra digit in lieu of a `+` sign.

The SQL standard requires that the precision of `NUMERIC(M,D)` be *exactly* *M* digits. For `DECIMAL(M,D)`, the standard requires a precision of at least *M* digits but permits more. In MySQL, `DECIMAL(M,D)` and `NUMERIC(M,D)` are the same, and both have a precision of exactly *M* digits.

For more detailed information about porting applications that rely on the old treatment of the `DECIMAL` data type, see the *MySQL 5.0 Reference Manual*.

12.18.3. Expression Handling

With precision math, exact-value numbers are used as given whenever possible. For example, numbers in comparisons are used exactly as given without a change in value. In strict SQL mode, for `INSERT` into a column with an exact data type (`DECIMAL` or integer), a number is inserted with its exact value if it is within the column range. When retrieved, the value should be the same as what was inserted. (Without strict mode, truncation for `INSERT` is permissible.)

Handling of a numeric expression depends on what kind of values the expression contains:

- If any approximate values are present, the expression is approximate and is evaluated using floating-point arithmetic.
- If no approximate values are present, the expression contains only exact values. If any exact value contains a fractional part (a value following the decimal point), the expression is evaluated using `DECIMAL` exact arithmetic and has a precision of 65 digits. The term “exact” is subject to the limits of what can be represented in binary. For example, $1.0/3.0$ can be approximated in decimal notation as $.333\dots$, but not written as an exact number, so $(1.0/3.0)*3.0$ does not evaluate to exactly 1.0 .
- Otherwise, the expression contains only integer values. The expression is exact and is evaluated using integer arithmetic and has a precision the same as `BIGINT` (64 bits).

If a numeric expression contains any strings, they are converted to double-precision floating-point values and the expression is approximate.

Inserts into numeric columns are affected by the SQL mode, which is controlled by the `sql_mode` [556] system variable. (See [Section 5.1.7, “Server SQL Modes”](#).) The following discussion mentions strict mode (selected by the `STRICT_ALL_TABLES` [616] or `STRICT_TRANS_TABLES` [617] mode values) and `ERROR_FOR_DIVISION_BY_ZERO` [613]. To turn on all restrictions, you can simply use `TRADITIONAL` [618] mode, which includes both strict mode values and `ERROR_FOR_DIVISION_BY_ZERO` [613]:

```
mysql> SET sql_mode='TRADITIONAL';
```

If a number is inserted into an exact type column (`DECIMAL` or integer), it is inserted with its exact value if it is within the column range.

If the value has too many digits in the fractional part, rounding occurs and a warning is generated. Rounding is done as described in [Section 12.18.4, “Rounding Behavior”](#).

If the value has too many digits in the integer part, it is too large and is handled as follows:

- If strict mode is not enabled, the value is truncated to the nearest legal value and a warning is generated.
- If strict mode is enabled, an overflow error occurs.

Underflow is not detected, so underflow handling is undefined.

By default, division by zero produces a result of `NULL` and no warning. With the `ERROR_FOR_DIVISION_BY_ZERO` [613] SQL mode enabled, MySQL handles division by zero differently:

- If strict mode is not enabled, a warning occurs.
- If strict mode is enabled, inserts and updates involving division by zero are prohibited, and an error occurs.

In other words, inserts and updates involving expressions that perform division by zero can be treated as errors, but this requires `ERROR_FOR_DIVISION_BY_ZERO` [613] in addition to strict mode.

Suppose that we have this statement:

```
INSERT INTO t SET i = 1/0;
```

This is what happens for combinations of strict and `ERROR_FOR_DIVISION_BY_ZERO` [613] modes.

<code>sql_mode</code> [556] Value	Result
' ' (Default)	No warning, no error; <code>i</code> is set to <code>NULL</code> .
strict	No warning, no error; <code>i</code> is set to <code>NULL</code> .
<code>ERROR_FOR_DIVISION_BY_ZERO</code> [613]	Warning, no error; <code>i</code> is set to <code>NULL</code> .
strict, <code>ERROR_FOR_DIVISION_BY_ZERO</code> [613]	Error condition; no row is inserted.

For inserts of strings into numeric columns, conversion from string to number is handled as follows if the string has nonnumeric contents:

- A string that does not begin with a number cannot be used as a number and produces an error in strict mode, or a warning otherwise. This includes the empty string.
- A string that begins with a number can be converted, but the trailing nonnumeric portion is truncated. If the truncated portion contains anything other than spaces, this produces an error in strict mode, or a warning otherwise.

12.18.4. Rounding Behavior

This section discusses precision math rounding for the `ROUND()` [1129] function and for inserts into columns with exact-value types (`DECIMAL` and integer).

The `ROUND()` [1129] function rounds differently depending on whether its argument is exact or approximate:

- For exact-value numbers, `ROUND()` [1129] uses the “round half up” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` [1129] uses the “round to nearest even” rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3          | 2            |
+-----+-----+
```

For inserts into a `DECIMAL` or integer column, the target is an exact data type, so rounding uses “round half away from zero,” regardless of whether the value to be inserted is exact or approximate:

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 2

mysql> SELECT d FROM t;
+-----+
| d      |
+-----+
| 3      |
| 3      |
+-----+
```

12.18.5. Precision Math Examples

This section provides some examples that show precision math query results in MySQL 5.5. These examples demonstrate the principles described in [Section 12.18.3, “Expression Handling”](#), and [Section 12.18.4, “Rounding Behavior”](#).

Example 1. Numbers are used with their exact value as given when possible:

```
mysql> SELECT (.1 + .2) = .3;
+-----+
| (.1 + .2) = .3 |
+-----+
| 1              |
+-----+
```

For floating-point values, results are inexact:

```
mysql> SELECT (.1E0 + .2E0) = .3E0;
+-----+
| (.1E0 + .2E0) = .3E0 |
+-----+
| 0                      |
+-----+
```

Another way to see the difference in exact and approximate value handling is to add a small number to a sum many times. Consider the following stored procedure, which adds `.0001` to a variable 1,000 times.

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
  DECLARE f FLOAT DEFAULT 0;
  WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
    SET i = i + 1;
  END WHILE;
  SELECT d, f;
END;
```

The sum for both `d` and `f` logically should be 1, but that is true only for the decimal calculation. The floating-point calculation introduces small errors:

```
+-----+-----+
| d      | f      |
+-----+-----+
| 1.0000 | 0.999999999999991 |
+-----+-----+
```

Example 2. Multiplication is performed with the scale required by standard SQL. That is, for two numbers `X1` and `X2` that have scale `S1` and `S2`, the scale of the result is `S1 + S2`:

```
mysql> SELECT .01 * .01;
+-----+
| .01 * .01 |
+-----+
| 0.0001    |
+-----+
```

Example 3. Rounding behavior for exact-value numbers is well-defined:

Rounding behavior (for example, with the `ROUND()` [1129] function) is independent of the implementation of the underlying C library, which means that results are consistent from platform to platform.

- Rounding for exact-value columns (`DECIMAL` and integer) and exact-valued numbers uses the “round half away from zero” rule. Values with a fractional part of .5 or greater are rounded away from zero to the nearest integer, as shown here:

```
mysql> SELECT ROUND(2.5), ROUND(-2.5);
+-----+-----+
| ROUND(2.5) | ROUND(-2.5) |
+-----+-----+
| 3          | -3          |
+-----+-----+
```

- Rounding for floating-point values uses the C library, which on many systems uses the “round to nearest even” rule. Values with any fractional part on such systems are rounded to the nearest even integer:

```
mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
+-----+-----+
| ROUND(2.5E0) | ROUND(-2.5E0) |
+-----+-----+
| 2            | -2            |
+-----+-----+
```

Example 4. In strict mode, inserting a value that is out of range for a column causes an error, rather than truncation to a legal value.

When MySQL is not running in strict mode, truncation to a legal value occurs:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT i FROM t;
+-----+
| i     |
+-----+
| 127   |
+-----+
1 row in set (0.00 sec)
```

However, an error occurs if strict mode is in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
```

```
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1
```

```
mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

Example 5: In strict mode and with `ERROR_FOR_DIVISION_BY_ZERO` [613] set, division by zero causes an error, not a result of `NULL`.

In nonstrict mode, division by zero has a result of `NULL`:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT i FROM t;
+-----+
| i     |
+-----+
| NULL  |
+-----+
1 row in set (0.03 sec)
```

However, division by zero is an error if the proper SQL modes are in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

Example 6. Exact-value literals are evaluated as exact values.

Prior to MySQL 5.0.3, exact-value and approximate-value literals both are evaluated as double-precision floating-point values:

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 4.1.18-log |
+-----+
1 row in set (0.01 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.07 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> DESCRIBE t;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| a     | double(3,1)  |      |     | 0.0     |      |
| b     | double       |      |     | 0       |      |
+-----+-----+-----+-----+-----+-----+-----+
```

```
2 rows in set (0.04 sec)
```

As of MySQL 5.0.3, the approximate-value literal is evaluated using floating point, but the exact-value literal is handled as [DECIMAL](#):

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 5.1.6-alpha-log |
+-----+
1 row in set (0.11 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.01 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> DESCRIBE t;
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| a     | decimal(2,1) unsigned | NO   |     | 0.0     |       |
| b     | double              | NO   |     | 0       |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

Example 7. If the argument to an aggregate function is an exact numeric type, the result is also an exact numeric type, with a scale at least that of the argument.

Consider these statements:

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

Before MySQL 5.0.3, the result is a double no matter the argument type:

```
mysql> DESCRIBE y;
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| AVG(i) | double(17,4)       | YES  |     | NULL    |       |
| AVG(d) | double(17,4)       | YES  |     | NULL    |       |
| AVG(f) | double             | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

As of MySQL 5.0.3, the result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type:

```
mysql> DESCRIBE y;
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| AVG(i) | decimal(14,4)      | YES  |     | NULL    |       |
| AVG(d) | decimal(14,4)      | YES  |     | NULL    |       |
| AVG(f) | double             | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

The result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type.

Chapter 13. SQL Statement Syntax

Table of Contents

13.1. Data Definition Statements	1248
13.1.1. ALTER DATABASE Syntax	1248
13.1.2. ALTER EVENT Syntax	1249
13.1.3. ALTER LOGFILE GROUP Syntax	1251
13.1.4. ALTER FUNCTION Syntax	1252
13.1.5. ALTER PROCEDURE Syntax	1252
13.1.6. ALTER SERVER Syntax	1253
13.1.7. ALTER TABLE Syntax	1253
13.1.8. ALTER TABLESPACE Syntax	1270
13.1.9. ALTER VIEW Syntax	1271
13.1.10. CREATE DATABASE Syntax	1272
13.1.11. CREATE EVENT Syntax	1272
13.1.12. CREATE FUNCTION Syntax	1277
13.1.13. CREATE INDEX Syntax	1277
13.1.14. CREATE LOGFILE GROUP Syntax	1281
13.1.15. CREATE PROCEDURE and CREATE FUNCTION Syntax	1283
13.1.16. CREATE SERVER Syntax	1288
13.1.17. CREATE TABLE Syntax	1289
13.1.18. CREATE TABLESPACE Syntax	1312
13.1.19. CREATE TRIGGER Syntax	1313
13.1.20. CREATE VIEW Syntax	1317
13.1.21. DROP DATABASE Syntax	1321
13.1.22. DROP EVENT Syntax	1322
13.1.23. DROP FUNCTION Syntax	1322
13.1.24. DROP INDEX Syntax	1322
13.1.25. DROP LOGFILE GROUP Syntax	1323
13.1.26. DROP PROCEDURE and DROP FUNCTION Syntax	1323
13.1.27. DROP SERVER Syntax	1323
13.1.28. DROP TABLE Syntax	1323
13.1.29. DROP TABLESPACE Syntax	1324
13.1.30. DROP TRIGGER Syntax	1324
13.1.31. DROP VIEW Syntax	1325
13.1.32. RENAME TABLE Syntax	1325
13.1.33. TRUNCATE TABLE Syntax	1326
13.2. Data Manipulation Statements	1327
13.2.1. CALL Syntax	1327
13.2.2. DELETE Syntax	1329
13.2.3. DO Syntax	1333
13.2.4. HANDLER Syntax	1333
13.2.5. INSERT Syntax	1334
13.2.6. LOAD DATA INFILE Syntax	1342
13.2.7. LOAD XML Syntax	1351
13.2.8. REPLACE Syntax	1357
13.2.9. SELECT Syntax	1358
13.2.10. Subquery Syntax	1377
13.2.11. UPDATE Syntax	1388
13.3. MySQL Transactional and Locking Statements	1390
13.3.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax	1390

13.3.2. Statements That Cannot Be Rolled Back	1392
13.3.3. Statements That Cause an Implicit Commit	1392
13.3.4. <code>SAVEPOINT</code> , <code>ROLLBACK TO SAVEPOINT</code> , and <code>RELEASE SAVEPOINT</code> Syntax	1393
13.3.5. <code>LOCK TABLES</code> and <code>UNLOCK TABLES</code> Syntax	1394
13.3.6. <code>SET TRANSACTION</code> Syntax	1400
13.3.7. XA Transactions	1402
13.4. Replication Statements	1406
13.4.1. SQL Statements for Controlling Master Servers	1406
13.4.2. SQL Statements for Controlling Slave Servers	1408
13.5. SQL Syntax for Prepared Statements	1415
13.5.1. <code>PREPARE</code> Syntax	1418
13.5.2. <code>EXECUTE</code> Syntax	1418
13.5.3. <code>DEALLOCATE PREPARE</code> Syntax	1419
13.5.4. Automatic Prepared Statement Repreparation	1419
13.6. MySQL Compound-Statement Syntax	1419
13.6.1. <code>BEGIN ... END</code> Compound-Statement Syntax	1419
13.6.2. Statement Label Syntax	1420
13.6.3. <code>DECLARE</code> Syntax	1421
13.6.4. Variables in Stored Programs	1421
13.6.5. Flow Control Statements	1423
13.6.6. Cursors	1427
13.6.7. Condition Handling	1428
13.7. Database Administration Statements	1443
13.7.1. Account Management Statements	1443
13.7.2. Table Maintenance Statements	1457
13.7.3. Plugin and User-Defined Function Statements	1465
13.7.4. <code>SET</code> Syntax	1468
13.7.5. <code>SHOW</code> Syntax	1471
13.7.6. Other Administrative Statements	1511
13.8. MySQL Utility Statements	1519
13.8.1. <code>DESCRIBE</code> Syntax	1519
13.8.2. <code>EXPLAIN</code> Syntax	1519
13.8.3. <code>HELP</code> Syntax	1520
13.8.4. <code>USE</code> Syntax	1522

This chapter describes the syntax for the SQL statements supported by MySQL.

13.1. Data Definition Statements

13.1.1. `ALTER DATABASE` Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification ...
ALTER {DATABASE | SCHEMA} db_name
    UPGRADE DATA DIRECTORY NAME

alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
    | [DEFAULT] COLLATE [=] collation_name
```

`ALTER DATABASE` enables you to change the overall characteristics of a database. These characteristics are stored in the `db.opt` file in the database directory. To use `ALTER DATABASE`, you need the `ALTER [695]` privilege on the database. `ALTER SCHEMA` is a synonym for `ALTER DATABASE`.

The database name can be omitted from the first syntax, in which case the statement applies to the default database.

National Language Characteristics

The `CHARACTER SET` clause changes the default database character set. The `COLLATE` clause changes the default database collation. [Section 10.1, “Character Set Support”](#), discusses character set and collation names.

You can see what character sets and collations are available using, respectively, the `SHOW CHARACTER SET` and `SHOW COLLATION` statements. See [Section 13.7.5.4, “SHOW CHARACTER SET Syntax”](#), and [Section 13.7.5.5, “SHOW COLLATION Syntax”](#), for more information.

If you change the default character set or collation for a database, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See [Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).)

Upgrading from Versions Older than MySQL 5.1

The syntax that includes the `UPGRADE DATA DIRECTORY NAME` clause updates the name of the directory associated with the database to use the encoding implemented in MySQL 5.1 for mapping database names to database directory names (see [Section 9.2.3, “Mapping of Identifiers to File Names”](#)). This clause is for use under these conditions:

- It is intended when upgrading MySQL to 5.1 or later from older versions.
- It is intended to update a database directory name to the current encoding format if the name contains special characters that need encoding.
- The statement is used by `mysqlcheck` (as invoked by `mysql_upgrade`).

For example, if a database in MySQL 5.0 has the name `a-b-c`, the name contains instances of the `-` (dash) character. In MySQL 5.0, the database directory is also named `a-b-c`, which is not necessarily safe for all file systems. In MySQL 5.1 and later, the same database name is encoded as `a@002db@002dc` to produce a file system-neutral directory name.

When a MySQL installation is upgraded to MySQL 5.1 or later from an older version, the server displays a name such as `a-b-c` (which is in the old format) as `#mysql50#a-b-c`, and you must refer to the name using the `#mysql50#` prefix. Use `UPGRADE DATA DIRECTORY NAME` in this case to explicitly tell the server to re-encode the database directory name to the current encoding format:

```
ALTER DATABASE `#mysql50#a-b-c` UPGRADE DATA DIRECTORY NAME;
```

After executing this statement, you can refer to the database as `a-b-c` without the special `#mysql50#` prefix.

13.1.2. ALTER EVENT Syntax

```
ALTER
  [DEFINER = { user | CURRENT_USER }]
  EVENT event_name
  [ON SCHEDULE schedule]
  [ON COMPLETION [NOT] PRESERVE]
  [RENAME TO new_event_name]
  [ENABLE | DISABLE | DISABLE ON SLAVE]
  [COMMENT 'comment']
  [DO event_body]
```

The `ALTER EVENT` statement changes one or more of the characteristics of an existing event without the need to drop and recreate it. The syntax for each of the `DEFINER`, `ON SCHEDULE`, `ON COMPLETION`,

COMMENT, ENABLE / DISABLE, and DO clauses is exactly the same as when used with CREATE EVENT. (See Section 13.1.11, “CREATE EVENT Syntax”.)

Any user can alter an event defined on a database for which that user has the EVENT [696] privilege. When a user executes a successful ALTER EVENT statement, that user becomes the definer for the affected event.

ALTER EVENT works only with an existing event:

```
mysql> ALTER EVENT no_such_event
>     ON SCHEDULE
>     EVERY '2:3' DAY_HOUR;
ERROR 1517 (HY000): Unknown event 'no_such_event'
```

In each of the following examples, assume that the event named `myevent` is defined as shown here:

```
CREATE EVENT myevent
ON SCHEDULE
  EVERY 6 HOUR
COMMENT 'A sample comment.'
DO
  UPDATE myschema.mytable SET mycol = mycol + 1;
```

The following statement changes the schedule for `myevent` from once every six hours starting immediately to once every twelve hours, starting four hours from the time the statement is run:

```
ALTER EVENT myevent
ON SCHEDULE
  EVERY 12 HOUR
STARTS CURRENT_TIMESTAMP + INTERVAL 4 HOUR;
```

It is possible to change multiple characteristics of an event in a single statement. This example changes the SQL statement executed by `myevent` to one that deletes all records from `mytable`; it also changes the schedule for the event such that it executes once, one day after this ALTER EVENT statement is run.

```
ALTER EVENT myevent
ON SCHEDULE
  AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
DO
  TRUNCATE TABLE myschema.mytable;
```

Specify the options in an ALTER EVENT statement only for those characteristics that you want to change; omitted options keep their existing values. This includes any default values for CREATE EVENT such as ENABLE.

To disable `myevent`, use this ALTER EVENT statement:

```
ALTER EVENT myevent
DISABLE;
```

The ON SCHEDULE clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *timestamp* or *interval* values which it contains. You cannot use stored routines or user-defined functions in such expressions, and you cannot use any table references; however, you can use SELECT FROM DUAL. This is true for both ALTER EVENT and CREATE EVENT statements. References to stored routines, user-defined functions, and tables in such cases are specifically not permitted, and fail with an error (see Bug #22830).

Although an ALTER EVENT statement that contains another ALTER EVENT statement in its DO clause appears to succeed, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

To rename an event, use the `ALTER EVENT` statement's `RENAME TO` clause. This statement renames the event `myevent` to `yourevent`:

```
ALTER EVENT myevent
  RENAME TO yourevent;
```

You can also move an event to a different database using `ALTER EVENT ... RENAME TO ...` and `db_name.event_name` notation, as shown here:

```
ALTER EVENT olddb.myevent
  RENAME TO newdb.myevent;
```

To execute the previous statement, the user executing it must have the `EVENT [696]` privilege on both the `olddb` and `newdb` databases.



Note

There is no `RENAME EVENT` statement.

The value `DISABLE ON SLAVE` is used on a replication slave instead of `ENABLED` or `DISABLED` to indicate an event that was created on the master and replicated to the slave, but that is not executed on the slave. Normally, `DISABLE ON SLAVE` is set automatically as required; however, there are some circumstances under which you may want or need to change it manually. See [Section 16.4.1.10, “Replication of Invoked Features”](#), for more information.

13.1.3. ALTER LOGFILE GROUP Syntax

```
ALTER LOGFILE GROUP logfile_group
  ADD UNDOFILE 'file_name'
  [INITIAL_SIZE [=] size]
  [WAIT]
  ENGINE [=] engine_name
```

This statement adds an `UNDO` file named `'file_name'` to an existing log file group `logfile_group`. An `ALTER LOGFILE GROUP` statement has one and only one `ADD UNDOFILE` clause. No `DROP UNDOFILE` clause is currently supported.



Note

All MySQL Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and an undo log file with the same name, or an undo log file and a data file with the same name.

The optional `INITIAL_SIZE` parameter sets the `UNDO` file's initial size in bytes; if not specified, the initial size default to `128M` (128 megabytes). You may optionally follow `size` with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (for megabytes) or `G` (for gigabytes).

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is `4G`. (Bug #29186)

The minimum permitted value for `INITIAL_SIZE` is `1M`. (Bug #29574)



Note

`WAIT` is parsed but otherwise ignored. This keyword currently has no effect, and is intended for future expansion.

The `ENGINE` parameter (required) determines the storage engine which is used by this log file group, with `engine_name` being the name of the storage engine. Currently, the only accepted values for `engine_name` are “NDBCLUSTER” and “NDB”. The two values are equivalent.

Here is an example, which assumes that the log file group `lg_3` has already been created using `CREATE LOGFILE GROUP` (see [Section 13.1.14](#), “`CREATE LOGFILE GROUP Syntax`”):

```
ALTER LOGFILE GROUP lg_3
  ADD UNDOFILE 'undo_10.dat'
  INITIAL_SIZE=32M
  ENGINE=NDBCLUSTER;
```

When `ALTER LOGFILE GROUP` is used with `ENGINE = NDBCLUSTER` (alternatively, `ENGINE = NDB`), an `UNDO` log file is created on each MySQL Cluster data node. You can verify that the `UNDO` files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example:

```
mysql> SELECT FILE_NAME, LOGFILE_GROUP_NUMBER, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg_3';
+-----+-----+-----+
| FILE_NAME | LOGFILE_GROUP_NUMBER | EXTRA |
+-----+-----+-----+
| newdata.dat | 0 | CLUSTER_NODE=3 |
| newdata.dat | 0 | CLUSTER_NODE=4 |
| undo_10.dat | 11 | CLUSTER_NODE=3 |
| undo_10.dat | 11 | CLUSTER_NODE=4 |
+-----+-----+-----+
4 rows in set (0.01 sec)
```

(See [Section 20.29.1](#), “[The INFORMATION_SCHEMA FILES Table](#)”.)

Memory used for `UNDO_BUFFER_SIZE` comes from the global pool whose size is determined by the value of the `SharedGlobalMemory` [2083] data node configuration parameter. This includes any default value implied for this option by the setting of the `InitialLogFileGroup` [2085] data node configuration parameter.

`ALTER LOGFILE GROUP` is useful only with Disk Data storage for MySQL Cluster. For more information, see [Section 17.5.12](#), “[MySQL Cluster Disk Data Tables](#)”.

13.1.4. ALTER FUNCTION Syntax

```
ALTER FUNCTION func_name [characteristic ...]

characteristic:
  COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

This statement can be used to change the characteristics of a stored function. More than one change may be specified in an `ALTER FUNCTION` statement. However, you cannot change the parameters or body of a stored function using this statement; to make such changes, you must drop and re-create the function using `DROP FUNCTION` and `CREATE FUNCTION`.

You must have the `ALTER ROUTINE` [695] privilege for the function. (That privilege is granted automatically to the function creator.) If binary logging is enabled, the `ALTER FUNCTION` statement might also require the `SUPER` [697] privilege, as described in [Section 19.7](#), “[Binary Logging of Stored Programs](#)”.

13.1.5. ALTER PROCEDURE Syntax

```
ALTER PROCEDURE proc_name [characteristic ...]
```

```

characteristic:
  COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

```

This statement can be used to change the characteristics of a stored procedure. More than one change may be specified in an `ALTER PROCEDURE` statement. However, you cannot change the parameters or body of a stored procedure using this statement; to make such changes, you must drop and re-create the procedure using `DROP PROCEDURE` and `CREATE PROCEDURE`.

You must have the `ALTER ROUTINE` [695] privilege for the procedure. By default, that privilege is granted automatically to the procedure creator. This behavior can be changed by disabling the `automatic_sp_privileges` [484] system variable. See Section 19.2.2, “Stored Routines and MySQL Privileges”.

13.1.6. ALTER SERVER Syntax

```

ALTER SERVER server_name
  OPTIONS (option [, option] ...)

```

Alters the server information for *server_name*, adjusting any of the options permitted in the `CREATE SERVER` statement. The corresponding fields in the `mysql.servers` table are updated accordingly. This statement requires the `SUPER` [697] privilege.

For example, to update the `USER` option:

```

ALTER SERVER s OPTIONS (USER 'sally');

```

`ALTER SERVER` does not cause an automatic commit.

In MySQL 5.5, `ALTER SERVER` is not written to the binary log, regardless of the logging format that is in use.

13.1.7. ALTER TABLE Syntax

```

ALTER [ONLINE | OFFLINE] [IGNORE] TABLE tbl_name
  [alter_specification [, alter_specification] ...]
  [partition_options]

alter_specification:
  table_options
  | ADD [COLUMN] col_name column_definition
    [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX|KEY} [index_name]
    [index_type] (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
    [index_type] (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]]
    UNIQUE [INDEX|KEY] [index_name]
    [index_type] (index_col_name,...) [index_option] ...
  | ADD FULLTEXT [INDEX|KEY] [index_name]
    (index_col_name,...) [index_option] ...
  | ADD SPATIAL [INDEX|KEY] [index_name]
    (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]]
    FOREIGN KEY [index_name] (index_col_name,...)
    reference_definition
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
    [FIRST|AFTER col_name]
  | MODIFY [COLUMN] col_name column_definition

```

```

    [FIRST | AFTER col_name]
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP {INDEX|KEY} index_name
| DROP FOREIGN KEY fk_symbol
| MAX_ROWS = rows
| DISABLE KEYS
| ENABLE KEYS
| RENAME [TO|AS] new_tbl_name
| ORDER BY col_name [, col_name] ...
| CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
| [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
| DISCARD TABLESPACE
| IMPORT TABLESPACE
| FORCE
| ADD PARTITION (partition_definition)
| DROP PARTITION partition_names
| TRUNCATE PARTITION {partition_names | ALL}
| COALESCE PARTITION number
| REORGANIZE PARTITION [partition_names INTO (partition_definitions)]
| ANALYZE PARTITION {partition_names | ALL}
| CHECK PARTITION {partition_names | ALL}
| OPTIMIZE PARTITION {partition_names | ALL}
| REBUILD PARTITION {partition_names | ALL}
| REPAIR PARTITION {partition_names | ALL}
| PARTITION BY partitioning_expression
| REMOVE PARTITIONING

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
| index_type
| WITH PARSER parser_name
| COMMENT 'string'

:
    table_option [[,] table_option] ... (see CREATE TABLE options)

:
    (see CREATE TABLE options)

```

ALTER TABLE changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change characteristics such as the storage engine used for the table or the table comment.

Partitioning-related clauses for **ALTER TABLE** can be used with partitioned tables for repartitioning, for adding, dropping, merging, and splitting partitions, and for performing partitioning maintenance. For more information, see [Section 13.1.7.1, “ALTER TABLE Partition Operations”](#). If an **ALTER TABLE** statement contains a partitioning operation, that must be the only operation specified in the statement.

Following the table name, specify the alterations to be made. If none are given, **ALTER TABLE** does nothing.

The syntax for many of the permissible alterations is similar to clauses of the **CREATE TABLE** statement. See [Section 13.1.17, “CREATE TABLE Syntax”](#), for more information.

Some operations may result in warnings if attempted on a table for which the storage engine does not support the operation. These warnings can be displayed with **SHOW WARNINGS**. See [Section 13.7.5.41, “SHOW WARNINGS Syntax”](#).

For information on troubleshooting `ALTER TABLE`, see [Section C.5.7.1, “Problems with ALTER TABLE”](#).

Storage, Performance, and Concurrency Considerations

In most cases, `ALTER TABLE` makes a temporary copy of the original table. MySQL waits for other operations that are modifying the table, then proceeds. It incorporates the alteration into the copy, deletes the original table, and renames the new one. While `ALTER TABLE` is executing, the original table is readable by other sessions. Updates and writes to the table that begin after the `ALTER TABLE` operation begins are stalled until the new table is ready, then are automatically redirected to the new table without any failed updates. The temporary table is created in the database directory of the new table. This can differ from the database directory of the original table for `ALTER TABLE` operations that rename the table to a different database.

For `MyISAM` tables, you can speed up index re-creation (the slowest part of the alteration process) by setting the `myisam_sort_buffer_size` [530] system variable to a high value.

For some operations, an in-place `ALTER TABLE` is possible that does not require a temporary table:

- For `ALTER TABLE tbl_name RENAME TO new_tbl_name` without any other options, MySQL simply renames any files that correspond to the table `tbl_name` without making a copy. (You can also use the `RENAME TABLE` statement to rename tables. See [Section 13.1.32, “RENAME TABLE Syntax”](#).) Any privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.
- Alterations that modify only table metadata and not table data can be made immediately by altering the table's `.frm` file and not touching table contents. The following changes are fast alterations that can be made this way:
 - Renaming a column, except for the `InnoDB` storage engine.
 - Changing the default value of a column (except for `NDB` tables; see [Limitations of NDBCLUSTER online operations](#)).
 - Changing the definition of an `ENUM` or `SET` column by adding new enumeration or set members to the *end* of the list of valid member values, as long as the storage side of the data type does not change. For example, adding a member to a `SET` column that has 8 members changes the required storage per value from 1 byte to 2 bytes; this will require a table copy. Adding members in the middle of the list causes renumbering of existing members, which requires a table copy.
- `ALTER TABLE` with `ADD PARTITION`, `DROP PARTITION`, `COALESCE PARTITION`, `REBUILD PARTITION`, or `REORGANIZE PARTITION` does not create any temporary tables (except when used with `NDB` tables); however, these operations can and do create temporary partition files.

`ADD` or `DROP` operations for `RANGE` or `LIST` partitions are immediate operations or nearly so. `ADD` or `COALESCE` operations for `HASH` or `KEY` partitions copy data between all partitions, unless `LINEAR HASH` or `LINEAR KEY` was used; this is effectively the same as creating a new table, although the `ADD` or `COALESCE` operation is performed partition by partition. `REORGANIZE` operations copy only changed partitions and do not touch unchanged ones.

- Renaming an index, except for `InnoDB`.
- Adding or dropping an index, for `InnoDB` and `NDB`.

You can force an `ALTER TABLE` operation that would otherwise not require a table copy to use the temporary table method (as supported in MySQL 5.0) by setting the `old_alter_table` [534] system variable to `ON`.

As of MySQL 5.5.11, you can also use `ALTER TABLE tbl_name FORCE` to perform a “null” alter operation that rebuilds the table. Previously the `FORCE` option was recognized but ignored.

For `NDBCLUSTER` tables, operations that add and drop indexes on variable-width columns occur online, without any table copying and without blocking concurrent DML actions for most of their duration. See [Section 13.1.7.2, “ALTER TABLE Online Operations”](#).

Usage Notes

- To use `ALTER TABLE`, you need `ALTER [695]`, `CREATE [695]`, and `INSERT [696]` privileges for the table. Renaming a table requires `ALTER [695]` and `DROP [696]` on the old table, `ALTER [695]`, `CREATE [695]`, and `INSERT [696]` on the new table.
- `IGNORE` is a MySQL extension to standard SQL. It controls how `ALTER TABLE` works if there are duplicates on unique keys in the new table or if warnings occur when strict mode is enabled. If `IGNORE` is not specified, the copy is aborted and rolled back if duplicate-key errors occur. If `IGNORE` is specified, only the first row is used of rows with duplicates on a unique key. The other conflicting rows are deleted. Incorrect values are truncated to the closest matching acceptable value.
- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.
- `table_option` signifies a table option of the kind that can be used in the `CREATE TABLE` statement, such as `ENGINE`, `AUTO_INCREMENT`, or `AVG_ROW_LENGTH`. ([Section 13.1.17, “CREATE TABLE Syntax”](#), lists all table options.) However, `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.

For example, to convert a table to be an `InnoDB` table, use this statement:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

See [Section 14.3.5.2, “Converting Tables from MyISAM to InnoDB”](#) for considerations when switching tables to the `InnoDB` storage engine.

When you specify an `ENGINE` clause, `ALTER TABLE` rebuilds the table. This is true even if the table already has the specified storage engine.

The outcome of attempting to change a table's storage engine is affected by whether the desired storage engine is available and the setting of the `NO_ENGINE_SUBSTITUTION [614]` SQL mode, as described in [Section 5.1.7, “Server SQL Modes”](#).

To prevent inadvertent loss of data, `ALTER TABLE` cannot be used to change the storage engine of a table to `MERGE` or `BLACKHOLE`.

To change the value of the `AUTO_INCREMENT` counter to be used for new rows, do this:

```
ALTER TABLE t2 AUTO_INCREMENT = value;
```

You cannot reset the counter to a value less than or equal to any that have already been used. For `MyISAM`, if the value is less than or equal to the maximum value currently in the `AUTO_INCREMENT` column, the value is reset to the current maximum plus one. For `InnoDB`, *if the value is less than the current maximum value in the column, no error occurs and the current sequence value is not changed.*

- You can issue multiple `ADD`, `ALTER`, `DROP`, and `CHANGE` clauses in a single `ALTER TABLE` statement, separated by commas. This is a MySQL extension to standard SQL, which permits only one of each clause per `ALTER TABLE` statement. For example, to drop multiple columns in a single statement, do this:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- `CHANGE col_name`, `DROP col_name`, and `DROP INDEX` are MySQL extensions to standard SQL.
- The word `COLUMN` is optional and can be omitted.
- `column_definition` clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. See [Section 13.1.17, “CREATE TABLE Syntax”](#).
- You can rename a column using a `CHANGE old_col_name new_col_name column_definition` clause. To do so, specify the old and new column names and the definition that the column currently has. For example, to rename an `INTEGER` column from `a` to `b`, you can do this:

```
ALTER TABLE t1 CHANGE a b INTEGER;
```

To change a column's type but not the name, `CHANGE` syntax still requires an old and new column name, even if they are the same. For example:

```
ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

You can also use `MODIFY` to change a column's type without renaming it:

```
ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

`MODIFY` is an extension to `ALTER TABLE` for Oracle compatibility.

When you use `CHANGE` or `MODIFY`, `column_definition` must include the data type and all attributes that should apply to the new column, other than index attributes such as `PRIMARY KEY` or `UNIQUE`. Attributes present in the original definition but not specified for the new definition are not carried forward. Suppose that a column `col1` is defined as `INT UNSIGNED DEFAULT 1 COMMENT 'my column'` and you modify the column as follows:

```
ALTER TABLE t1 MODIFY col1 BIGINT;
```

The resulting column will be defined as `BIGINT`, but will not include the attributes `UNSIGNED DEFAULT 1 COMMENT 'my column'`. To retain them, the statement should be:

```
ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my column';
```

- When you change a data type using `CHANGE` or `MODIFY`, MySQL tries to convert existing column values to the new type as well as possible.



Warning

This conversion may result in alteration of data. For example, if you shorten a string column, values may be truncated. To prevent the operation from succeeding if conversions to the new data type would result in loss of data, enable strict SQL mode before using `ALTER TABLE` (see [Section 5.1.7, “Server SQL Modes”](#)).

- To add a column at a specific position within a table row, use `FIRST` or `AFTER col_name`. The default is to add the column last. You can also use `FIRST` and `AFTER` in `CHANGE` or `MODIFY` operations to reorder columns within a table.
- `ALTER ... SET DEFAULT` or `ALTER ... DROP DEFAULT` specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be `NULL`, the new default is `NULL`. If the column cannot be `NULL`, MySQL assigns a default value as described in [Section 11.5, “Data Type Default Values”](#).

- `DROP INDEX` removes an index. This is a MySQL extension to standard SQL. See [Section 13.1.24, “DROP INDEX Syntax”](#). If you are unsure of the index name, use `SHOW INDEX FROM tbl_name`.
- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well. If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.
- If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use `DROP TABLE` instead.
- `DROP PRIMARY KEY` drops the [primary key](#). If there is no primary key, an error occurs. For information about the performance characteristics of primary keys, especially for [InnoDB](#) tables, see [Section 8.3.2, “Using Primary Keys”](#).

If you add a `UNIQUE INDEX` or `PRIMARY KEY` to a table, MySQL stores it before any nonunique index to permit detection of duplicate keys as early as possible.

- Some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is `USING type_name`. For details about `USING`, see [Section 13.1.13, “CREATE INDEX Syntax”](#). The preferred position is after the column list. Support for use of the option before the column list will be removed in a future MySQL release.

index_option values specify additional options for an index. `USING` is one such option. For details about permissible *index_option* values, see [Section 13.1.13, “CREATE INDEX Syntax”](#).

- After an `ALTER TABLE` statement, it may be necessary to run `ANALYZE TABLE` to update index cardinality information. See [Section 13.7.5.23, “SHOW INDEX Syntax”](#).
- `ORDER BY` enables you to create the new table with the rows in a specific order. Note that the table does not remain in this order after inserts and deletes. This option is useful primarily when you know that you are mostly to query the rows in a certain order most of the time. By using this option after major changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.

`ORDER BY` syntax permits one or more column names to be specified for sorting, each of which optionally can be followed by `ASC` or `DESC` to indicate ascending or descending sort order, respectively. The default is ascending order. Only column names are permitted as sort criteria; arbitrary expressions are not permitted. This clause should be given last after any other clauses.

`ORDER BY` does not make sense for [InnoDB](#) tables that contain a user-defined clustered index (`PRIMARY KEY` or `NOT NULL UNIQUE` index). [InnoDB](#) always orders table rows according to such an index if one is present.



Note

When used on a partitioned table, `ALTER TABLE ... ORDER BY` orders rows within each partition only.

- If you use `ALTER TABLE` on a [MyISAM](#) table, all nonunique indexes are created in a separate batch (as for `REPAIR TABLE`). This should make `ALTER TABLE` much faster when you have many indexes.

This feature can be activated explicitly for a [MyISAM](#) table. `ALTER TABLE ... DISABLE KEYS` tells MySQL to stop updating nonunique indexes. `ALTER TABLE ... ENABLE KEYS` then should be used to re-create missing indexes. MySQL does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable

speedup. Using `ALTER TABLE ... DISABLE KEYS` requires the `INDEX [696]` privilege in addition to the privileges mentioned earlier.

While the nonunique indexes are disabled, they are ignored for statements such as `SELECT` and `EXPLAIN` that otherwise would use them.

- If `ALTER TABLE` for an `InnoDB` table results in changes to column values (for example, because a column is truncated), `InnoDB`'s `FOREIGN KEY` constraint checks do not notice possible violations caused by changing the values.
- The `FOREIGN KEY` and `REFERENCES` clauses are supported by the `InnoDB` storage engine, which implements `ADD [CONSTRAINT [symbol]] FOREIGN KEY (...) REFERENCES ... (...)`. See [Section 14.3.5.4, “FOREIGN KEY Constraints”](#). For other storage engines, the clauses are parsed but ignored. The `CHECK` clause is parsed but ignored by all storage engines. See [Section 13.1.17, “CREATE TABLE Syntax”](#). The reason for accepting but ignoring syntax clauses is for compatibility, to make it easier to port code from other SQL servers, and to run applications that create tables with references. See [Section 1.8.5, “MySQL Differences from Standard SQL”](#).



Important

The inline `REFERENCES` specifications where the references are defined as part of the column specification are silently ignored by `InnoDB`. `InnoDB` only accepts `REFERENCES` clauses defined as part of a separate `FOREIGN KEY` specification.



Note

Partitioned tables do not support foreign keys. See [Section 18.5, “Restrictions and Limitations on Partitioning”](#), for more information.

- `InnoDB` supports the use of `ALTER TABLE` to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

For more information, see [Section 14.3.5.4, “FOREIGN KEY Constraints”](#).

- You cannot add a foreign key and drop a foreign key in separate clauses of a single `ALTER TABLE` statement. You must use separate statements.
- For an `InnoDB` table that is created with its own tablespace in an `.ibd` file, that file can be discarded and imported. To discard the `.ibd` file, use this statement:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

This deletes the current `.ibd` file, so be sure that you have a backup first. Attempting to access the table while the tablespace file is discarded results in an error.

To import the backup `.ibd` file back into the table, copy it into the database directory, and then issue this statement:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

The tablespace file must have been created on the server into which it is imported later.

See [Section 14.3.3, “Using Per-Table Tablespaces”](#).

- To change the table default character set and all character columns (`CHAR`, `VARCHAR`, `TEXT`) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

For a column that has a data type of `VARCHAR` or one of the `TEXT` types, `CONVERT TO CHARACTER SET` will change the data type as necessary to ensure that the new column is long enough to store as many characters as the original column. For example, a `TEXT` column has two length bytes, which store the byte-length of values in the column, up to a maximum of 65,535. For a `latin1 TEXT` column, each character requires a single byte, so the column can store up to 65,535 characters. If the column is converted to `utf8`, each character might require up to three bytes, for a maximum possible length of $3 \times 65,535 = 196,605$ bytes. That length will not fit in a `TEXT` column's length bytes, so MySQL will convert the data type to `MEDIUMTEXT`, which is the smallest string type for which the length bytes can record a value of 196,605. Similarly, a `VARCHAR` column might be converted to `MEDIUMTEXT`.

To avoid data type changes of the type just described, do not use `CONVERT TO CHARACTER SET`. Instead, use `MODIFY` to change individual columns. For example:

```
ALTER TABLE t MODIFY latin1_text_col TEXT CHARACTER SET utf8;
ALTER TABLE t MODIFY latin1_varchar_col VARCHAR(M) CHARACTER SET utf8;
```

If you specify `CONVERT TO CHARACTER SET binary`, the `CHAR`, `VARCHAR`, and `TEXT` columns are converted to their corresponding binary string types (`BINARY`, `VARBINARY`, `BLOB`). This means that the columns no longer will have a character set and a subsequent `CONVERT TO` operation will not apply to them.

If `charset_name` is `DEFAULT`, the database character set is used.



Warning

The `CONVERT TO` operation converts column values between the character sets. This is *not* what you want if you have a column in one character set (like `latin1`) but the stored values actually use some other, incompatible character set (like `utf8`). In this case, you have to do the following for each such column:

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

The reason this works is that there is no conversion when you convert to or from `BLOB` columns.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word `DEFAULT` is optional. The default character set is the character set that is used if you do not specify the character set for columns that you add to a table later (for example, with `ALTER TABLE ... ADD column`).

With the `mysql_info()` C API function, you can find out how many rows were copied by `ALTER TABLE`, and (when `IGNORE` is used) how many rows were deleted due to duplication of unique key values. See [Section 22.8.3.35, "mysql_info\(\)"](#).

13.1.7.1. ALTER TABLE Partition Operations

Partitioning-related clauses for `ALTER TABLE` can be used with partitioned tables for repartitioning, for adding, dropping, merging, and splitting partitions, and for performing partitioning maintenance. If an `ALTER TABLE` statement contains a partitioning operation, that must be the only operation specified in the statement.

- Simply using a *partition_options* clause with `ALTER TABLE` on a partitioned table repartitions the table according to the partitioning scheme defined by the *partition_options*. This clause always begins with `PARTITION BY`, and follows the same syntax and other rules as apply to the *partition_options* clause for `CREATE TABLE` (see [Section 13.1.17, “CREATE TABLE Syntax”](#), for more detailed information), and can also be used to partition an existing table that is not already partitioned. For example, consider a (nonpartitioned) table defined as shown here:

```
CREATE TABLE t1 (
  id INT,
  year_col INT
);
```

This table can be partitioned by `HASH`, using the `id` column as the partitioning key, into 8 partitions by means of this statement:

```
ALTER TABLE t1
  PARTITION BY HASH(id)
  PARTITIONS 8;
```

The table that results from using an `ALTER TABLE ... PARTITION BY` statement must follow the same rules as one created using `CREATE TABLE ... PARTITION BY`. This includes the rules governing the relationship between any unique keys (including any primary key) that the table might have, and the column or columns used in the partitioning expression, as discussed in [Section 18.5.1, “Partitioning Keys, Primary Keys, and Unique Keys”](#). The `CREATE TABLE ... PARTITION BY` rules for specifying the number of partitions also apply to `ALTER TABLE ... PARTITION BY`.

The *partition_definition* clause for `ALTER TABLE ADD PARTITION` supports the same options as the clause of the same name for the `CREATE TABLE` statement. (See [Section 13.1.17, “CREATE TABLE Syntax”](#), for the syntax and description.) Suppose that you have the partitioned table created as shown here:

```
CREATE TABLE t1 (
  id INT,
  year_col INT
)
PARTITION BY RANGE (year_col) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (1999)
);
```

You can add a new partition `p3` to this table for storing values less than `2002` as follows:

```
ALTER TABLE t1 ADD PARTITION (PARTITION p3 VALUES LESS THAN (2002));
```

`DROP PARTITION` can be used to drop one or more `RANGE` or `LIST` partitions. This statement cannot be used with `HASH` or `KEY` partitions; instead, use `COALESCE PARTITION` (see below). Any data that was stored in the dropped partitions named in the *partition_names* list is discarded. For example, given the table `t1` defined previously, you can drop the partitions named `p0` and `p1` as shown here:

```
ALTER TABLE t1 DROP PARTITION p0, p1;
```



Note

`DROP PARTITION` does not work with tables that use the `NDBCLUSTER` storage engine. See [Section 18.3.1, “Management of RANGE and LIST Partitions”](#), and [Section 17.1.6, “Known Limitations of MySQL Cluster”](#).

`ADD PARTITION` and `DROP PARTITION` do not currently support `IF [NOT] EXISTS`.

Renames of partitioned table are supported. You can rename individual partitions indirectly using `ALTER TABLE ... REORGANIZE PARTITION`; however, this operation makes a copy of the partition's data..

Beginning with MySQL 5.5.0, it is possible to delete rows from selected partitions using the `TRUNCATE PARTITION` option. This option takes a comma-separated list of one or more partition names. For example, consider the table `t1` as defined here:

```
CREATE TABLE t1 (
  id INT,
  year_col INT
)
PARTITION BY RANGE (year_col) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (1999),
  PARTITION p3 VALUES LESS THAN (2003),
  PARTITION p4 VALUES LESS THAN (2007)
);
```

To delete all rows from partition `p0`, you can use the following statement:

```
ALTER TABLE t1 TRUNCATE PARTITION p0;
```

The statement just shown has the same effect as the following `DELETE` statement:

```
DELETE FROM t1 WHERE year_col < 1991;
```

When truncating multiple partitions, the partitions do not have to be contiguous: This can greatly simplify delete operations on partitioned tables that would otherwise require very complex `WHERE` conditions if done with `DELETE` statements. For example, this statement deletes all rows from partitions `p1` and `p3`:

```
ALTER TABLE t1 TRUNCATE PARTITION p1, p3;
```

An equivalent `DELETE` statement is shown here:

```
DELETE FROM t1 WHERE
  (year_col >= 1991 AND year_col < 1995)
  OR
  (year_col >= 2003 AND year_col < 2007);
```

You can also use the `ALL` keyword in place of the list of partition names; in this case, the statement acts on all partitions in the table.

`TRUNCATE PARTITION` merely deletes rows; it does not alter the definition of the table itself, or of any of its partitions.



Note

`TRUNCATE PARTITION` does not work with subpartitions.

You can verify that the rows were dropped by checking the `INFORMATION_SCHEMA.PARTITIONS` table, using a query such as this one:

```
SELECT PARTITION_NAME, TABLE_ROWS
  FROM INFORMATION_SCHEMA.PARTITIONS
 WHERE TABLE_NAME = 't1';
```

`TRUNCATE PARTITION` is supported only for partitioned tables that use the `MyISAM`, `InnoDB`, or `MEMORY` storage engine. It also works on `BLACKHOLE` tables (but has no effect). It is not supported for `ARCHIVE` tables.

`COALESCE PARTITION` can be used with a table that is partitioned by `HASH` or `KEY` to reduce the number of partitions by *number*. Suppose that you have created table `t2` using the following definition:

```
CREATE TABLE t2 (
  name VARCHAR (30),
  started DATE
)
PARTITION BY HASH( YEAR(started) )
PARTITIONS 6;
```

You can reduce the number of partitions used by `t2` from 6 to 4 using the following statement:

```
ALTER TABLE t2 COALESCE PARTITION 2;
```

The data contained in the last *number* partitions will be merged into the remaining partitions. In this case, partitions 4 and 5 will be merged into the first 4 partitions (the partitions numbered 0, 1, 2, and 3).

To change some but not all the partitions used by a partitioned table, you can use `REORGANIZE PARTITION`. This statement can be used in several ways:

- To merge a set of partitions into a single partition. This can be done by naming several partitions in the *partition_names* list and supplying a single definition for *partition_definition*.
- To split an existing partition into several partitions. You can accomplish this by naming a single partition for *partition_names* and providing multiple *partition_definitions*.
- To change the ranges for a subset of partitions defined using `VALUES LESS THAN` or the value lists for a subset of partitions defined using `VALUES IN`.
- This statement may also be used without the *partition_names INTO (partition_definitions)* option on tables that are automatically partitioned using `HASH` partitioning to force redistribution of data. (Currently, only `NDBCLUSTER` tables are automatically partitioned in this way.) This is useful in MySQL Cluster where, after you have added new MySQL Cluster data nodes online to an existing MySQL Cluster, you wish to redistribute existing MySQL Cluster table data to the new data nodes. In such cases, you should invoke the statement with the `ONLINE` option; in other words, as shown here:

```
ALTER ONLINE TABLE table REORGANIZE PARTITION;
```

You cannot perform other DDL concurrently with online table reorganization—that is, no other DDL statements can be issued while an `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement is executing. For more information about adding MySQL Cluster data nodes online, see [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#).

`ALTER ONLINE TABLE ... REORGANIZE PARTITION` does not work with tables which were created using the `MAX_ROWS` option, because it uses the constant `MAX_ROWS` value specified in the original `CREATE TABLE` statement to determine the number of partitions required, so no new partitions are created. Beginning with MySQL Cluster NDB 7.2.6, you can use `ALTER ONLINE TABLE ... MAX_ROWS=rows` to increase the maximum number of rows for the table; after this, `ALTER ONLINE TABLE ... REORGANIZE PARTITION` can use this new, larger value to increase the number of partitions. The value of *rows* must be greater than the value specified for `MAX_ROWS` in the original `CREATE TABLE` statement for this to work.

Attempting to use `REORGANIZE PARTITION` without the `partition_names INTO (partition_definitions)` option on explicitly partitioned tables results in the error `REORGANIZE PARTITION without parameters can only be used on auto-partitioned tables using HASH partitioning`.



Note

For partitions that have not been explicitly named, MySQL automatically provides the default names `p0`, `p1`, `p2`, and so on. The same is true with regard to subpartitions.

For more detailed information about and examples of `ALTER TABLE ... REORGANIZE PARTITION` statements, see [Section 18.3.1, “Management of RANGE and LIST Partitions”](#).

- Several additional options provide partition maintenance and repair functionality analogous to that implemented for nonpartitioned tables by statements such as `CHECK TABLE` and `REPAIR TABLE` (which are also supported for partitioned tables; see [Section 13.7.2, “Table Maintenance Statements”](#) for more information). These include `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, `REBUILD PARTITION`, and `REPAIR PARTITION`. Each of these options takes a `partition_names` clause consisting of one or more names of partitions, separated by commas. The partitions must already exist in the table to be altered. You can also use the `ALL` keyword in place of `partition_names`, in which case the statement acts on all partitions in the table. For more information and examples, see [Section 18.3.3, “Maintenance of Partitions”](#).

Prior to MySQL 5.5.30, it was not safe to execute multiple concurrent `REBUILD TABLE` operations on partitioned tables, whether on the same or different tables. (Bug #14589559, Bug #66645)

Some MySQL storage engines, such as `InnoDB`, do not support per-partition optimization. For a partitioned table using such a storage engine, `ALTER TABLE ... OPTIMIZE PARTITION` rebuilds the entire table. This is a known issue. (Bug #42822) Beginning with MySQL 5.5.30, running this statement on such a table causes the entire table to be rebuilt and analyzed, and an appropriate warning to be issued. (Bug #11751825)

To work around this problem, use the statements `ALTER TABLE ... REBUILD PARTITION` and `ALTER TABLE ... ANALYZE PARTITION` instead.

The `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, and `REPAIR PARTITION` options are not permitted for tables which are not partitioned.

- `REMOVE PARTITIONING` enables you to remove a table's partitioning without otherwise affecting the table or its data. This option can be combined with other `ALTER TABLE` options such as those used to add, drop, or rename drop columns or indexes.
- Using the `ENGINE` option with `ALTER TABLE` changes the storage engine used by the table without affecting the partitioning.

Only a single instance of any one of the following options can be used in a given `ALTER TABLE` statement: `PARTITION BY`, `ADD PARTITION`, `DROP PARTITION`, `TRUNCATE PARTITION`, `REORGANIZE PARTITION`, or `COALESCE PARTITION`, `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, `REBUILD PARTITION`, `REMOVE PARTITIONING`.

For example, the following two statements are invalid:

```
ALTER TABLE t1 ANALYZE PARTITION p1, ANALYZE PARTITION p2;
ALTER TABLE t1 ANALYZE PARTITION p1, CHECK PARTITION p2;
```

In the first case, you can analyze partitions `p1` and `p2` of table `t1` concurrently using a single statement with a single `ANALYZE PARTITION` option that lists both of the partitions to be analyzed, like this:

```
ALTER TABLE t1 ANALYZE PARTITION p1, p2;
```

In the second case, it is not possible to perform `ANALYZE` and `CHECK` operations on different partitions of the same table concurrently. Instead, you must issue two separate statements, like this:

```
ALTER TABLE t1 ANALYZE PARTITION p1;
ALTER TABLE t1 CHECK PARTITION p2;
```

13.1.7.2. ALTER TABLE Online Operations

Operations that add and drop indexes on variable-width columns of `NDBCLUSTER` tables occur online. Online operations are noncopying; that is, they do not require that indexes be re-created. They do not lock the table being altered from access by other API nodes in a MySQL Cluster (but see *Limitations* later in this section). Such operations do not require single user mode for `NDBCLUSTER` table alterations made in a cluster with multiple API nodes; transactions can continue uninterrupted during online DDL operations.

The `ONLINE` keyword can be used to perform online `ADD COLUMN`, `ADD INDEX` (including `CREATE INDEX` statements), and `DROP INDEX` operations on `NDBCLUSTER` tables. Online renaming of `NDBCLUSTER` tables is also supported.

The `ONLINE` and `OFFLINE` keywords are supported only in MySQL Cluster. For standard MySQL Server 5.5 releases:

- The server determines automatically whether an `ADD INDEX` or `DROP INDEX` operation can be (and is) performed online or offline; if the column is of a variable-width data type, the operation is performed online. It is not possible to override the server behavior in this regard.
- Attempting to use the `ONLINE` or `OFFLINE` keyword in an `ALTER TABLE`, `CREATE INDEX`, or `DROP INDEX` statement results in an error.

Currently you cannot add disk-based columns to `NDBCLUSTER` tables online. This means that, if you wish to add an in-memory column to an `NDBCLUSTER` table that uses a table-level `STORAGE DISK` option, you must declare the new column as using memory-based storage explicitly. For example—assuming that you have already created tablespace `ts1`—suppose that you create table `t1` as follows:

```
mysql> CREATE TABLE t1 (
>   c1 INT NOT NULL PRIMARY KEY,
>   c2 VARCHAR(30)
> )
> TABLESPACE ts1 STORAGE DISK
> ENGINE NDBCLUSTER;
Query OK, 0 rows affected (1.73 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

You can add a new in-memory column to this table online as shown here:

```
mysql> ALTER ONLINE TABLE t1 ADD COLUMN c3 INT COLUMN_FORMAT DYNAMIC STORAGE MEMORY;
Query OK, 0 rows affected (1.25 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

This statement fails if the `STORAGE MEMORY` option is omitted:

```
mysql> ALTER ONLINE TABLE t1 ADD COLUMN c3 INT COLUMN_FORMAT DYNAMIC;
ERROR 1235 (42000): This version of MySQL doesn't yet support
'ALTER ONLINE TABLE t1 ADD COLUMN c3 INT COLUMN_FORMAT DYNAMIC'
```

If you omit the `COLUMN_FORMAT DYNAMIC` option, the dynamic column format is employed automatically, but a warning is issued, as shown here:

```
mysql> ALTER ONLINE TABLE t1 ADD COLUMN c3 INT STORAGE MEMORY;
Query OK, 0 rows affected, 1 warning (1.17 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1478 | Converted FIXED field to DYNAMIC to enable on-line ADD COLUMN |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
      Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  `c2` varchar(30) DEFAULT NULL,
  `c3` int(11) /*!50120 STORAGE MEMORY */ /*!50120 COLUMN_FORMAT DYNAMIC */ DEFAULT NULL,
  `t4` int(11) /*!50120 STORAGE MEMORY */ DEFAULT NULL,
  PRIMARY KEY (`c1`)
) /*!50100 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.03 sec)
```



Note

The `STORAGE` and `COLUMN_FORMAT` keywords are supported only in MySQL Cluster; in any other version of MySQL, attempting to use either of these keywords in a `CREATE TABLE` or `ALTER TABLE` statement results in an error.

It is also possible to use the statement `ALTER ONLINE TABLE ... REORGANIZE PARTITION` with no `partition_names INTO (partition_definitions)` option on `NDBCLUSTER` tables. This can be used to redistribute MySQL Cluster data among new data nodes that have been added to the cluster online. For more information about this statement, see [Section 13.1.7.1, “ALTER TABLE Partition Operations”](#) For more information about adding data nodes online to a MySQL Cluster, see [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#).

Limitations of `NDBCLUSTER` online operations

Online `DROP COLUMN` operations are not supported.

Online `ALTER TABLE`, `CREATE INDEX`, or `DROP INDEX` statements that add columns or add or drop indexes are subject to the following limitations:

- A given online `ALTER TABLE` can use only one of `ADD COLUMN`, `ADD INDEX`, or `DROP INDEX`. One or more columns can be added online in a single statement; only one index may be created or dropped online in a single statement.
- The table being altered is not locked with respect to API nodes other than the one on which an online `ALTER TABLE ADD COLUMN`, `ADD INDEX`, or `DROP INDEX` operation (or `CREATE INDEX` or `DROP INDEX` statement) is run. However, the table is locked against any other operations originating on the *same* API node while the online operation is being executed.
- The table to be altered must have an explicit primary key; the hidden primary key created by the `NDBCLUSTER` storage engine is not sufficient for this purpose.
- The storage engine used by the table cannot be changed online.
- When used with MySQL Cluster Disk Data tables, it is not possible to change the storage type (`DISK` or `MEMORY`) of a column online. This means, that when you add or drop an index in such a way that the

**Note**

Existing columns, including the table's primary key, need not be dynamic; only the column or columns to be added online must be dynamic.

```
mysql> CREATE TABLE t2 (
>   c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY COLUMN_FORMAT FIXED
> ) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (2.10 sec)

mysql> ALTER ONLINE TABLE t2 ADD COLUMN c2 INT;
Query OK, 0 rows affected, 1 warning (0.78 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                                                 |
+-----+-----+-----+
| Warning | 1475 | Converted FIXED field to DYNAMIC to enable on-line ADD COLUMN |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Columns are not converted from `FIXED` to `DYNAMIC` column format by renaming operations. For more information about `COLUMN_FORMAT`, see [Section 13.1.17, "CREATE TABLE Syntax"](#).

The `KEY`, `CONSTRAINT`, and `IGNORE` keywords are supported in `ALTER TABLE` statements using the `ONLINE` keyword.

13.1.7.3. ALTER TABLE Examples

Begin with a table `t1` that is created as shown here:

```
CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from `t1` to `t2`:

```
ALTER TABLE t1 RENAME t2;
```

To change column `a` from `INTEGER` to `TINYINT NOT NULL` (leaving the name the same), and to change column `b` from `CHAR(10)` to `CHAR(20)` as well as renaming it from `b` to `c`:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new `TIMESTAMP` column named `d`:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

To add an index on column `d` and a `UNIQUE` index on column `a`:

```
ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);
```

To remove column `c`:

```
ALTER TABLE t2 DROP COLUMN c;
```

To add a new `AUTO_INCREMENT` integer column named `c`:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (c);
```

We indexed `c` (as a `PRIMARY KEY`) because `AUTO_INCREMENT` columns must be indexed, and we declare `c` as `NOT NULL` because primary key columns cannot be `NULL`.

For **NDB** tables, it is also possible to change the storage type used for a table or column. For example, consider an **NDB** table created as shown here:

```
mysql> CREATE TABLE t1 (c1 INT) TABLESPACE ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.27 sec)
```

To convert this table to disk-based storage, you can use the following **ALTER TABLE** statement:

```
mysql> ALTER TABLE t1 TABLESPACE ts_1 STORAGE DISK;
Query OK, 0 rows affected (2.99 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
      Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */
ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

It is not necessary that the tablespace was referenced when the table was originally created; however, the tablespace must be referenced by the **ALTER TABLE**:

```
mysql> CREATE TABLE t2 (c1 INT) ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.00 sec)

mysql> ALTER TABLE t2 STORAGE DISK;
ERROR 1005 (HY000): Can't create table 'c.#sql-1750_3' (errno: 140)
mysql> ALTER TABLE t2 TABLESPACE ts_1 STORAGE DISK;
Query OK, 0 rows affected (3.42 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> SHOW CREATE TABLE t2\G
***** 1. row *****
      Table: t1
Create Table: CREATE TABLE `t2` (
  `c1` int(11) DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */
ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

To change the storage type of an individual column, you can use **ALTER TABLE ... MODIFY [COLUMN]**. For example, suppose you create a MySQL Cluster Disk Data table with two columns, using this **CREATE TABLE** statement:

```
mysql> CREATE TABLE t3 (c1 INT, c2 INT)
->     TABLESPACE ts_1 STORAGE DISK ENGINE NDB;
Query OK, 0 rows affected (1.34 sec)
```

To change column **c2** from disk-based to in-memory storage, include a **STORAGE MEMORY** clause in the column definition used by the **ALTER TABLE** statement, as shown here:

```
mysql> ALTER TABLE t3 MODIFY c2 INT STORAGE MEMORY;
Query OK, 0 rows affected (3.14 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

You can make an in-memory column into a disk-based column by using **STORAGE DISK** in a similar fashion.

Column **c1** uses disk-based storage, since this is the default for the table (determined by the table-level **STORAGE DISK** clause in the **CREATE TABLE** statement). However, column **c2** uses in-memory storage, as can be seen here in the output of **SHOW CREATE TABLE**:

```
mysql> SHOW CREATE TABLE t3\G
```

```
***** 1. row *****
Table: t3
Create Table: CREATE TABLE `t3` (
  `c1` int(11) DEFAULT NULL,
  `c2` int(11) /*!50120 STORAGE MEMORY */ DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.02 sec)
```

When you add an `AUTO_INCREMENT` column, column values are filled in with sequence numbers automatically. For `MyISAM` tables, you can set the first sequence number by executing `SET INSERT_ID=value` before `ALTER TABLE` or by using the `AUTO_INCREMENT=value` table option. See Section 5.1.4, “Server System Variables”.

With `MyISAM` tables, if you do not change the `AUTO_INCREMENT` column, the sequence number is not affected. If you drop an `AUTO_INCREMENT` column and then add another `AUTO_INCREMENT` column, the numbers are resequenced beginning with 1.

When replication is used, adding an `AUTO_INCREMENT` column to a table might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to the table `t1`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table `t1` has columns `col1` and `col2`.

This set of statements will also produce a new table `t2` identical to `t1`, with the addition of an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```



Important

To guarantee the same ordering on both master and slave, *all* columns of `t1` must be referenced in the `ORDER BY` clause.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

13.1.8. ALTER TABLESPACE Syntax

```
ALTER TABLESPACE tablespace_name
  {ADD|DROP} DATAFILE 'file_name'
  [INITIAL_SIZE [=] size]
  [WAIT]
  ENGINE [=] engine_name
```

This statement can be used either to add a new data file, or to drop a data file from a tablespace.

The `ADD DATAFILE` variant enables you to specify an initial size using an `INITIAL_SIZE` clause, where `size` is measured in bytes; the default value is `128M` (128 megabytes). You may optionally follow this

integer value with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (for megabytes) or `G` (for gigabytes).



Note

All MySQL Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and an data file with the same name, or an undo log file and a tablespace with the same name.

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is `4G`. (Bug #29186)

`INITIAL_SIZE` is rounded, explicitly, as for `CREATE TABLESPACE`.

Once a data file has been created, its size cannot be changed; however, you can add more data files to the tablespace using additional `ALTER TABLESPACE ... ADD DATAFILE` statements.

Using `DROP DATAFILE` with `ALTER TABLESPACE` drops the data file '`file_name`' from the tablespace. You cannot drop a data file from a tablespace which is in use by any table; in other words, the data file must be empty (no extents used). See [Section 17.5.12.1, “MySQL Cluster Disk Data Objects”](#). In addition, any data file to be dropped must previously have been added to the tablespace with `CREATE TABLESPACE` or `ALTER TABLESPACE`.

Both `ALTER TABLESPACE ... ADD DATAFILE` and `ALTER TABLESPACE ... DROP DATAFILE` require an `ENGINE` clause which specifies the storage engine used by the tablespace. Currently, the only accepted values for `engine_name` are `NDB` and `NDBCLUSTER`.

`WAIT` is parsed but otherwise ignored, and so has no effect in MySQL 5.5. It is intended for future expansion.

When `ALTER TABLESPACE ... ADD DATAFILE` is used with `ENGINE = NDB`, a data file is created on each Cluster data node. You can verify that the data files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example, the following query shows all data files belonging to the tablespace named `newts`:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_NAME, EXTRA
  -> FROM INFORMATION_SCHEMA.FILES
  -> WHERE TABLESPACE_NAME = 'newts' AND FILE_TYPE = 'DATAFILE';
+-----+-----+-----+
| LOGFILE_GROUP_NAME | FILE_NAME | EXTRA |
+-----+-----+-----+
| lg_3               | newdata.dat | CLUSTER_NODE=3 |
| lg_3               | newdata.dat | CLUSTER_NODE=4 |
| lg_3               | newdata2.dat | CLUSTER_NODE=3 |
| lg_3               | newdata2.dat | CLUSTER_NODE=4 |
+-----+-----+-----+
2 rows in set (0.03 sec)
```

See [Section 20.29.1, “The INFORMATION_SCHEMA FILES Table”](#).

`ALTER TABLESPACE` is useful only with Disk Data storage for MySQL Cluster. See [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#).

13.1.9. ALTER VIEW Syntax

```
ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
```

```
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement changes the definition of a view, which must exist. The syntax is similar to that for `CREATE VIEW` and the effect is the same as for `CREATE OR REPLACE VIEW`. See [Section 13.1.20, “CREATE VIEW Syntax”](#). This statement requires the `CREATE VIEW` [696] and `DROP` [696] privileges for the view, and some privilege for each column referred to in the `SELECT` statement. `ALTER VIEW` is permitted only to the definer or users with the `SUPER` [697] privilege.

13.1.10. CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
    | [DEFAULT] COLLATE [=] collation_name
```

`CREATE DATABASE` creates a database with the given name. To use this statement, you need the `CREATE` [695] privilege for the database. `CREATE SCHEMA` is a synonym for `CREATE DATABASE`.

An error occurs if the database exists and you did not specify `IF NOT EXISTS`.

As of MySQL 5.5.3, `CREATE DATABASE` is not permitted within a session that has an active `LOCK TABLES` statement.

`create_specification` options specify database characteristics. Database characteristics are stored in the `db.opt` file in the database directory. The `CHARACTER SET` clause specifies the default database character set. The `COLLATE` clause specifies the default database collation. [Section 10.1, “Character Set Support”](#), discusses character set and collation names.

A database in MySQL is implemented as a directory containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the `CREATE DATABASE` statement creates only a directory under the MySQL data directory and the `db.opt` file. Rules for permissible database names are given in [Section 9.2, “Schema Object Names”](#). If a database name contains special characters, the name for the database directory contains encoded versions of those characters as described in [Section 9.2.3, “Mapping of Identifiers to File Names”](#).

If you manually create a directory under the data directory (for example, with `mkdir`), the server considers it a database directory and it shows up in the output of `SHOW DATABASES`.

You can also use the `mysqladmin` program to create databases. See [Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”](#).

13.1.11. CREATE EVENT Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'comment']
    DO event_body;
```

```

schedule:
  AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
  [STARTS timestamp [+ INTERVAL interval] ...]
  [ENDS timestamp [+ INTERVAL interval] ...]

interval:
  quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
            WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
            DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}

```

This statement creates and schedules a new event. The event will not run unless the Event Scheduler is enabled. For information about checking Event Scheduler status and enabling it if necessary, see [Section 19.4.2, “Event Scheduler Configuration”](#).

`CREATE EVENT` requires the [EVENT \[696\]](#) privilege for the schema in which the event is to be created. It might also require the [SUPER \[697\]](#) privilege, depending on the [DEFINER](#) value, as described later in this section.

The minimum requirements for a valid `CREATE EVENT` statement are as follows:

- The keywords `CREATE EVENT` plus an event name, which uniquely identifies the event in a database schema.
- An `ON SCHEDULE` clause, which determines when and how often the event executes.
- A `DO` clause, which contains the SQL statement to be executed by an event.

This is an example of a minimal `CREATE EVENT` statement:

```

CREATE EVENT myevent
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
  DO
    UPDATE myschema.mytable SET mycol = mycol + 1;

```

The previous statement creates an event named `myevent`. This event executes once—one hour following its creation—by running an SQL statement that increments the value of the `myschema.mytable` table's `mycol` column by 1.

The `event_name` must be a valid MySQL identifier with a maximum length of 64 characters. Event names are not case sensitive, so you cannot have two events named `myevent` and `MyEvent` in the same schema. In general, the rules governing event names are the same as those for names of stored routines. See [Section 9.2, “Schema Object Names”](#).

An event is associated with a schema. If no schema is indicated as part of `event_name`, the default (current) schema is assumed. To create an event in a specific schema, qualify the event name with a schema using `schema_name.event_name` syntax.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at event execution time. If a `user` value is given, it should be a MySQL account specified as `'user_name'@'host_name'` (the same format used in the `GRANT` statement), [CURRENT_USER \[1192\]](#), or [CURRENT_USER\(\) \[1192\]](#). The default `DEFINER` value is the user who executes the `CREATE EVENT` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the valid `DEFINER` user values:

- If you do not have the [SUPER \[697\]](#) privilege, the only permitted `user` value is your own account, either specified literally or by using [CURRENT_USER \[1192\]](#). You cannot set the definer to some other account.

- If you have the [SUPER \[697\]](#) privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.
- Although it is possible to create an event with a nonexistent [DEFINER](#) account, an error occurs at event execution time if the account does not exist.

For more information about event security, see [Section 19.6, “Access Control for Stored Programs and Views”](#).

Within an event, the [CURRENT_USER\(\) \[1192\]](#) function returns the account used to check privileges at event execution time, which is the [DEFINER](#) user. For information about user auditing within events, see [Section 6.3.11, “SQL-Based MySQL Account Activity Auditing”](#).

[IF NOT EXISTS](#) has the same meaning for [CREATE EVENT](#) as for [CREATE TABLE](#): If an event named *event_name* already exists in the same schema, no action is taken, and no error results. (However, a warning is generated in such cases.)

The [ON SCHEDULE](#) clause determines when, how often, and for how long the *event_body* defined for the event repeats. This clause takes one of two forms:

- [AT timestamp](#) is used for a one-time event. It specifies that the event executes one time only at the date and time given by *timestamp*, which must include both the date and time, or must be an expression that resolves to a datetime value. You may use a value of either the [DATETIME](#) or [TIMESTAMP](#) type for this purpose. If the date is in the past, a warning occurs, as shown here:

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2006-02-10 23:59:01 |
+-----+
1 row in set (0.04 sec)

mysql> CREATE EVENT e_totals
  ->     ON SCHEDULE AT '2006-02-10 23:59:00'
  ->     DO INSERT INTO test.totals VALUES (NOW());
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1588
Message: Event execution time is in the past and ON COMPLETION NOT
PRESERVE is set. The event was dropped immediately after
creation.
```

[CREATE EVENT](#) statements which are themselves invalid—for whatever reason—fail with an error.

You may use [CURRENT_TIMESTAMP \[1135\]](#) to specify the current date and time. In such a case, the event acts as soon as it is created.

To create an event which occurs at some point in the future relative to the current date and time—such as that expressed by the phrase “three weeks from now”—you can use the optional clause [+ INTERVAL interval](#). The *interval* portion consists of two parts, a quantity and a unit of time, and follows the same syntax rules that govern intervals used in the [DATE_ADD\(\) \[1135\]](#) function (see [Section 12.7, “Date and Time Functions”](#)). The units keywords are also the same, except that you cannot use any units involving microseconds when defining an event. With some interval types, complex time units may be used. For example, “two minutes and ten seconds” can be expressed as [+ INTERVAL '2:10' MINUTE_SECOND](#).

You can also combine intervals. For example, `AT CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY` is equivalent to “three weeks and two days from now”. Each portion of such a clause must begin with `+ INTERVAL`.

- To repeat actions at a regular interval, use an `EVERY` clause. The `EVERY` keyword is followed by an *interval* as described in the previous discussion of the `AT` keyword. (`+ INTERVAL` is *not* used with `EVERY`.) For example, `EVERY 6 WEEK` means “every six weeks”.

Although `+ INTERVAL` clauses are not permitted in an `EVERY` clause, you can use the same complex time units permitted in a `+ INTERVAL`.

An `EVERY` clause may contain an optional `STARTS` clause. `STARTS` is followed by a *timestamp* value that indicates when the action should begin repeating, and may also use `+ INTERVAL interval` to specify an amount of time “from now”. For example, `EVERY 3 MONTH STARTS CURRENT_TIMESTAMP + INTERVAL 1 WEEK` means “every three months, beginning one week from now”. Similarly, you can express “every two weeks, beginning six hours and fifteen minutes from now” as `EVERY 2 WEEK STARTS CURRENT_TIMESTAMP + INTERVAL '6:15' HOUR_MINUTE`. Not specifying `STARTS` is the same as using `STARTS CURRENT_TIMESTAMP`—that is, the action specified for the event begins repeating immediately upon creation of the event.

An `EVERY` clause may contain an optional `ENDS` clause. The `ENDS` keyword is followed by a *timestamp* value that tells MySQL when the event should stop repeating. You may also use `+ INTERVAL interval` with `ENDS`; for instance, `EVERY 12 HOUR STARTS CURRENT_TIMESTAMP + INTERVAL 30 MINUTE ENDS CURRENT_TIMESTAMP + INTERVAL 4 WEEK` is equivalent to “every twelve hours, beginning thirty minutes from now, and ending four weeks from now”. Not using `ENDS` means that the event continues executing indefinitely.

`ENDS` supports the same syntax for complex time units as `STARTS` does.

You may use `STARTS`, `ENDS`, both, or neither in an `EVERY` clause.

If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the `GET_LOCK()` [1199] function, or row or table locking.

The `ON SCHEDULE` clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *timestamp* or *interval* values which it contains. You may not use stored functions or user-defined functions in such expressions, nor may you use any table references; however, you may use `SELECT FROM DUAL`. This is true for both `CREATE EVENT` and `ALTER EVENT` statements. References to stored functions, user-defined functions, and tables in such cases are specifically not permitted, and fail with an error (see Bug #22830).

Times in the `ON SCHEDULE` clause are interpreted using the current session `time_zone` [567] value. This becomes the event time zone; that is, the time zone that is used for event scheduling and is in effect within the event as it executes. These times are converted to UTC and stored along with the event time zone in the `mysql.event` table. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. For additional information about representation of event times, see Section 19.4.4, “Event Metadata”. See also Section 13.7.5.19, “`SHOW EVENTS` Syntax”, and Section 20.7, “The `INFORMATION_SCHEMA` EVENTS Table”.

Normally, once an event has expired, it is immediately dropped. You can override this behavior by specifying `ON COMPLETION PRESERVE`. Using `ON COMPLETION NOT PRESERVE` merely makes the default nonpersistent behavior explicit.

You can create an event but prevent it from being active using the `DISABLE` keyword. Alternatively, you can use `ENABLE` to make explicit the default status, which is active. This is most useful in conjunction with `ALTER EVENT` (see [Section 13.1.2, “ALTER EVENT Syntax”](#)).

A third value may also appear in place of `ENABLED` or `DISABLED`; `DISABLE ON SLAVE` is set for the status of an event on a replication slave to indicate that the event was created on the master and replicated to the slave, but is not executed on the slave. See [Section 16.4.1.10, “Replication of Invoked Features”](#).

You may supply a comment for an event using a `COMMENT` clause. `comment` may be any string of up to 64 characters that you wish to use for describing the event. The comment text, being a string literal, must be surrounded by quotation marks.

The `DO` clause specifies an action carried by the event, and consists of an SQL statement. Nearly any valid MySQL statement that can be used in a stored routine can also be used as the action statement for a scheduled event. (See [Section E.1, “Restrictions on Stored Programs”](#).) For example, the following event `e_hourly` deletes all rows from the `sessions` table once per hour, where this table is part of the `site_activity` schema:

```
CREATE EVENT e_hourly
ON SCHEDULE
  EVERY 1 HOUR
  COMMENT 'Clears out sessions table each hour.'
DO
  DELETE FROM site_activity.sessions;
```

MySQL stores the `sql_mode` [\[556\]](#) system variable setting that is in effect at the time an event is created, and always executes the event with this setting in force, *regardless of the current server SQL mode*.

A `CREATE EVENT` statement that contains an `ALTER EVENT` statement in its `DO` clause appears to succeed; however, when the server attempts to execute the resulting scheduled event, the execution fails with an error.



Note

Statements such as `SELECT` or `SHOW` that merely return a result set have no effect when used in an event; the output from these is not sent to the MySQL Monitor, nor is it stored anywhere. However, you can use statements such as `SELECT ... INTO` and `INSERT INTO ... SELECT` that store a result. (See the next example in this section for an instance of the latter.)

The schema to which an event belongs is the default schema for table references in the `DO` clause. Any references to tables in other schemas must be qualified with the proper schema name.

As with stored routines, you can use compound-statement syntax in the `DO` clause by using the `BEGIN` and `END` keywords, as shown here:

```
delimiter |
CREATE EVENT e_daily
ON SCHEDULE
  EVERY 1 DAY
  COMMENT 'Saves total number of sessions then clears the table each day'
DO
  BEGIN
    INSERT INTO site_activity.totals (time, total)
      SELECT CURRENT_TIMESTAMP, COUNT(*)
        FROM site_activity.sessions;
    DELETE FROM site_activity.sessions;
  END |
```

```
delimiter ;
```

This example uses the `delimiter` command to change the statement delimiter. See [Section 19.1, “Defining Stored Programs”](#).

More complex compound statements, such as those used in stored routines, are possible in an event. This example uses local variables, an error handler, and a flow control construct:

```
delimiter |
CREATE EVENT e
ON SCHEDULE
  EVERY 5 SECOND
DO
  BEGIN
    DECLARE v INTEGER;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;

    SET v = 0;

    WHILE v < 5 DO
      INSERT INTO t1 VALUES (0);
      UPDATE t2 SET s1 = s1 + 1;
      SET v = v + 1;
    END WHILE;
  END |
delimiter ;
```

There is no way to pass parameters directly to or from events; however, it is possible to invoke a stored routine with parameters within an event:

```
CREATE EVENT e_call_myproc
ON SCHEDULE
  AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
DO CALL myproc(5, 27);
```

If an event's definer has the [SUPER \[697\]](#) privilege, the event can read and write global variables. As granting this privilege entails a potential for abuse, extreme care must be taken in doing so.

Generally, any statements that are valid in stored routines may be used for action statements executed by events. For more information about statements permissible within stored routines, see [Section 19.2.1, “Stored Routine Syntax”](#). You can create an event as part of a stored routine, but an event cannot be created by another event.

13.1.12. CREATE FUNCTION Syntax

The `CREATE FUNCTION` statement is used to create stored functions and user-defined functions (UDFs):

- For information about creating stored functions, see [Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).
- For information about creating user-defined functions, see [Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”](#).

13.1.13. CREATE INDEX Syntax

```
CREATE [ONLINE|OFFLINE] [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
  [index_type]
  ON tbl_name (index_col_name,...)
```

```

[index_option] ...

index_col_name:
  col_name [(length)] [ASC | DESC]

index_type:
  USING {BTREE | HASH}

index_option:
  KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'

```

`CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See [Section 13.1.7, “ALTER TABLE Syntax”](#). `CREATE INDEX` cannot be used to create a `PRIMARY KEY`; use `ALTER TABLE` instead. For more information about indexes, see [Section 8.3.1, “How MySQL Uses Indexes”](#).

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See [Section 13.1.17, “CREATE TABLE Syntax”](#). This guideline is especially important for `InnoDB` tables, where the primary key determines the physical layout of rows in the data file. `CREATE INDEX` enables you to add indexes to existing tables.

A column list of the form `(col1,col2,...)` creates a multiple-column index. Index key values are formed by concatenating the values of the given columns.

Indexes can be created that use only the leading part of column values, using `col_name(length)` syntax to specify an index prefix length:

- Prefixes can be specified for `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` columns.
- `BLOB` and `TEXT` columns also can be indexed, but a prefix length *must* be given.
- Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for `CHAR`, `VARCHAR`, and `TEXT` columns, and the first *length* bytes of each column value for `BINARY`, `VARBINARY`, and `BLOB` columns.
- For spatial columns, prefix values cannot be given, as described later in this section.

The statement shown here creates an index using the first 10 characters of the `name` column:

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, this index should not be much slower than an index created from the entire `name` column. Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up `INSERT` operations.

Prefix support and lengths of prefixes (where supported) are storage engine dependent. For example, a prefix can be up to 1000 bytes long for `MyISAM` tables, and 767 bytes for `InnoDB` tables. The `NDBCLUSTER` storage engine does not support prefixes (see [Section 17.1.6.6, “Unsupported or Missing Features in MySQL Cluster”](#)).



Note

Prefix limits are measured in bytes, whereas the prefix length in `CREATE INDEX` statements is interpreted as number of characters for nonbinary data types (`CHAR`, `VARCHAR`, `TEXT`). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

Indexes on variable-width columns of `NDBCLUSTER` tables are created online; that is, without any table copying. The table is not locked against access from other MySQL Cluster API nodes, although it is locked against other operations on the *same* API node for the duration of the operation. This is done automatically by the server whenever it determines that it is possible to do so; you do not have to use any special SQL syntax or server options to cause it to happen.

In standard MySQL 5.5 releases, it is not possible to override the server when it determines that an index is to be created without table copying. In MySQL Cluster, you can create indexes offline (which causes the table to be locked to all API nodes in the cluster) using the `OFFLINE` keyword. The rules and limitations governing `CREATE OFFLINE INDEX` and `CREATE ONLINE INDEX` are the same as for `ALTER OFFLINE TABLE ... ADD INDEX` and `ALTER ONLINE TABLE ... ADD INDEX`. You cannot cause the noncopying creation of an index that would normally be created offline by using the `ONLINE` keyword: If it is not possible to perform the `CREATE INDEX` operation without table copying, the server ignores the `ONLINE` keyword. For more information, see [Section 13.1.7.2, “ALTER TABLE Online Operations”](#).



Note

The `ONLINE` and `OFFLINE` keywords are available only in MySQL Cluster; attempting to use these keywords in standard MySQL Server 5.5 releases results in a syntax error.

A `UNIQUE` index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a `UNIQUE` index permits multiple `NULL` values for columns that can contain `NULL`. If you specify a prefix value for a column in a `UNIQUE` index, the column values must be unique within the prefix.

`FULLTEXT` indexes are supported only for `MyISAM` tables and can include only `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See [Section 12.9, “Full-Text Search Functions”](#), for details of operation.

The `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` storage engines support spatial columns such as (`POINT` and `GEOMETRY`. ([Section 12.17, “Spatial Extensions”](#), describes the spatial data types.) However, support for spatial column indexing varies among engines. Spatial and nonspatial indexes are available according to the following rules.

Spatial indexes (created using `SPATIAL INDEX`) have these characteristics:

- Available only for `MyISAM` tables. Specifying `SPATIAL INDEX` for other storage engines results in an error.
- Indexed columns must be `NOT NULL`.
- In MySQL 5.5, column prefix lengths are prohibited. The full width of each column is indexed.

Characteristics of nonspatial indexes (created with `INDEX`, `UNIQUE`, or `PRIMARY KEY`):

- Permitted for any storage engine that supports spatial columns except `ARCHIVE`.
- Columns can be `NULL` unless the index is a primary key.
- For each spatial column in a non-`SPATIAL` index except `POINT` columns, a column prefix length must be specified. (This is the same requirement as for indexed `BLOB` columns.) The prefix length is given in bytes.
- The index type for a non-`SPATIAL` index depends on the storage engine. Currently, B-tree is used.

In MySQL 5.5:

- You can add an index on a column that can have `NULL` values only if you are using the `MyISAM`, `InnoDB`, or `MEMORY` storage engine.
- You can add an index on a `BLOB` or `TEXT` column only if you are using the `MyISAM`, or `InnoDB` storage engine.

An `index_col_name` specification can end with `ASC` or `DESC`. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

Following the index column list, index options can be given. An `index_option` value can be any of the following:

- `KEY_BLOCK_SIZE [=] value`

For `compressed InnoDB` tables, optionally specifies the size in bytes to use for `pages`. The value is treated as a hint; a different size could be used if necessary. A value of 0 represents the default compressed page size. See [Section 14.4.3, “InnoDB Data Compression”](#) for usage details.



Note

Oracle recommends enabling `innodb_strict_mode [1572]` when using the `KEY_BLOCK_SIZE` clause for `InnoDB` tables. See [Section 14.4.8.4, “InnoDB Strict Mode”](#) for details.

- `index_type`

Some storage engines permit you to specify an index type when creating an index. The permissible index type values supported by different storage engines are shown in the following table. Where multiple index types are listed, the first one is the default when no index type specifier is given.

Storage Engine	Permissible Index Types
<code>MyISAM</code>	<code>BTREE</code>
<code>InnoDB</code>	<code>BTREE</code>
<code>MEMORY/HEAP</code>	<code>HASH</code> , <code>BTREE</code>
<code>NDB</code>	<code>BTREE</code> , <code>HASH</code> (see note in text)

Example:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index ON lookup (id) USING BTREE;
```

`BTREE` indexes are implemented by the `NDBCLUSTER` storage engine as T-tree indexes.



Note

For indexes on `NDB` table columns, the `USING` option can be specified only for a unique index or primary key. `USING HASH` prevents the creation of an implicit ordered index; otherwise, creating a unique index or primary key on an `NDB` table automatically results in the creation of both an ordered index and a hash index, each of which indexes the same set of columns.

This means that a query using a unique index or primary key on a `NULL` column is always handled by `NDB` with a full scan of the table. In particular, if you plan to

use an `IS NULL` or `IS NOT NULL` condition involving a unique index or primary key column of an `NDB` table, you should create any such index without `USING HASH`.

The `index_type` clause cannot be used together with `SPATIAL INDEX`.

If you specify an index type that is not valid for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type. The parser recognizes `RTREE` as a type name, but currently this cannot be specified for any storage engine.

Use of this option before the `ON tbl_name` clause is deprecated; support for use of the option in this position will be removed in a future MySQL release. If an `index_type` option is given in both the earlier and later positions, the final option applies.

`TYPE type_name` is recognized as a synonym for `USING type_name`. However, `USING` is the preferred form.

- `WITH PARSER parser_name`

This option can be used only with `FULLTEXT` indexes. It associates a parser plugin with the index if full-text indexing and searching operations need special handling. See [Section 23.2, “The MySQL Plugin API”](#), for details on creating plugins.

- `COMMENT 'string'`

As of MySQL 5.5.3, index definitions can include an optional comment of up to 1024 characters.

13.1.14. CREATE LOGFILE GROUP Syntax

```
CREATE LOGFILE GROUP logfile_group
  ADD UNDOFILE 'undo_file'
  [INITIAL_SIZE [=] initial_size]
  [UNDO_BUFFER_SIZE [=] undo_buffer_size]
  [REDO_BUFFER_SIZE [=] redo_buffer_size]
  [NODEGROUP [=] nodegroup_id]
  [WAIT]
  [COMMENT [=] comment_text]
  ENGINE [=] engine_name
```

This statement creates a new log file group named `logfile_group` having a single `UNDO` file named `'undo_file'`. A `CREATE LOGFILE GROUP` statement has one and only one `ADD UNDOFILE` clause. For rules covering the naming of log file groups, see [Section 9.2, “Schema Object Names”](#).



Note

All MySQL Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group with the same name, or a tablespace and a data file with the same name.

In MySQL Cluster NDB 7.2, you can have only one log file group per Cluster at any given time. (See Bug #16386)

The optional `INITIAL_SIZE` parameter sets the `UNDO` file's initial size; if not specified, it defaults to `128M` (128 megabytes). The optional `UNDO_BUFFER_SIZE` parameter sets the size used by the `UNDO` buffer for the log file group; The default value for `UNDO_BUFFER_SIZE` is `8M` (eight megabytes); this value cannot exceed the amount of system memory available. Both of these parameters are specified in bytes. You may

optionally follow either or both of these with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (for megabytes) or `G` (for gigabytes).

The memory used for both `INITIAL_SIZE` and `UNDO_BUFFER_SIZE` comes from the global pool whose size is determined by the value of the `SharedGlobalMemory` [2083] data node configuration parameter. This includes any default value implied for these options by the setting of the `InitialLogFileGroup` [2085] data node configuration parameter.

The maximum permitted for `UNDO_BUFFER_SIZE` is `600M`.

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is `4G`.

The minimum permitted value for `INITIAL_SIZE` is `1M`.

The `ENGINE` option determines the storage engine to be used by this log file group, with `engine_name` being the name of the storage engine. In MySQL 5.5, this must be `NDB` (or `NDBCLUSTER`). If `ENGINE` is not set, MySQL tries to use the engine specified by the `default_storage_engine` [494] server system variable (formerly `storage_engine` [559]). In any case, if the engine is not specified as `NDB` or `NDBCLUSTER`, the `CREATE LOGFILE GROUP` statement appears to succeed but actually fails to create the log file group, as shown here:

```
mysql> CREATE LOGFILE GROUP lg1
->     ADD UNDOFILE 'undo.dat' INITIAL_SIZE = 10M;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message
+-----+-----+-----+
| Error | 1478 | Table storage engine 'InnoDB' does not support the create option 'TABLESPACE or LOGFILE GROUP
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> DROP LOGFILE GROUP lg1 ENGINE = NDB;
ERROR 1529 (HY000): Failed to drop LOGFILE GROUP

mysql> CREATE LOGFILE GROUP lg1
->     ADD UNDOFILE 'undo.dat' INITIAL_SIZE = 10M
->     ENGINE = NDB;
Query OK, 0 rows affected (2.97 sec)
```

The fact that the `CREATE LOGFILE GROUP` statement does not actually return an error when a non-`NDB` storage engine is named, but rather appears to succeed, is a known issue which we hope to address in a future release of MySQL Cluster.

`REDO_BUFFER_SIZE`, `NODEGROUP`, `WAIT`, and `COMMENT` are parsed but ignored, and so have no effect in MySQL 5.5. These options are intended for future expansion.

When used with `ENGINE [=] NDB`, a log file group and associated `UNDO` log file are created on each Cluster data node. You can verify that the `UNDO` files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example:

```
mysql> SELECT LOGFILE_GROUP_NAME, LOGFILE_GROUP_NUMBER, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE FILE_NAME = 'undo_10.dat';
+-----+-----+-----+
| LOGFILE_GROUP_NAME | LOGFILE_GROUP_NUMBER | EXTRA
+-----+-----+-----+
| lg_3                | 11                    | CLUSTER_NODE=3
| lg_3                | 11                    | CLUSTER_NODE=4
+-----+-----+-----+
2 rows in set (0.06 sec)
```

`CREATE LOGFILE GROUP` is useful only with Disk Data storage for MySQL Cluster. See [Section 17.5.12](#), “MySQL Cluster Disk Data Tables”.

13.1.15. CREATE PROCEDURE and CREATE FUNCTION Syntax

```
CREATE
  [DEFINER = { user | CURRENT_USER }]
  PROCEDURE sp_name ([proc_parameter[,...]])
  [characteristic ...] routine_body

CREATE
  [DEFINER = { user | CURRENT_USER }]
  FUNCTION sp_name ([func_parameter[,...]])
  RETURNS type
  [characteristic ...] routine_body

proc_parameter:
  [ IN | OUT | INOUT ] param_name type

func_parameter:
  param_name type

type:
  Any valid MySQL data type

characteristic:
  COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

routine_body:
Valid SQL routine statement

These statements create stored routines. By default, a routine is associated with the default database. To associate the routine explicitly with a given database, specify the name as `db_name.sp_name` when you create it.

The `CREATE FUNCTION` statement is also used in MySQL to support UDFs (user-defined functions). See [Section 23.3](#), “Adding New Functions to MySQL”. A UDF can be regarded as an external stored function. Stored functions share their namespace with UDFs. See [Section 9.2.4](#), “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

To invoke a stored procedure, use the `CALL` statement (see [Section 13.2.1](#), “CALL Syntax”). To invoke a stored function, refer to it in an expression. The function returns a value during expression evaluation.

`CREATE PROCEDURE` and `CREATE FUNCTION` require the `CREATE ROUTINE` [695] privilege. They might also require the `SUPER` [697] privilege, depending on the `DEFINER` value, as described later in this section. If binary logging is enabled, `CREATE FUNCTION` might require the `SUPER` [697] privilege, as described in [Section 19.7](#), “Binary Logging of Stored Programs”.

By default, MySQL automatically grants the `ALTER ROUTINE` [695] and `EXECUTE` [696] privileges to the routine creator. This behavior can be changed by disabling the `automatic_sp_privileges` [484] system variable. See [Section 19.2.2](#), “Stored Routines and MySQL Privileges”.

The `DEFINER` and `SQL SECURITY` clauses specify the security context to be used when checking access privileges at routine execution time, as described later in this section.

If the routine name is the same as the name of a built-in SQL function, a syntax error occurs unless you use a space between the name and the following parenthesis when defining the routine or invoking it later. For this reason, avoid using the names of existing SQL functions for your own stored routines.

The `IGNORE_SPACE` [613] SQL mode applies to built-in functions, not to stored routines. It is always permissible to have spaces after a stored routine name, regardless of whether `IGNORE_SPACE` [613] is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of `()` should be used. Parameter names are not case sensitive.

Each parameter is an `IN` parameter by default. To specify otherwise for a parameter, use the keyword `OUT` or `INOUT` before the parameter name.



Note

Specifying a parameter as `IN`, `OUT`, or `INOUT` is valid only for a `PROCEDURE`. For a `FUNCTION`, parameters are always regarded as `IN` parameters.

An `IN` parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An `OUT` parameter passes a value from the procedure back to the caller. Its initial value is `NULL` within the procedure, and its value is visible to the caller when the procedure returns. An `INOUT` parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each `OUT` or `INOUT` parameter, pass a user-defined variable in the `CALL` statement that invokes the procedure so that you can obtain its value when the procedure returns. If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.

The following example shows a simple stored procedure that uses an `OUT` parameter:

```
mysql> delimiter //
mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
-> BEGIN
->   SELECT COUNT(*) INTO param1 FROM t;
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+-----+
| @a    |
+-----+
| 3     |
+-----+
1 row in set (0.00 sec)
```

The example uses the `mysql` client `delimiter` command to change the statement delimiter from `;` to `//` while the procedure is being defined. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself. See [Section 19.1, “Defining Stored Programs”](#).

The `RETURNS` clause may be specified only for a `FUNCTION`, for which it is mandatory. It indicates the return type of the function, and the function body must contain a `RETURN value` statement. If the `RETURN` statement returns a value of a different type, the value is coerced to the proper type. For example, if a function specifies an `ENUM` or `SET` value in the `RETURNS` clause, but the `RETURN` statement returns an integer, the value returned from the function is the string for the corresponding `ENUM` member of set of `SET` members.

The following example function takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
      -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world!  |
+-----+
1 row in set (0.00 sec)
```

Parameter types and function return types can be declared to use any valid data type, except that the `COLLATE` attribute cannot be used prior to MySQL 5.5.3. As of 5.5.3, `COLLATE` can be used if preceded by the `CHARACTER SET` attribute.

The `routine_body` consists of a valid SQL routine statement. This can be a simple statement such as `SELECT` or `INSERT`, or a compound statement written using `BEGIN` and `END`. Compound statements can contain declarations, loops, and other control structure statements. The syntax for these statements is described in [Section 13.6, “MySQL Compound-Statement Syntax”](#).

MySQL permits routines to contain DDL statements, such as `CREATE` and `DROP`. MySQL also permits stored procedures (but not stored functions) to contain SQL transaction statements such as `COMMIT`. Stored functions may not contain statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.

Statements that return a result set can be used within a stored procedure but not within a stored function. This prohibition includes `SELECT` statements that do not have an `INTO var_list` clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. For statements that can be determined at function definition time to return a result set, a `Not allowed to return a result set from a function` error occurs (`ER_SP_NO_RETSET [4007]`). For statements that can be determined only at runtime to return a result set, a `PROCEDURE %s can't return a result set in the given context` error occurs (`ER_SP_BADSELECT [4000]`).

`USE` statements within stored routines are not permitted. When a routine is invoked, an implicit `USE db_name` is performed (and undone when the routine terminates). This causes the routine to have the given default database while it executes. References to objects in databases other than the routine default database should be qualified with the appropriate database name.

For additional information about statements that are not permitted in stored routines, see [Section E.1, “Restrictions on Stored Programs”](#).

For information about invoking stored procedures from within programs written in a language that has a MySQL interface, see [Section 13.2.1, “CALL Syntax”](#).

MySQL stores the `sql_mode [556]` system variable setting that is in effect at the time a routine is created, and always executes the routine with this setting in force, *regardless of the server SQL mode in effect when the routine is invoked*.

The switch from the SQL mode of the invoker to that of the routine occurs after evaluation of arguments and assignment of the resulting values to routine parameters. If you define a routine in strict SQL mode but invoke it in nonstrict mode, assignment of arguments to routine parameters does not take place in strict

mode. If you require that expressions passed to a routine be assigned in strict SQL mode, you should invoke the routine with strict mode in effect.

The `COMMENT` characteristic is a MySQL extension, and may be used to describe the stored routine. This information is displayed by the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements.

The `LANGUAGE` characteristic indicates the language in which the routine is written. The server ignores this characteristic; only SQL routines are supported.

A routine is considered “deterministic” if it always produces the same result for the same input parameters, and “not deterministic” otherwise. If neither `DETERMINISTIC` nor `NOT DETERMINISTIC` is given in the routine definition, the default is `NOT DETERMINISTIC`. To declare that a function is deterministic, you must specify `DETERMINISTIC` explicitly.

Assessment of the nature of a routine is based on the “honesty” of the creator: MySQL does not check that a routine declared `DETERMINISTIC` is free of statements that produce nondeterministic results. However, misdeclaring a routine might affect results or affect performance. Declaring a nondeterministic routine as `DETERMINISTIC` might lead to unexpected results by causing the optimizer to make incorrect execution plan choices. Declaring a deterministic routine as `NONDETERMINISTIC` might diminish performance by causing available optimizations not to be used.

If binary logging is enabled, the `DETERMINISTIC` characteristic affects which routine definitions MySQL accepts. See [Section 19.7, “Binary Logging of Stored Programs”](#).

A routine that contains the `NOW()` [1143] function (or its synonyms) or `RAND()` [1128] is nondeterministic, but it might still be replication-safe. For `NOW()` [1143], the binary log includes the timestamp and replicates correctly. `RAND()` [1128] also replicates correctly as long as it is called only a single time during the execution of a routine. (You can consider the routine execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

Several characteristics provide information about the nature of data use by the routine. In MySQL, these characteristics are advisory only. The server does not use them to constrain what kinds of statements a routine will be permitted to execute.

- `CONTAINS SQL` indicates that the routine does not contain statements that read or write data. This is the default if none of these characteristics is given explicitly. Examples of such statements are `SET @x = 1` or `DO RELEASE_LOCK('abc')`, which execute but neither read nor write data.
- `NO SQL` indicates that the routine contains no SQL statements.
- `READS SQL DATA` indicates that the routine contains statements that read data (for example, `SELECT`), but not statements that write data.
- `MODIFIES SQL DATA` indicates that the routine contains statements that may write data (for example, `INSERT` or `DELETE`).

The `SQL SECURITY` characteristic can be `DEFINER` or `INVOKER` to specify the security context; that is, whether the routine executes using the privileges of the account named in the routine `DEFINER` clause or the user who invokes it. This account must have permission to access the database with which the routine is associated. The default value is `DEFINER`. The user who invokes the routine must have the `EXECUTE` [696] privilege for it, as must the `DEFINER` account if the routine executes in definer security context.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at routine execution time for routines that have the `SQL SECURITY DEFINER` characteristic.

If a `user` value is given for the `DEFINER` clause, it should be a MySQL account specified as `'user_name'@'host_name'` (the same format used in the `GRANT` statement), `CURRENT_USER` [1192], or

`CURRENT_USER()` [1192]. The default `DEFINER` value is the user who executes the `CREATE PROCEDURE` or `CREATE FUNCTION` or statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the valid `DEFINER` user values:

- If you do not have the `SUPER` [697] privilege, the only permitted `user` value is your own account, either specified literally or by using `CURRENT_USER` [1192]. You cannot set the definer to some other account.
- If you have the `SUPER` [697] privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.
- Although it is possible to create a routine with a nonexistent `DEFINER` account, an error occurs at routine execution time if the `SQL SECURITY` value is `DEFINER` but the definer account does not exist.

For more information about stored routine security, see [Section 19.6, “Access Control for Stored Programs and Views”](#).

Within a stored routine that is defined with the `SQL SECURITY DEFINER` characteristic, `CURRENT_USER` [1192] returns the routine's `DEFINER` value. For information about user auditing within stored routines, see [Section 6.3.11, “SQL-Based MySQL Account Activity Auditing”](#).

Consider the following procedure, which displays a count of the number of MySQL accounts listed in the `mysql.user` table:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure is assigned a `DEFINER` account of `'admin'@'localhost'` no matter which user defines it. It executes with the privileges of that account no matter which user invokes it (because the default security characteristic is `DEFINER`). The procedure succeeds or fails depending on whether invoker has the `EXECUTE` [696] privilege for it and `'admin'@'localhost'` has the `SELECT` [697] privilege for the `mysql.user` table.

Now suppose that the procedure is defined with the `SQL SECURITY INVOKER` characteristic:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
SQL SECURITY INVOKER
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure still has a `DEFINER` of `'admin'@'localhost'`, but in this case, it executes with the privileges of the invoking user. Thus, the procedure succeeds or fails depending on whether the invoker has the `EXECUTE` [696] privilege for it and the `SELECT` [697] privilege for the `mysql.user` table.

The server handles the data type of a routine parameter, local routine variable created with `DECLARE`, or function return value as follows:

- Assignments are checked for data type mismatches and overflow. Conversion and overflow problems result in warnings, or errors in strict SQL mode.
- Only scalar values can be assigned. For example, a statement such as `SET x = (SELECT 1, 2)` is invalid.
- For character data types, if there is a `CHARACTER SET` attribute in the declaration, the specified character set and its default collation is used. If the `COLLATE` attribute is also present, that collation is used rather than the default collation. If there is no `CHARACTER SET` attribute, the database character set and collation in effect at routine creation time are used. (The database character set and collation are

given by the value of the `character_set_database` [487] and `collation_database` [489] system variables.)

Prior to MySQL 5.5.3, if there is a `CHARACTER SET` attribute in the declaration, the `COLLATE` attribute is not supported, and the character set's default collation is used. (This includes use of `BINARY`, which in this context specifies the binary collation of the character set.) If there is no `CHARACTER SET` attribute, the database character set and its default collation (rather than the database collation) are used.

If you change the database default character set or collation, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults.

13.1.16. CREATE SERVER Syntax

```
CREATE SERVER server_name
  FOREIGN DATA WRAPPER wrapper_name
  OPTIONS (option [, option] ...)

option:
{ HOST character-literal
| DATABASE character-literal
| USER character-literal
| PASSWORD character-literal
| SOCKET character-literal
| OWNER character-literal
| PORT numeric-literal }
```

This statement creates the definition of a server for use with the `FEDERATED` storage engine. The `CREATE SERVER` statement creates a new row in the `servers` table in the `mysql` database. This statement requires the `SUPER` [697] privilege.

The `server_name` should be a unique reference to the server. Server definitions are global within the scope of the server, it is not possible to qualify the server definition to a specific database. `server_name` has a maximum length of 64 characters (names longer than 64 characters are silently truncated), and is case insensitive. You may specify the name as a quoted string.

The `wrapper_name` should be `mysql`, and may be quoted with single quotation marks. Other values for `wrapper_name` are not currently supported.

For each `option` you must specify either a character literal or numeric literal. Character literals are UTF-8, support a maximum length of 64 characters and default to a blank (empty) string. String literals are silently truncated to 64 characters. Numeric literals must be a number between 0 and 9999, default value is 0.



Note

The `OWNER` option is currently not applied, and has no effect on the ownership or operation of the server connection that is created.

The `CREATE SERVER` statement creates an entry in the `mysql.servers` table that can later be used with the `CREATE TABLE` statement when creating a `FEDERATED` table. The options that you specify will be used to populate the columns in the `mysql.servers` table. The table columns are `Server_name`, `Host`, `Db`, `Username`, `Password`, `Port` and `Socket`.

For example:

```
CREATE SERVER s
  FOREIGN DATA WRAPPER mysql
  OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
```

Be sure to specify all options necessary to establish a connection to the server. The user name, host name, and database name are mandatory. Other options might be required as well, such as password.

The data stored in the table can be used when creating a connection to a `FEDERATED` table:

```
CREATE TABLE t (s1 INT) ENGINE=FEDERATED CONNECTION='s';
```

For more information, see [Section 14.11, “The `FEDERATED` Storage Engine”](#).

`CREATE SERVER` does not cause an automatic commit.

In MySQL 5.5, `CREATE SERVER` is not written to the binary log, regardless of the logging format that is in use.

13.1.17. CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    [partition_options]
    select_statement
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    { LIKE old_tbl_name | (LIKE old_tbl_name) }
```

```
create_definition:
    col_name column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
    [index_option] ...
  | {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
    [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
    [index_name] [index_type] (index_col_name,...)
    [index_option] ...
  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)
    [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name,...) reference_definition
  | CHECK (expr)
```

```
column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
    [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
    [COMMENT 'string']
    [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
    [STORAGE {DISK|MEMORY|DEFAULT}]
    [reference_definition]
```

```
data_type:
    BIT[(length)]
  | TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
```

```

| NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
| DATE
| TIME
| TIMESTAMP
| DATETIME
| YEAR
| CHAR[(length)]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| VARCHAR(length)
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| BINARY[(length)]
| VARBINARY(length)
| TINYBLOB
| BLOB
| MEDIUMBLOB
| LONGBLOB
| TINYTEXT [BINARY]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| TEXT [BINARY]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| MEDIUMTEXT [BINARY]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| LONGTEXT [BINARY]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| ENUM(value1,value2,value3,...)
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| SET(value1,value2,value3,...)
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| spatial_type

index_col_name:
  col_name [(length)] [ASC | DESC]

index_type:
  USING {BTREE | HASH}

index_option:
  KEY_BLOCK_SIZE [=] value
| index_type
| WITH PARSER parser_name
| COMMENT 'string'

reference_definition:
  REFERENCES tbl_name (index_col_name,...)
  [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
  [ON DELETE reference_option]
  [ON UPDATE reference_option]

reference_option:
  RESTRICT | CASCADE | SET NULL | NO ACTION

table_options:
  table_option [[,] table_option] ...

table_option:
  ENGINE [=] engine_name
| AUTO_INCREMENT [=] value
| AVG_ROW_LENGTH [=] value
| [DEFAULT] CHARACTER SET [=] charset_name
| CHECKSUM [=] {0 | 1}
| [DEFAULT] COLLATE [=] collation_name
| COMMENT [=] 'string'
| CONNECTION [=] 'connect_string'
| DATA DIRECTORY [=] 'absolute path to directory'
| DELAY_KEY_WRITE [=] {0 | 1}
| INDEX DIRECTORY [=] 'absolute path to directory'
| INSERT_METHOD [=] { NO | FIRST | LAST }

```

```

| KEY_BLOCK_SIZE [=] value
| MAX_ROWS [=] value
| MIN_ROWS [=] value
| PACK_KEYS [=] {0 | 1 | DEFAULT}
| PASSWORD [=] 'string'
| ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
| TABLESPACE tablespace_name [STORAGE {DISK|MEMORY|DEFAULT}]
| UNION [=] (tbl_name[, tbl_name]...)

partition_options:
PARTITION BY
  { [LINEAR] HASH(expr)
  | [LINEAR] KEY(column_list)
  | RANGE{(expr) | COLUMNS(column_list)}
  | LIST{(expr) | COLUMNS(column_list)} }
[PARTITIONS num]
[SUBPARTITION BY
  { [LINEAR] HASH(expr)
  | [LINEAR] KEY(column_list) }
[SUBPARTITIONS num]
]
[(partition_definition [, partition_definition] ...)]

partition_definition:
PARTITION partition_name
  [VALUES
    {LESS THAN {(expr | value_list) | MAXVALUE}
    |
    IN (value_list)}]
  [[STORAGE] ENGINE [=] engine_name]
  [COMMENT [=] 'comment_text' ]
  [DATA DIRECTORY [=] 'data_dir' ]
  [INDEX DIRECTORY [=] 'index_dir' ]
  [MAX_ROWS [=] max_number_of_rows]
  [MIN_ROWS [=] min_number_of_rows]
  [TABLESPACE [=] tablespace_name]
  [NODEGROUP [=] node_group_id]
  [(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
SUBPARTITION logical_name
  [[STORAGE] ENGINE [=] engine_name]
  [COMMENT [=] 'comment_text' ]
  [DATA DIRECTORY [=] 'data_dir' ]
  [INDEX DIRECTORY [=] 'index_dir' ]
  [MAX_ROWS [=] max_number_of_rows]
  [MIN_ROWS [=] min_number_of_rows]
  [TABLESPACE [=] tablespace_name]
  [NODEGROUP [=] node_group_id]

select_statement:
[IGNORE | REPLACE] [AS] SELECT ... (Some valid select statement)

```

CREATE TABLE creates a table with the given name. You must have the **CREATE** [695] privilege for the table.

Rules for permissible table names are given in [Section 9.2, “Schema Object Names”](#). By default, the table is created in the default database, using the **InnoDB** storage engine. An error occurs if the table exists, if there is no default database, or if the database does not exist.

The table name can be specified as *db_name.tbl_name* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists. If you use quoted identifiers, quote the database and table names separately. For example, write ``mydb`.`mytbl``, not ``mydb.mytbl``.

You can use the `TEMPORARY` keyword when creating a table. A `TEMPORARY` table is visible only to the current connection, and is dropped automatically when the connection is closed. This means that two different connections can use the same temporary table name without conflicting with each other or with an existing non-`TEMPORARY` table of the same name. (The existing table is hidden until the temporary table is dropped.) To create temporary tables, you must have the `CREATE TEMPORARY TABLES` [695] privilege.



Note

`CREATE TABLE` does not automatically commit the current active transaction if you use the `TEMPORARY` keyword.

The keywords `IF NOT EXISTS` prevent an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the `CREATE TABLE` statement.

MySQL represents each table by an `.frm` table format (definition) file in the database directory. The storage engine for the table might create other files as well.

For `InnoDB` tables, the file storage is controlled by the `innodb_file_per_table` [1555] configuration option. When this option is turned off, all `InnoDB` tables and indexes are stored in the `system tablespace`, represented by one or more `.ibd` files. For each `InnoDB` table created when this option is turned on, the table data and all associated indexes are stored in a `.ibd` file located inside the database directory.

For `MyISAM` tables, the storage engine creates data and index files. Thus, for each `MyISAM` table `tbl_name`, there are three disk files.

File	Purpose
<code>tbl_name.frm</code>	Table format (definition) file
<code>tbl_name.MYD</code>	Data file
<code>tbl_name.MYI</code>	Index file

Chapter 14, *Storage Engines*, describes what files each storage engine creates to represent tables. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in Section 9.2.3, “Mapping of Identifiers to File Names”.

`data_type` represents the data type in a column definition. `spatial_type` represents a spatial data type. The data type syntax shown is representative only. For a full description of the syntax available for specifying column data types, as well as information about the properties of each type, see Chapter 11, *Data Types*, and Section 12.17, “Spatial Extensions”.

Some attributes do not apply to all data types. `AUTO_INCREMENT` applies only to integer and floating-point types. `DEFAULT` does not apply to the `BLOB` or `TEXT` types.

- If neither `NULL` nor `NOT NULL` is specified, the column is treated as though `NULL` had been specified.
- An integer or floating-point column can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is `value+1`, where `value` is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`.

To retrieve an `AUTO_INCREMENT` value after inserting a row, use the `LAST_INSERT_ID()` [1194] SQL function or the `mysql_insert_id()` C API function. See Section 12.14, “Information Functions”, and Section 22.8.3.37, “`mysql_insert_id()`”.

If the `NO_AUTO_VALUE_ON_ZERO` [613] SQL mode is enabled, you can store `0` in `AUTO_INCREMENT` columns as `0` without generating a new sequence value. See Section 5.1.7, “Server SQL Modes”.

**Note**

There can be only one `AUTO_INCREMENT` column per table, it must be indexed, and it cannot have a `DEFAULT` value. An `AUTO_INCREMENT` column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers “wrap” over from positive to negative and also to ensure that you do not accidentally get an `AUTO_INCREMENT` column that contains 0.

For `MyISAM` tables, you can specify an `AUTO_INCREMENT` secondary column in a multiple-column key. See [Section 3.6.9, “Using `AUTO_INCREMENT`”](#).

To make MySQL compatible with some ODBC applications, you can find the `AUTO_INCREMENT` value for the last inserted row with the following query:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

For information about `InnoDB` and `AUTO_INCREMENT`, see [Section 14.3.5.3, “`AUTO_INCREMENT` Handling in `InnoDB`”](#). For information about `AUTO_INCREMENT` and MySQL Replication, see [Section 16.4.1.1, “Replication and `AUTO_INCREMENT`”](#).

- Character data types (`CHAR`, `VARCHAR`, `TEXT`) can include `CHARACTER SET` and `COLLATE` attributes to specify the character set and collation for the column. For details, see [Section 10.1, “Character Set Support”](#). `CHARSET` is a synonym for `CHARACTER SET`. Example:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

MySQL 5.5 interprets length specifications in character column definitions in characters. (Versions before MySQL 4.1 interpreted them in bytes.) Lengths for `BINARY` and `VARBINARY` are in bytes.

- The `DEFAULT` clause specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` [1143] or `CURRENT_DATE` [1135]. The exception is that you can specify `CURRENT_TIMESTAMP` [1135] as the default for a `TIMESTAMP` column. See [Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP`”](#).

If a column definition includes no explicit `DEFAULT` value, MySQL determines the default value as described in [Section 11.5, “Data Type Default Values”](#).

`BLOB` and `TEXT` columns cannot be assigned a default value.

`CREATE TABLE` fails if a date-valued default is not correct according to the `NO_ZERO_IN_DATE` [615] SQL mode, even if strict SQL mode is not enabled. For example, `c1 DATE DEFAULT '2010-00-00'` causes `CREATE TABLE` to fail with `Invalid default value for 'c1'`.

- A comment for a column can be specified with the `COMMENT` option, up to 1024 characters long (255 characters before MySQL 5.5.3). The comment is displayed by the `SHOW CREATE TABLE` and `SHOW FULL COLUMNS` statements.
- In MySQL Cluster, it is also possible to specify a data storage format for individual columns of `NDB` tables using `COLUMN_FORMAT`. Permissible column formats are `FIXED`, `DYNAMIC`, and `DEFAULT`. `FIXED` is used to specify fixed-width storage, `DYNAMIC` permits the column to be variable-width, and `DEFAULT` causes the column to use fixed-width or variable-width storage as determined by the column's data type (possibly overridden by a `ROW_FORMAT` specifier).

For `NDB` tables, the default value for `COLUMN_FORMAT` is `DEFAULT`.

`COLUMN_FORMAT` currently has no effect on columns of tables using storage engines other than `NDB`. The `COLUMN_FORMAT` keyword is supported only in the build of `mysqld` that is supplied with MySQL Cluster; it is not recognized in any other version of MySQL, where attempting to use `COLUMN_FORMAT` causes a syntax error.

- For `NDB` tables, it is also possible to specify whether the column is stored on disk or in memory by using a `STORAGE` clause. `STORAGE DISK` causes the column to be stored on disk, and `STORAGE MEMORY` causes in-memory storage to be used. The `CREATE TABLE` statement used must still include a `TABLESPACE` clause:

```
mysql> CREATE TABLE t1 (
->   c1 INT STORAGE DISK,
->   c2 INT STORAGE MEMORY
-> ) ENGINE NDB;
ERROR 1005 (HY000): Can't create table 'c.t1' (errno: 140)

mysql> CREATE TABLE t1 (
->   c1 INT STORAGE DISK,
->   c2 INT STORAGE MEMORY
-> ) TABLESPACE ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.06 sec)
```

For `NDB` tables, `STORAGE DEFAULT` is equivalent to `STORAGE MEMORY`.

The `STORAGE` clause has no effect on tables using storage engines other than `NDB`. The `STORAGE` keyword is supported only in the build of `mysqld` that is supplied with MySQL Cluster; it is not recognized in any other version of MySQL, where any attempt to use the `STORAGE` keyword causes a syntax error.

- `KEY` is normally a synonym for `INDEX`. The key attribute `PRIMARY KEY` can also be specified as just `KEY` when given in a column definition. This was implemented for compatibility with other database systems.
- A `UNIQUE` index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a `UNIQUE` index permits multiple `NULL` values for columns that can contain `NULL`.
- A `PRIMARY KEY` is a unique index where all key columns must be defined as `NOT NULL`. If they are not explicitly declared as `NOT NULL`, MySQL declares them so implicitly (and silently). A table can have only one `PRIMARY KEY`. The name of a `PRIMARY KEY` is always `PRIMARY`, which thus cannot be used as the name for any other kind of index.

If you do not have a `PRIMARY KEY` and an application asks for the `PRIMARY KEY` in your tables, MySQL returns the first `UNIQUE` index that has no `NULL` columns as the `PRIMARY KEY`.

In `InnoDB` tables, keep the `PRIMARY KEY` short to minimize storage overhead for secondary indexes. Each secondary index entry contains a copy of the primary key columns for the corresponding row. (See [Section 14.3.11, “InnoDB Table and Index Structures”](#).)

- In the created table, a `PRIMARY KEY` is placed first, followed by all `UNIQUE` indexes, and then the nonunique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated `UNIQUE` keys.
- A `PRIMARY KEY` can be a multiple-column index. However, you cannot create a multiple-column index using the `PRIMARY KEY` key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate `PRIMARY KEY(index_col_name, ...)` clause.

- If a [PRIMARY KEY](#) or [UNIQUE](#) index consists of only one column that has an integer type, you can also refer to the column as `_rowid` in `SELECT` statements.
- In MySQL, the name of a [PRIMARY KEY](#) is `PRIMARY`. For other indexes, if you do not assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (`_2`, `_3`, ...) to make it unique. You can see index names for a table using `SHOW INDEX FROM tbl_name`. See [Section 13.7.5.23, “SHOW INDEX Syntax”](#).
- Some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is `USING type_name`.

Example:

```
CREATE TABLE lookup
  (id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

The preferred position for `USING` is after the index column list. It can be given before the column list, but support for use of the option in that position is deprecated and will be removed in a future MySQL release.

index_option values specify additional options for an index. `USING` is one such option. For details about permissible *index_option* values, see [Section 13.1.13, “CREATE INDEX Syntax”](#).

For more information about indexes, see [Section 8.3.1, “How MySQL Uses Indexes”](#).

- In MySQL 5.5, only the [MyISAM](#), [InnoDB](#), and [MEMORY](#) storage engines support indexes on columns that can have `NULL` values. In other cases, you must declare indexed columns as `NOT NULL` or an error results.
- For [CHAR](#), [VARCHAR](#), [BINARY](#), and [VARBINARY](#) columns, indexes can be created that use only the leading part of column values, using `col_name(length)` syntax to specify an index prefix length. [BLOB](#) and [TEXT](#) columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for [CHAR](#), [VARCHAR](#), and [TEXT](#) columns, and the first *length* bytes of each column value for [BINARY](#), [VARBINARY](#), and [BLOB](#) columns. Indexing only a prefix of column values like this can make the index file much smaller. See [Section 8.3.4, “Column Indexes”](#).

Only the [MyISAM](#) and [InnoDB](#) storage engines support indexing on [BLOB](#) and [TEXT](#) columns. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for [InnoDB](#) tables). Note that prefix limits are measured in bytes, whereas the prefix length in `CREATE TABLE` statements is interpreted as number of characters for nonbinary data types ([CHAR](#), [VARCHAR](#), [TEXT](#)). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

- An *index_col_name* specification can end with `ASC` or `DESC`. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.
- When you use `ORDER BY` or `GROUP BY` on a column in a `SELECT`, the server sorts values using only the initial number of bytes indicated by the `max_sort_length` [\[524\]](#) system variable.
- You can create special [FULLTEXT](#) indexes, which are used for full-text searches. Only the [MyISAM](#) storage engine supports [FULLTEXT](#) indexes. They can be created only from [CHAR](#), [VARCHAR](#), and [TEXT](#) columns. Indexing always happens over the entire column; column prefix indexing is not supported and

any prefix length is ignored if specified. See [Section 12.9, “Full-Text Search Functions”](#), for details of operation. A `WITH PARSER` clause can be specified as an `index_option` value to associate a parser plugin with the index if full-text indexing and searching operations need special handling. This clause is valid only for `FULLTEXT` indexes. See [Section 23.2, “The MySQL Plugin API”](#), for details on creating plugins.

- You can create `SPATIAL` indexes on spatial data types. Spatial types are supported only for `MyISAM` tables and indexed columns must be declared as `NOT NULL`. See [Section 12.17, “Spatial Extensions”](#).
- As of MySQL 5.5.3, index definitions can include an optional comment of up to 1024 characters.
- `InnoDB` tables support checking of foreign key constraints. See [Section 14.3, “The InnoDB Storage Engine”](#). Note that the `FOREIGN KEY` syntax in `InnoDB` is more restrictive than the syntax presented for the `CREATE TABLE` statement at the beginning of this section: The columns of the referenced table must always be explicitly named. `InnoDB` supports both `ON DELETE` and `ON UPDATE` actions on foreign keys. For the precise syntax, see [Section 14.3.5.4, “FOREIGN KEY Constraints”](#).

For other storage engines, MySQL Server parses and ignores the `FOREIGN KEY` and `REFERENCES` syntax in `CREATE TABLE` statements. The `CHECK` clause is parsed but ignored by all storage engines. See [Section 1.8.5.4, “Foreign Key Differences”](#).



Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential integrity constraint definitions. Use of an explicit `MATCH` clause will not have the specified effect, and also causes `ON DELETE` and `ON UPDATE` clauses to be ignored. For these reasons, specifying `MATCH` should be avoided.

The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key. `InnoDB` essentially implements the semantics defined by `MATCH SIMPLE`, which permit a foreign key to be all or partially `NULL`. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.

Additionally, MySQL and `InnoDB` require that the referenced columns be indexed for performance. However, the system does not enforce a requirement that the referenced columns be `UNIQUE` or be declared `NOT NULL`. The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only `UNIQUE` and `NOT NULL` keys.

Furthermore, `InnoDB` does not recognize or support “inline `REFERENCES` specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. `InnoDB` accepts `REFERENCES` clauses only when specified as part of a separate `FOREIGN KEY` specification. For other storage engines, MySQL Server parses and ignores foreign key specifications.



Note

Partitioned tables do not support foreign keys. See [Section 18.5, “Restrictions and Limitations on Partitioning”](#), for more information.

- There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table and depends on the factors discussed in [Section E.10.4, “Table Column-Count and Row-Size Limits”](#).

The `TABLESPACE` and `STORAGE` table options are employed only with `NDBCLUSTER` tables. The tablespace named `tablespace_name` must already have been created using `CREATE TABLESPACE`. `STORAGE` determines the type of storage used (disk or memory), and can be one of `DISK`, `MEMORY`, or `DEFAULT`.

`TABLESPACE ... STORAGE DISK` assigns a table to a MySQL Cluster Disk Data tablespace. See [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#), for more information.



Important

A `STORAGE` clause cannot be used in a `CREATE TABLE` statement without a `TABLESPACE` clause.

The `ENGINE` table option specifies the storage engine for the table.

The `ENGINE` table option takes the storage engine names shown in the following table.

Storage Engine	Description
InnoDB	Transaction-safe tables with row locking and foreign keys. The default storage engine for new tables. See Section 14.3, “The InnoDB Storage Engine” , and in particular Section 14.3.1, “InnoDB as the Default MySQL Storage Engine” if you have MySQL experience but are new to InnoDB.
MyISAM	The binary portable storage engine that is primarily used for read-only or read-mostly workloads. See Section 14.5, “The MyISAM Storage Engine” .
MEMORY	The data for this storage engine is stored only in memory. See Section 14.6, “The MEMORY Storage Engine” .
CSV	Tables that store rows in comma-separated values format. See Section 14.7, “The CSV Storage Engine” .
ARCHIVE	The archiving storage engine. See Section 14.8, “The ARCHIVE Storage Engine” .
EXAMPLE	An example engine. See Section 14.12, “The EXAMPLE Storage Engine” .
FEDERATED	Storage engine that accesses remote tables. See Section 14.11, “The FEDERATED Storage Engine” .
HEAP	This is a synonym for <code>MEMORY</code> .
MERGE	A collection of <code>MyISAM</code> tables used as one table. Also known as <code>MRG_MyISAM</code> . See Section 14.10, “The MERGE Storage Engine” .
NDBCLUSTER	Clustered, fault-tolerant, memory-based tables. Also known as <code>NDB</code> . See Chapter 17, MySQL Cluster NDB 7.2 .

If a storage engine is specified that is not available, MySQL uses the default engine instead. Normally, this is `MyISAM`. For example, if a table definition includes the `ENGINE=INNODB` option but the MySQL server does not support `INNODB` tables, the table is created as a `MyISAM` table. This makes it possible to have a replication setup where you have transactional tables on the master but tables created on the slave are nontransactional (to get more speed). In MySQL 5.5, a warning occurs if the storage engine specification is not honored.

Engine substitution can be controlled by the setting of the `NO_ENGINE_SUBSTITUTION` [614] SQL mode, as described in [Section 5.1.7, “Server SQL Modes”](#).

**Note**

The older `TYPE` option was synonymous with `ENGINE`. `TYPE` was deprecated in MySQL 4.0 and removed in MySQL 5.5. *When upgrading to MySQL 5.5 or later, you must convert existing applications that rely on `TYPE` to use `ENGINE` instead.*

The other table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them. These options apply to all storage engines unless otherwise indicated. Options that do not apply to a given storage engine may be accepted and remembered as part of the table definition. Such options then apply if you later use `ALTER TABLE` to convert the table to use a different storage engine.

- `AUTO_INCREMENT`

The initial `AUTO_INCREMENT` value for the table. In MySQL 5.5, this works for `MyISAM`, `MEMORY`, `InnoDB`, and `ARCHIVE` tables. To set the first auto-increment value for engines that do not support the `AUTO_INCREMENT` table option, insert a “dummy” row with a value one less than the desired value after creating the table, and then delete the dummy row.

For engines that support the `AUTO_INCREMENT` table option in `CREATE TABLE` statements, you can also use `ALTER TABLE tbl_name AUTO_INCREMENT = N` to reset the `AUTO_INCREMENT` value. The value cannot be set lower than the maximum value currently in the column.

- `AVG_ROW_LENGTH`

An approximation of the average row length for your table. You need to set this only for large tables with variable-size rows.

When you create a `MyISAM` table, MySQL uses the product of the `MAX_ROWS` and `AVG_ROW_LENGTH` options to decide how big the resulting table is. If you don't specify either option, the maximum size for `MyISAM` data and index files is 256TB by default. (If your operating system does not support files that large, table sizes are constrained by the file size limit.) If you want to keep down the pointer sizes to make the index smaller and faster and you don't really need big files, you can decrease the default pointer size by setting the `myisam_data_pointer_size` [527] system variable. (See [Section 5.1.4, “Server System Variables”](#).) If you want all your tables to be able to grow above the default limit and are willing to have your tables slightly slower and larger than necessary, you can increase the default pointer size by setting this variable. Setting the value to 7 permits table sizes up to 65,536TB.

- `[DEFAULT] CHARACTER SET`

Specify a default character set for the table. `CHARSET` is a synonym for `CHARACTER SET`. If the character set name is `DEFAULT`, the database character set is used.

- `CHECKSUM`

Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update, but also makes it easier to find corrupted tables. The `CHECKSUM TABLE` statement reports the checksum. (`MyISAM` only.)

- `[DEFAULT] COLLATE`

Specify a default collation for the table.

- `COMMENT`

A comment for the table, up to 2048 characters long (60 characters before MySQL 5.5.3).

- `CONNECTION`

The connection string for a `FEDERATED` table.



Note

Older versions of MySQL used a `COMMENT` option for the connection string.

- `DATA DIRECTORY`, `INDEX DIRECTORY`

By using `DATA DIRECTORY='directory'` or `INDEX DIRECTORY='directory'` you can specify where the `MyISAM` storage engine should put a table's data file and index file. The directory must be the full path name to the directory, not a relative path.



Important

Table-level `DATA DIRECTORY` and `INDEX DIRECTORY` options are ignored for partitioned tables. (Bug #32091)

These options work only when you are not using the `--skip-symbolic-links` [462] option. Your operating system must also have a working, thread-safe `realpath()` call. See Section 8.11.3.1.2, “Using Symbolic Links for `MyISAM` Tables on Unix”, for more complete information.

If a `MyISAM` table is created with no `DATA DIRECTORY` option, the `.MYD` file is created in the database directory. By default, if `MyISAM` finds an existing `.MYD` file in this case, it overwrites it. The same applies to `.MYI` files for tables created with no `INDEX DIRECTORY` option. To suppress this behavior, start the server with the `--keep_files_on_create` [507] option, in which case `MyISAM` will not overwrite existing files and returns an error instead.

If a `MyISAM` table is created with a `DATA DIRECTORY` or `INDEX DIRECTORY` option and an existing `.MYD` or `.MYI` file is found, `MyISAM` always returns an error. It will not overwrite a file in the specified directory.



Important

You cannot use path names that contain the MySQL data directory with `DATA DIRECTORY` or `INDEX DIRECTORY`. This includes partitioned tables and individual table partitions. (See Bug #32167.)

- `DELAY_KEY_WRITE`

Set this to 1 if you want to delay key updates for the table until the table is closed. See the description of the `delay_key_write` [494] system variable in Section 5.1.4, “Server System Variables”. (`MyISAM` only.)

- `INSERT_METHOD`

If you want to insert data into a `MERGE` table, you must specify with `INSERT_METHOD` the table into which the row should be inserted. `INSERT_METHOD` is an option useful for `MERGE` tables only. Use a value of `FIRST` or `LAST` to have inserts go to the first or last table, or a value of `NO` to prevent inserts. See Section 14.10, “The `MERGE` Storage Engine”.

- `KEY_BLOCK_SIZE`

For `compressed InnoDB` tables, optionally specifies the size in bytes to use for `pages`. The value is treated as a hint; a different size could be used if necessary. A value of 0 represents that the default compressed page size. See Section 14.4.3, “InnoDB Data Compression” for usage details.

Individual index definitions can specify a `KEY_BLOCK_SIZE` value of their own to override the table value.



Note

Oracle recommends enabling `innodb_strict_mode` [1572] when using the `KEY_BLOCK_SIZE` clause for InnoDB tables. See Section 14.4.8.4, “InnoDB Strict Mode” for details.

- `MAX_ROWS`

The maximum number of rows you plan to store in the table. This is not a hard limit, but rather a hint to the storage engine that the table must be able to store at least this many rows.

The NDB storage engine treats this value as a maximum. If you plan to create very large MySQL Cluster tables (containing millions of rows), you should use this option to insure that NDB allocates sufficient number of index slots in the hash table used for storing hashes of the table's primary keys by setting `MAX_ROWS = 2 * rows`, where `rows` is the number of rows that you expect to insert into the table.

The maximum `MAX_ROWS` value is 4294967295; larger values are truncated to this limit.

- `MIN_ROWS`

The minimum number of rows you plan to store in the table. The MEMORY storage engine uses this option as a hint about memory use.

- `PACK_KEYS`

`PACK_KEYS` takes effect only with MyISAM tables. Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of keys. Setting it to `DEFAULT` tells the storage engine to pack only long `CHAR`, `VARCHAR`, `BINARY`, or `VARBINARY` columns.

If you do not use `PACK_KEYS`, the default is to pack strings, but not numbers. If you use `PACK_KEYS=1`, numbers are packed as well.

When packing binary number keys, MySQL uses prefix compression:

- Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.
- The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

This means that if you have many equal keys on two consecutive rows, all following “same” keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes `storage_size_for_key + pointer_size` (where the pointer size is usually 4). Conversely, you get a significant benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key is not a key that can have `NULL` values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is `NULL`.)

- `PASSWORD`

This option is unused. If you have a need to scramble your `.frm` files and make them unusable to any other MySQL server, please contact our sales department.

- `ROW_FORMAT`

Defines the physical format in which the rows are stored. The choices differ depending on the storage engine used for the table.

For `InnoDB` tables:

- Rows are stored in compact format (`ROW_FORMAT=COMPACT`) by default.
- The noncompact format used in older versions of MySQL can still be requested by specifying `ROW_FORMAT=REDUNDANT`.
- To enable compression for `InnoDB` tables, specify `ROW_FORMAT=COMPRESSED` and follow the procedures in [Section 14.4.3, “InnoDB Data Compression”](#).
- For more efficient `InnoDB` storage of data types, especially `BLOB` types, specify `ROW_FORMAT=DYNAMIC` and follow the procedures in [Section 14.4.5.3, “Barracuda File Format: DYNAMIC and COMPRESSED Row Formats”](#). Both the `COMPRESSED` and `DYNAMIC` row formats require creating the table with the configuration settings `innodb_file_per_table=1` [1555] and `innodb_file_format=barracuda` [1554].
- When you specify a non-default `ROW_FORMAT` clause, consider also enabling the `innodb_strict_mode` [1572] configuration option. See [Section 14.4.8.4, “InnoDB Strict Mode”](#) for details.

For `MyISAM` tables, the option value can be `FIXED` or `DYNAMIC` for static or variable-length row format. `myisampack` sets the type to `COMPRESSED`. See [Section 14.5.3, “MyISAM Table Storage Formats”](#).



Note

When executing a `CREATE TABLE` statement, if you specify a row format which is not supported by the storage engine that is used for the table, the table is created using that storage engine's default row format. The information reported in this column in response to `SHOW TABLE STATUS` is the actual row format used. This may differ from the value in the `Create_options` column because the original `CREATE TABLE` definition is retained during creation.

- `UNION`

`UNION` is used when you want to access a collection of identical `MyISAM` tables as one. This works only with `MERGE` tables. See [Section 14.10, “The MERGE Storage Engine”](#).

You must have `SELECT` [697], `UPDATE` [697], and `DELETE` [696] privileges for the tables you map to a `MERGE` table.



Note

Formerly, all tables used had to be in the same database as the `MERGE` table itself. This restriction no longer applies.

`partition_options` can be used to control partitioning of the table created with `CREATE TABLE`.



Important

Not all options shown in the syntax for `partition_options` at the beginning of this section are available for all partitioning types. Please see the listings for the following individual types for information specific to each type, and see [Chapter 18](#),

Partitioning, for more complete information about the workings of and uses for partitioning in MySQL, as well as additional examples of table creation and other statements relating to MySQL partitioning.

If used, a *partition_options* clause begins with `PARTITION BY`. This clause contains the function that is used to determine the partition; the function returns an integer value ranging from 1 to *num*, where *num* is the number of partitions. (The maximum number of user-defined partitions which a table may contain is 1024; the number of subpartitions—discussed later in this section—is included in this maximum.) The choices that are available for this function in MySQL 5.5 are shown in the following list:

- `HASH(expr)`: Hashes one or more columns to create a key for placing and locating rows. *expr* is an expression using one or more table columns. This can be any valid MySQL expression (including MySQL functions) that yields a single integer value. For example, these are both valid `CREATE TABLE` statements using `PARTITION BY HASH`:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5))
  PARTITION BY HASH(col1);

CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATETIME)
  PARTITION BY HASH ( YEAR(col3) );
```

You may not use either `VALUES LESS THAN` or `VALUES IN` clauses with `PARTITION BY HASH`.

`PARTITION BY HASH` uses the remainder of *expr* divided by the number of partitions (that is, the modulus). For examples and additional information, see [Section 18.2.4, “HASH Partitioning”](#).

The `LINEAR` keyword entails a somewhat different algorithm. In this case, the number of the partition in which a row is stored is calculated as the result of one or more logical `AND` [1094] operations. For discussion and examples of linear hashing, see [Section 18.2.4.1, “LINEAR HASH Partitioning”](#).

- `KEY(column_list)`: This is similar to `HASH`, except that MySQL supplies the hashing function so as to guarantee an even data distribution. The *column_list* argument is simply a list of 1 or more table columns (maximum: 16). This example shows a simple table partitioned by key, with 4 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
  PARTITION BY KEY(col3)
  PARTITIONS 4;
```

For tables that are partitioned by key, you can employ linear partitioning by using the `LINEAR` keyword. This has the same effect as with tables that are partitioned by `HASH`. That is, the partition number is found using the `&` [1182] operator rather than the modulus (see [Section 18.2.4.1, “LINEAR HASH Partitioning”](#), and [Section 18.2.5, “KEY Partitioning”](#), for details). This example uses linear partitioning by key to distribute data between 5 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
  PARTITION BY LINEAR KEY(col3)
  PARTITIONS 5;
```

You may not use either `VALUES LESS THAN` or `VALUES IN` clauses with `PARTITION BY KEY`.

- `RANGE(expr)`: In this case, *expr* shows a range of values using a set of `VALUES LESS THAN` operators. When using range partitioning, you must define at least one partition using `VALUES LESS THAN`. You cannot use `VALUES IN` with range partitioning.



Note

For tables partitioned by `RANGE`, `VALUES LESS THAN` must be used with either an integer literal value or an expression that evaluates to a single integer value.

In MySQL 5.5, you can overcome this limitation in a table that is defined using `PARTITION BY RANGE COLUMNS`, as described later in this section.

Suppose that you have a table that you wish to partition on a column containing year values, according to the following scheme.

Partition Number:	Years Range:
0	1990 and earlier
1	1991 to 1994
2	1995 to 1998
3	1999 to 2002
4	2003 to 2005
5	2006 and later

A table implementing such a partitioning scheme can be realized by the `CREATE TABLE` statement shown here:

```
CREATE TABLE t1 (
  year_col INT,
  some_data INT
)
PARTITION BY RANGE (year_col) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (1999),
  PARTITION p3 VALUES LESS THAN (2002),
  PARTITION p4 VALUES LESS THAN (2006),
  PARTITION p5 VALUES LESS THAN MAXVALUE
);
```

`PARTITION ... VALUES LESS THAN ...` statements work in a consecutive fashion. `VALUES LESS THAN MAXVALUE` works to specify “leftover” values that are greater than the maximum value otherwise specified.

Note that `VALUES LESS THAN` clauses work sequentially in a manner similar to that of the `case` portions of a `switch ... case` block (as found in many programming languages such as C, Java, and PHP). That is, the clauses must be arranged in such a way that the upper limit specified in each successive `VALUES LESS THAN` is greater than that of the previous one, with the one referencing `MAXVALUE` coming last of all in the list.

- `RANGE COLUMNS(column_list)`: This variant on `RANGE` was introduced in MySQL 5.5.0 to facilitate partition pruning for queries using range conditions on multiple columns (that is, having conditions such as `WHERE a = 1 AND b < 10` or `WHERE a = 1 AND b = 10 AND c < 10`). It enables you to specify value ranges in multiple columns by using a list of columns in the `COLUMNS` clause and a set of column values in each `PARTITION ... VALUES LESS THAN (value_list)` partition definition clause. (In the simplest case, this set consists of a single column.) The maximum number of columns that can be referenced in the `column_list` and `value_list` is 16.

The `column_list` used in the `COLUMNS` clause may contain only names of columns; each column in the list must be one of the following MySQL data types: the integer types; the string types; and time or date column types. Columns using `BLOB`, `TEXT`, `SET`, `ENUM`, `BIT`, or spatial data types are not permitted; columns that use floating-point number types are also not permitted. You also may not use functions or arithmetic expressions in the `COLUMNS` clause.

The `VALUES LESS THAN` clause used in a partition definition must specify a literal value for each column that appears in the `COLUMNS()` clause; that is, the list of values used for each `VALUES LESS THAN` clause must contain the same number of values as there are columns listed in the `COLUMNS` clause. An attempt to use more or fewer values in a `VALUES LESS THAN` clause than there are in the `COLUMNS` clause causes the statement to fail with the error `Inconsistency in usage of column lists for partitioning...` You cannot use `NULL` for any value appearing in `VALUES LESS THAN`. It is possible to use `MAXVALUE` more than once for a given column other than the first, as shown in this example:

```
CREATE TABLE rc (
  a INT NOT NULL,
  b INT NOT NULL
)
PARTITION BY RANGE COLUMNS(a,b) (
  PARTITION p0 VALUES LESS THAN (10,5),
  PARTITION p1 VALUES LESS THAN (20,10),
  PARTITION p2 VALUES LESS THAN (MAXVALUE,15),
  PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)
);
```

Each value used in a `VALUES LESS THAN` value list must match the type of the corresponding column exactly; no conversion is made. For example, you cannot use the string `'1'` for a value that matches a column that uses an integer type (you must use the numeral `1` instead), nor can you use the numeral `1` for a value that matches a column that uses a string type (in such a case, you must use a quoted string: `'1'`).

For more information, see [Section 18.2.1, “RANGE Partitioning”](#), and [Section 18.4, “Partition Pruning”](#).

- `LIST(expr)`: This is useful when assigning partitions based on a table column with a restricted set of possible values, such as a state or country code. In such a case, all rows pertaining to a certain state or country can be assigned to a single partition, or a partition can be reserved for a certain set of states or countries. It is similar to `RANGE`, except that only `VALUES IN` may be used to specify permissible values for each partition.

`VALUES IN` is used with a list of values to be matched. For instance, you could create a partitioning scheme such as the following:

```
CREATE TABLE client_firms (
  id INT,
  name VARCHAR(35)
)
PARTITION BY LIST (id) (
  PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
  PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
  PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
  PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
);
```

When using list partitioning, you must define at least one partition using `VALUES IN`. You cannot use `VALUES LESS THAN` with `PARTITION BY LIST`.



Note

For tables partitioned by `LIST`, the value list used with `VALUES IN` must consist of integer values only. In MySQL 5.5, you can overcome this limitation using partitioning by `LIST COLUMNS`, which is described later in this section.

- `LIST COLUMNS(column_list)`: This variant on `LIST` was introduced in MySQL 5.5.0 to facilitate partition pruning for queries using comparison conditions on multiple columns (that is, having conditions

such as `WHERE a = 5 AND b = 5` or `WHERE a = 1 AND b = 10 AND c = 5`). It enables you to specify values in multiple columns by using a list of columns in the `COLUMNS` clause and a set of column values in each `PARTITION ... VALUES IN (value_list)` partition definition clause.

The rules governing regarding data types for the column list used in `LIST COLUMNS(column_list)` and the value list used in `VALUES IN(value_list)` are the same as those for the column list used in `RANGE COLUMNS(column_list)` and the value list used in `VALUES LESS THAN(value_list)`, respectively, except that in the `VALUES IN` clause, `MAXVALUE` is not permitted, and you may use `NULL`.

There is one important difference between the list of values used for `VALUES IN` with `PARTITION BY LIST COLUMNS` as opposed to when it is used with `PARTITION BY LIST`. When used with `PARTITION BY LIST COLUMNS`, each element in the `VALUES IN` clause must be a set of column values; the number of values in each set must be the same as the number of columns used in the `COLUMNS` clause, and the data types of these values must match those of the columns (and occur in the same order). In the simplest case, the set consists of a single column. The maximum number of columns that can be used in the `column_list` and in the elements making up the `value_list` is 16.

The table defined by the following `CREATE TABLE` statement provides an example of a table using `LIST COLUMNS` partitioning:

```
CREATE TABLE lc (
  a INT NULL,
  b INT NULL
)
PARTITION BY LIST COLUMNS(a,b) (
  PARTITION p0 VALUES IN( (0,0), (NULL,NULL) ),
  PARTITION p1 VALUES IN( (0,1), (0,2), (0,3), (1,1), (1,2) ),
  PARTITION p2 VALUES IN( (1,0), (2,0), (2,1), (3,0), (3,1) ),
  PARTITION p3 VALUES IN( (1,3), (2,2), (2,3), (3,2), (3,3) )
);
```

- The number of partitions may optionally be specified with a `PARTITIONS num` clause, where `num` is the number of partitions. If both this clause *and* any `PARTITION` clauses are used, `num` must be equal to the total number of any partitions that are declared using `PARTITION` clauses.



Note

Whether or not you use a `PARTITIONS` clause in creating a table that is partitioned by `RANGE` or `LIST`, you must still include at least one `PARTITION VALUES` clause in the table definition (see below).

- A partition may optionally be divided into a number of subpartitions. This can be indicated by using the optional `SUBPARTITION BY` clause. Subpartitioning may be done by `HASH` or `KEY`. Either of these may be `LINEAR`. These work in the same way as previously described for the equivalent partitioning types. (It is not possible to subpartition by `LIST` or `RANGE`.)

The number of subpartitions can be indicated using the `SUBPARTITIONS` keyword followed by an integer value.

- Rigorous checking of the value used in `PARTITIONS` or `SUBPARTITIONS` clauses is applied and this value must adhere to the following rules:
 - The value must be a positive, nonzero integer.
 - No leading zeros are permitted.
 - The value must be an integer literal, and cannot not be an expression. For example, `PARTITIONS 0.2E+01` is not permitted, even though `0.2E+01` evaluates to `2`. (Bug #15890)

**Note**

The expression (*expr*) used in a `PARTITION BY` clause cannot refer to any columns not in the table being created; such references are specifically not permitted and cause the statement to fail with an error. (Bug #29444)

Each partition may be individually defined using a *partition_definition* clause. The individual parts making up this clause are as follows:

- `PARTITION partition_name`: This specifies a logical name for the partition.
- A `VALUES` clause: For range partitioning, each partition must include a `VALUES LESS THAN` clause; for list partitioning, you must specify a `VALUES IN` clause for each partition. This is used to determine which rows are to be stored in this partition. See the discussions of partitioning types in [Chapter 18, Partitioning](#), for syntax examples.
- An optional `COMMENT` clause may be used to specify a string that describes the partition. Example:

```
COMMENT = 'Data for the years previous to 1999'
```

- `DATA DIRECTORY` and `INDEX DIRECTORY` may be used to indicate the directory where, respectively, the data and indexes for this partition are to be stored. Both the *data_dir* and the *index_dir* must be absolute system path names. Example:

```
CREATE TABLE th (id INT, name VARCHAR(30), adate DATE)
PARTITION BY LIST(YEAR(adata))
(
  PARTITION p1999 VALUES IN (1995, 1999, 2003)
    DATA DIRECTORY = '/var/appdata/95/data'
    INDEX DIRECTORY = '/var/appdata/95/idx',
  PARTITION p2000 VALUES IN (1996, 2000, 2004)
    DATA DIRECTORY = '/var/appdata/96/data'
    INDEX DIRECTORY = '/var/appdata/96/idx',
  PARTITION p2001 VALUES IN (1997, 2001, 2005)
    DATA DIRECTORY = '/var/appdata/97/data'
    INDEX DIRECTORY = '/var/appdata/97/idx',
  PARTITION p2002 VALUES IN (1998, 2002, 2006)
    DATA DIRECTORY = '/var/appdata/98/data'
    INDEX DIRECTORY = '/var/appdata/98/idx'
);
```

`DATA DIRECTORY` and `INDEX DIRECTORY` behave in the same way as in the `CREATE TABLE` statement's *table_option* clause as used for `MyISAM` tables.

One data directory and one index directory may be specified per partition. If left unspecified, the data and indexes are stored by default in the table's database directory.

On Windows, the `DATA DIRECTORY` and `INDEX DIRECTORY` options are not supported for individual partitions or subpartitions. These options are ignored on Windows, except that a warning is generated. (Bug #30459)

**Note**

The `DATA DIRECTORY` and `INDEX DIRECTORY` options are ignored for creating partitioned tables if `NO_DIR_IN_CREATE` [614] is in effect. (Bug #24633)

- `MAX_ROWS` and `MIN_ROWS` may be used to specify, respectively, the maximum and minimum number of rows to be stored in the partition. The values for *max_number_of_rows* and *min_number_of_rows* must be positive integers. As with the table-level options with the same names, these act only as “suggestions” to the server and are not hard limits.

- The optional `TABLESPACE` clause may be used to designate a tablespace for the partition. Used for MySQL Cluster only.
- The partitioning handler accepts a `[STORAGE] ENGINE` option for both `PARTITION` and `SUBPARTITION`. Currently, the only way in which this can be used is to set all partitions or all subpartitions to the same storage engine, and an attempt to set different storage engines for partitions or subpartitions in the same table will give rise to the error `ERROR 1469 (HY000): The mix of handlers in the partitions is not permitted in this version of MySQL`. We expect to lift this restriction on partitioning in a future MySQL release.
- The `NODEGROUP` option can be used to make this partition act as part of the node group identified by `node_group_id`. This option is applicable only to MySQL Cluster.
- The partition definition may optionally contain one or more `subpartition_definition` clauses. Each of these consists at a minimum of the `SUBPARTITION name`, where `name` is an identifier for the subpartition. Except for the replacement of the `PARTITION` keyword with `SUBPARTITION`, the syntax for a subpartition definition is identical to that for a partition definition.

Subpartitioning must be done by `HASH` or `KEY`, and can be done only on `RANGE` or `LIST` partitions. See [Section 18.2.6, “Subpartitioning”](#).

Partitions can be modified, merged, added to tables, and dropped from tables. For basic information about the MySQL statements to accomplish these tasks, see [Section 13.1.7, “ALTER TABLE Syntax”](#). For more detailed descriptions and examples, see [Section 18.3, “Partition Management”](#).



Important

The original `CREATE TABLE` statement, including all specifications and table options are stored by MySQL when the table is created. The information is retained so that if you change storage engines, collations or other settings using an `ALTER TABLE` statement, the original table options specified are retained. This enables you to change between `InnoDB` and `MyISAM` table types even though the row formats supported by the two engines are different.

Because the text of the original statement is retained, but due to the way that certain values and options may be silently reconfigured (such as the `ROW_FORMAT`), the active table definition (accessible through `DESCRIBE` or with `SHOW TABLE STATUS`) and the table creation string (accessible through `SHOW CREATE TABLE`) will report different values.

You can create one table from another by adding a `SELECT` statement at the end of the `CREATE TABLE` statement:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

For more information, see [Section 13.1.17.1, “CREATE TABLE ... SELECT Syntax”](#).

Use `LIKE` to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

The copy is created using the same version of the table storage format as the original table. The `SELECT [697]` privilege is required on the original table.

`LIKE` works only for base tables, not for views.



Important

Beginning with MySQL 5.5.3, you cannot execute `CREATE TABLE` or `CREATE TABLE ... LIKE` while a `LOCK TABLES` statement is in effect.

Also as of MySQL 5.5.3, `CREATE TABLE ... LIKE` makes the same checks as `CREATE TABLE` and does not just copy the `.frm` file. This means that if the current SQL mode is different from the mode in effect when the original table was created, the table definition might be considered invalid for the new mode and the statement will fail.

`CREATE TABLE ... LIKE` does not preserve any `DATA DIRECTORY` or `INDEX DIRECTORY` table options that were specified for the original table, or any foreign key definitions.

If the original table is a `TEMPORARY` table, `CREATE TABLE ... LIKE` does not preserve `TEMPORARY`. To create a `TEMPORARY` destination table, use `CREATE TEMPORARY TABLE ... LIKE`.

13.1.17.1. CREATE TABLE ... SELECT Syntax

You can create one table from another by adding a `SELECT` statement at the end of the `CREATE TABLE` statement:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the `SELECT`. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
->     PRIMARY KEY (a), KEY(b))
->     ENGINE=MyISAM SELECT b,c FROM test2;
```

This creates a `MyISAM` table with three columns, `a`, `b`, and `c`. The `ENGINE` option is part of the `CREATE TABLE` statement, and should not be used following the `SELECT`; this would result in a syntax error. The same is true for other `CREATE TABLE` options such as `CHARSET`.

Notice that the columns from the `SELECT` statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+----+
| n  |
+----+
| 1  |
+----+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM bar;
+-----+-----+
| m      | n  |
+-----+-----+
| NULL  | 1  |
+-----+-----+
1 row in set (0.00 sec)
```

For each row in table `foo`, a row is inserted in `bar` with the values from `foo` and default values for the new columns.

In a table resulting from `CREATE TABLE ... SELECT`, columns named only in the `CREATE TABLE` part come first. Columns named in both parts or only in the `SELECT` part come after that. The data type of `SELECT` columns can be overridden by also specifying the column in the `CREATE TABLE` part.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

You can precede the `SELECT` by `IGNORE` or `REPLACE` to indicate how to handle rows that duplicate unique key values. With `IGNORE`, new rows that duplicate an existing row on a unique key value are discarded. With `REPLACE`, new rows replace rows that have the same unique key value. If neither `IGNORE` nor `REPLACE` is specified, duplicate unique key values result in an error.

Because the ordering of the rows in the underlying `SELECT` statements cannot always be determined, `CREATE TABLE ... IGNORE SELECT` and `CREATE TABLE ... REPLACE SELECT` statements in MySQL 5.5.18 and later are flagged as unsafe for statement-based replication. With this change, such statements produce a warning in the log when using statement-based mode and are logged using the row-based format when using `MIXED` mode. See also [Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”](#).

`CREATE TABLE ... SELECT` does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the `SELECT` statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Some conversion of data types might occur. For example, the `AUTO_INCREMENT` attribute is not preserved, and `VARCHAR` columns can become `CHAR` columns. Retrained attributes are `NULL` (or `NOT NULL`) and, for those columns that have them, `CHARACTER SET`, `COLLATION`, `COMMENT`, and the `DEFAULT` clause.

When creating a table with `CREATE TABLE ... SELECT`, make sure to alias any function calls or expressions in the query. If you do not, the `CREATE` statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
  SELECT artist.name, COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work ON artist.id = work.artist_id
  GROUP BY artist.id;
```

You can also explicitly specify the data type for a generated column:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

For `CREATE TABLE ... SELECT`, if `IF NOT EXISTS` is given and the destination table already exists, the result is version dependent. Before MySQL 5.5.6, MySQL handles the statement as follows:

- The table definition given in the `CREATE TABLE` part is ignored. No error occurs, even if the definition does not match that of the existing table. MySQL attempts to insert the rows from the `SELECT` part anyway.
- If there is a mismatch between the number of columns in the table and the number of columns produced by the `SELECT` part, the selected values are assigned to the rightmost columns. For example, if the table contains n columns and the `SELECT` produces m columns, where $m < n$, the selected values are assigned to the m rightmost columns in the table. Each of the initial $n - m$ columns is assigned its default value, either that specified explicitly in the column definition or the implicit column data type default if the definition contains no default. If the `SELECT` part produces too many columns ($m > n$), an error occurs.
- If strict SQL mode is enabled and any of these initial columns do not have an explicit default value, the statement fails with an error.

The following example illustrates `IF NOT EXISTS` handling:

```
mysql> CREATE TABLE t1 (i1 INT DEFAULT 0, i2 INT, i3 INT, i4 INT);
```

```
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE IF NOT EXISTS t1 (c1 CHAR(10)) SELECT 1, 2;
Query OK, 1 row affected, 1 warning (0.01 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+-----+-----+-----+-----+
| i1   | i2   | i3   | i4   |
+-----+-----+-----+-----+
| 0    | NULL | 1    | 2    |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

As of MySQL 5.5.6, handling of `CREATE TABLE IF NOT EXISTS ... SELECT` statements was changed for the case that the destination table already exists. This change also involves a change in MySQL 5.1 beginning with 5.1.51.

- Previously, for `CREATE TABLE IF NOT EXISTS ... SELECT`, MySQL produced a warning that the table exists, but inserted the rows and wrote the statement to the binary log anyway. By contrast, `CREATE TABLE ... SELECT` (without `IF NOT EXISTS`) failed with an error, but MySQL inserted no rows and did not write the statement to the binary log.
- MySQL now handles both statements the same way when the destination table exists, in that neither statement inserts rows or is written to the binary log. The difference between them is that MySQL produces a warning when `IF NOT EXISTS` is present and an error when it is not.

This change means that, for the preceding example, the `CREATE TABLE IF NOT EXISTS ... SELECT` statement inserts nothing into the destination table as of MySQL 5.5.6.

This change in handling of `IF NOT EXISTS` results in an incompatibility for statement-based replication from a MySQL 5.1 master with the original behavior and a MySQL 5.5 slave with the new behavior. Suppose that `CREATE TABLE IF NOT EXISTS ... SELECT` is executed on the master and the destination table exists. The result is that rows are inserted on the master but not on the slave. (Row-based replication does not have this problem.)

To address this issue, statement-based binary logging for `CREATE TABLE IF NOT EXISTS ... SELECT` is changed in MySQL 5.1 as of 5.1.51:

- If the destination table does not exist, there is no change: The statement is logged as is.
- If the destination table does exist, the statement is logged as the equivalent pair of `CREATE TABLE IF NOT EXISTS` and `INSERT ... SELECT` statements. (If the `SELECT` in the original statement is preceded by `IGNORE` or `REPLACE`, the `INSERT` becomes `INSERT IGNORE` or `REPLACE`, respectively.)

This change provides forward compatibility for statement-based replication from MySQL 5.1 to 5.5 because when the destination table exists, the rows will be inserted on both the master and slave. To take advantage of this compatibility measure, the 5.1 server must be at least 5.1.51 and the 5.5 server must be at least 5.5.6.

To upgrade an existing 5.1-to-5.5 replication scenario, upgrade the master first to 5.1.51 or higher. Note that this differs from the usual replication upgrade advice of upgrading the slave first.

A workaround for applications that wish to achieve the original effect (rows inserted regardless of whether the destination table exists) is to use `CREATE TABLE IF NOT EXISTS` and `INSERT ... SELECT` statements rather than `CREATE TABLE IF NOT EXISTS ... SELECT` statements.

Along with the change just described, the following related change was made: Previously, if an existing view was named as the destination table for `CREATE TABLE IF NOT EXISTS ... SELECT`, rows were

inserted into the underlying base table and the statement was written to the binary log. As of MySQL 5.1.51 and 5.5.6, nothing is inserted or logged.

To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts during `CREATE TABLE ... SELECT`.

13.1.17.2. Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a `CREATE TABLE` or `ALTER TABLE` statement. These might be changes to a data type, to attributes associated with a data type, or to an index specification.

All changes are subject to the internal row-size limit of 65,535 bytes, which may cause some attempts at data type changes to fail. See [Section E.10.4, “Table Column-Count and Row-Size Limits”](#).

- `TIMESTAMP` display sizes are discarded.

Also note that `TIMESTAMP` columns are `NOT NULL` by default.

- Columns that are part of a `PRIMARY KEY` are made `NOT NULL` even if not declared that way.
- Trailing spaces are automatically deleted from `ENUM` and `SET` member values when the table is created.
- MySQL maps certain data types used by other SQL database vendors to MySQL types. See [Section 11.8, “Using Data Types from Other Database Engines”](#).
- If you include a `USING` clause to specify an index type that is not permitted for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.
- If strict SQL mode is not enabled, a `VARCHAR` column with a length specification greater than 65535 is converted to `TEXT`, and a `VARBINARY` column with a length specification greater than 65535 is converted to `BLOB`. Otherwise, an error occurs in either of these cases.
- Specifying the `CHARACTER SET binary` attribute for a character data type causes the column to be created as the corresponding binary data type: `CHAR` becomes `BINARY`, `VARCHAR` becomes `VARBINARY`, and `TEXT` becomes `BLOB`. For the `ENUM` and `SET` data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

To see whether MySQL used a data type other than the one you specified, issue a `DESCRIBE` or `SHOW CREATE TABLE` statement after creating or altering the table.

Certain other data type changes can occur if you compress a table using `mysampack`. See [Section 14.5.3.3, “Compressed Table Characteristics”](#).

13.1.18. CREATE TABLESPACE Syntax

```
CREATE TABLESPACE tablespace_name
  ADD DATAFILE 'file_name'
  USE LOGFILE GROUP logfile_group
  [EXTENT_SIZE [=] extent_size]
  [INITIAL_SIZE [=] initial_size]
  [AUTOEXTEND_SIZE [=] autoextend_size]
  [MAX_SIZE [=] max_size]
  [NODEGROUP [=] nodegroup_id]
  [WAIT]
  [COMMENT [=] comment_text]
  ENGINE [=] engine_name
```

This statement is used to create a tablespace, which can contain one or more data files, providing storage space for tables. One data file is created and added to the tablespace using this statement. Additional data files may be added to the tablespace by using the [ALTER TABLESPACE](#) statement (see [Section 13.1.8, “ALTER TABLESPACE Syntax”](#)). For rules covering the naming of tablespaces, see [Section 9.2, “Schema Object Names”](#).



Note

All MySQL Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group with the same name, or a tablespace and a data file with the same name.

A log file group of one or more [UNDO](#) log files must be assigned to the tablespace to be created with the [USE LOGFILE GROUP](#) clause. *logfile_group* must be an existing log file group created with [CREATE LOGFILE GROUP](#) (see [Section 13.1.14, “CREATE LOGFILE GROUP Syntax”](#)). Multiple tablespaces may use the same log file group for [UNDO](#) logging.

The [EXTENT_SIZE](#) sets the size, in bytes, of the extents used by any files belonging to the tablespace. The default value is 1M. The minimum size is 32K, and theoretical maximum is 2G, although the practical maximum size depends on a number of factors. In most cases, changing the extent size does not have any measurable effect on performance, and the default value is recommended for all but the most unusual situations.

An *extent* is a unit of disk space allocation. One extent is filled with as much data as that extent can contain before another extent is used. In theory, up to 65,535 (64K) extents may be used per data file; however, the recommended maximum is 32,768 (32K). The recommended maximum size for a single data file is 32G—that is, 32K extents × 1 MB per extent. In addition, once an extent is allocated to a given partition, it cannot be used to store data from a different partition; an extent cannot store data from more than one partition. This means, for example, that a tablespace having a single datafile whose [INITIAL_SIZE](#) is 256 MB and whose [EXTENT_SIZE](#) is 128M has just two extents, and so can be used to store data from at most two different disk data table partitions.

You can see how many extents remain free in a given data file by querying the [INFORMATION_SCHEMA.FILES](#) table, and so derive an estimate for how much space remains free in the file. For further discussion and examples, see [Section 20.29.1, “The INFORMATION_SCHEMA FILES Table”](#).

The [INITIAL_SIZE](#) parameter sets the data file's total size in bytes. Once the file has been created, its size cannot be changed; however, you can add more data files to the tablespace using [ALTER TABLESPACE ... ADD DATAFILE](#). See [Section 13.1.8, “ALTER TABLESPACE Syntax”](#).

[INITIAL_SIZE](#) is optional; its default value is [128M](#).

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is 4G. (Bug #29186)

When setting `EXTENT_SIZE` or `INITIAL_SIZE` (either or both), you may optionally follow the number with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (for megabytes) or `G` (for gigabytes).

`INITIAL_SIZE`, `EXTENT_SIZE`, and `UNDO_BUFFER_SIZE` are subject to rounding as follows:

- `EXTENT_SIZE` and `UNDO_BUFFER_SIZE` are each rounded up to the nearest whole multiple of 32K.
- `INITIAL_SIZE` is rounded *down* to the nearest whole multiple of 32K.

For data files, `INITIAL_SIZE` is subject to further rounding; the result just obtained is rounded up to the nearest whole multiple of `EXTENT_SIZE` (after any rounding).

The rounding just described is done explicitly, and a warning is issued by the MySQL Server when any such rounding is performed. The rounded values are also used by the NDB kernel for calculating `INFORMATION_SCHEMA.FILES` column values and other purposes. However, to avoid an unexpected result, we suggest that you always use whole multiples of 32K in specifying these options.

`AUTOEXTEND_SIZE`, `MAX_SIZE`, `NODEGROUP`, `WAIT`, and `COMMENT` are parsed but ignored, and so currently have no effect. These options are intended for future expansion.

The `ENGINE` parameter determines the storage engine which uses this tablespace, with `engine_name` being the name of the storage engine. Currently, `engine_name` must be one of the values `NDB` or `NDBCLUSTER`.

When `CREATE TABLESPACE` is used with `ENGINE = NDB`, a tablespace and associated data file are created on each Cluster data node. You can verify that the data files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_NAME, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE TABLESPACE_NAME = 'newts' AND FILE_TYPE = 'DATAFILE';
+-----+-----+-----+
| LOGFILE_GROUP_NAME | FILE_NAME | EXTRA          |
+-----+-----+-----+
| lg_3               | newdata.dat | CLUSTER_NODE=3 |
| lg_3               | newdata.dat | CLUSTER_NODE=4 |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

(See [Section 20.29.1](#), “The `INFORMATION_SCHEMA.FILES` Table”.)

`CREATE TABLESPACE` is useful only with Disk Data storage for MySQL Cluster. See [Section 17.5.12](#), “MySQL Cluster Disk Data Tables”.

13.1.19. CREATE TRIGGER Syntax

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_body
```

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named `tbl_name`, which must refer to a permanent table. You cannot associate a trigger with a `TEMPORARY` table or a view.

`CREATE TRIGGER` requires the `TRIGGER [697]` privilege for the table associated with the trigger. The statement might also require the `SUPER [697]` privilege, depending on the `DEFINER` value, as described later in this section. If binary logging is enabled, `CREATE TRIGGER` might require the `SUPER [697]` privilege, as described in [Section 19.7, “Binary Logging of Stored Programs”](#).

The `DEFINER` clause determines the security context to be used when checking access privileges at trigger activation time. See later in this section for more information.

`trigger_time` is the trigger action time. It can be `BEFORE` or `AFTER` to indicate that the trigger activates before or after each row to be modified.

`trigger_event` indicates the kind of statement that activates the trigger. The `trigger_event` can be one of the following:

- `INSERT`: The trigger is activated whenever a new row is inserted into the table; for example, through `INSERT`, `LOAD DATA`, and `REPLACE` statements.
- `UPDATE`: The trigger is activated whenever a row is modified; for example, through `UPDATE` statements.
- `DELETE`: The trigger is activated whenever a row is deleted from the table; for example, through `DELETE` and `REPLACE` statements. However, `DROP TABLE` and `TRUNCATE TABLE` statements on the table do *not* activate this trigger, because they do not use `DELETE`. Dropping a partition does not activate `DELETE` triggers, either. See [Section 13.1.33, “TRUNCATE TABLE Syntax”](#).

It is important to understand that the `trigger_event` does not represent a literal type of SQL statement that activates the trigger so much as it represents a type of table operation. For example, an `INSERT` trigger is activated by not only `INSERT` statements but also `LOAD DATA` statements because both statements insert rows into a table.

A potentially confusing example of this is the `INSERT INTO ... ON DUPLICATE KEY UPDATE ...` syntax: a `BEFORE INSERT` trigger will activate for every row, followed by either an `AFTER INSERT` trigger or both the `BEFORE UPDATE` and `AFTER UPDATE` triggers, depending on whether there was a duplicate key for the row.

There cannot be two triggers for a given table that have the same trigger action time and event. For example, you cannot have two `BEFORE UPDATE` triggers for a table. But you can have a `BEFORE UPDATE` and a `BEFORE INSERT` trigger, or a `BEFORE UPDATE` and an `AFTER UPDATE` trigger.

`trigger_body` is the statement to execute when the trigger activates. If you want to execute multiple statements, use the `BEGIN ... END` compound statement construct. This also enables you to use the same statements that are permissible within stored routines. See [Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”](#). Some statements are not permitted in triggers; see [Section E.1, “Restrictions on Stored Programs”](#).

You can refer to columns in the subject table (the table associated with the trigger) by using the aliases `OLD` and `NEW`. `OLD.col_name` refers to a column of an existing row before it is updated or deleted. `NEW.col_name` refers to the column of a new row to be inserted or an existing row after it is updated.

MySQL stores the `sql_mode [556]` system variable setting that is in effect at the time a trigger is created, and always executes the trigger with this setting in force, *regardless of the server SQL mode in effect when the event begins executing*.



Note

Currently, cascaded foreign key actions do not activate triggers.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at trigger activation time. If a `user` value is given, it should be a MySQL account specified as `'user_name'@'host_name'` (the same format used in the `GRANT` statement), `CURRENT_USER` [1192], or `CURRENT_USER()` [1192]. The default `DEFINER` value is the user who executes the `CREATE TRIGGER` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the valid `DEFINER` user values:

- If you do not have the `SUPER` [697] privilege, the only permitted `user` value is your own account, either specified literally or by using `CURRENT_USER` [1192]. You cannot set the definer to some other account.
- If you have the `SUPER` [697] privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.
- Although it is possible to create a trigger with a nonexistent `DEFINER` account, it is not a good idea for such triggers to be activated until the account actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.

MySQL takes the `DEFINER` user into account when checking trigger privileges as follows:

- At `CREATE TRIGGER` time, the user who issues the statement must have the `TRIGGER` [697] privilege.
- At trigger activation time, privileges are checked against the `DEFINER` user. This user must have these privileges:
 - The `TRIGGER` [697] privilege.
 - The `SELECT` [697] privilege for the subject table if references to table columns occur using `OLD.col_name` or `NEW.col_name` in the trigger definition.
 - The `UPDATE` [697] privilege for the subject table if table columns are targets of `SET NEW.col_name = value` assignments in the trigger definition.
 - Whatever other privileges normally are required for the statements executed by the trigger.

For more information about trigger security, see [Section 19.6, “Access Control for Stored Programs and Views”](#).

Within a trigger, the `CURRENT_USER()` [1192] function returns the account used to check privileges at trigger activation time. This is the `DEFINER` user, not the user whose actions caused the trigger to be activated. For information about user auditing within triggers, see [Section 6.3.11, “SQL-Based MySQL Account Activity Auditing”](#).

If you use `LOCK TABLES` to lock a table that has triggers, the tables used within the trigger are also locked, as described in [Section 13.3.5.2, “LOCK TABLES and Triggers”](#).

In MySQL 5.5, you can write triggers containing direct references to tables by name, such as the trigger named `testref` shown in this example:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

delimiter |
```

```

CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW BEGIN
  INSERT INTO test2 SET a2 = NEW.a1;
  DELETE FROM test3 WHERE a3 = NEW.a1;
  UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END;
|
delimiter ;

INSERT INTO test3 (a3) VALUES
(NULL), (NULL), (NULL), (NULL), (NULL),
(NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
(0), (0), (0), (0), (0), (0), (0), (0), (0), (0);

```

Suppose that you insert the following values into table `test1` as shown here:

```

mysql> INSERT INTO test1 VALUES
-> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8 Duplicates: 0 Warnings: 0

```

As a result, the data in the four tables will be as follows:

```

mysql> SELECT * FROM test1;
+-----+
| a1    |
+-----+
| 1     |
| 3     |
| 1     |
| 7     |
| 1     |
| 8     |
| 4     |
| 4     |
+-----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+-----+
| a2    |
+-----+
| 1     |
| 3     |
| 1     |
| 7     |
| 1     |
| 8     |
| 4     |
| 4     |
+-----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
+-----+
| a3    |
+-----+
| 2     |
| 5     |
| 6     |
| 9     |
| 10    |
+-----+
5 rows in set (0.00 sec)

```

```
mysql> SELECT * FROM test4;
+-----+-----+
| a4 | b4 |
+-----+-----+
| 1 | 3 |
| 2 | 0 |
| 3 | 1 |
| 4 | 2 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 1 |
| 9 | 0 |
| 10 | 0 |
+-----+-----+
10 rows in set (0.00 sec)
```

13.1.20. CREATE VIEW Syntax

```
CREATE
  [OR REPLACE]
  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
  [DEFINER = { user | CURRENT_USER }]
  [SQL SECURITY { DEFINER | INVOKER }]
  VIEW view_name [(column_list)]
  AS select_statement
  [WITH [CASCADED | LOCAL] CHECK OPTION]
```

The `CREATE VIEW` statement creates a new view, or replaces an existing one if the `OR REPLACE` clause is given. If the view does not exist, `CREATE OR REPLACE VIEW` is the same as `CREATE VIEW`. If the view does exist, `CREATE OR REPLACE VIEW` is the same as `ALTER VIEW`.

The `select_statement` is a `SELECT` statement that provides the definition of the view. (When you select from the view, you select in effect using the `SELECT` statement.) `select_statement` can select from base tables or other views.

The view definition is “frozen” at creation time, so changes to the underlying tables afterward do not affect the view definition. For example, if a view is defined as `SELECT *` on a table, new columns added to the table later do not become part of the view.

The `ALGORITHM` clause affects how MySQL processes the view. The `DEFINER` and `SQL SECURITY` clauses specify the security context to be used when checking access privileges at view invocation time. The `WITH CHECK OPTION` clause can be given to constrain inserts or updates to rows in tables referenced by the view. These clauses are described later in this section.

The `CREATE VIEW` statement requires the `CREATE VIEW [696]` privilege for the view, and some privilege for each column selected by the `SELECT` statement. For columns used elsewhere in the `SELECT` statement you must have the `SELECT [697]` privilege. If the `OR REPLACE` clause is present, you must also have the `DROP [696]` privilege for the view. `CREATE VIEW` might also require the `SUPER [697]` privilege, depending on the `DEFINER` value, as described later in this section.

When a view is referenced, privilege checking occurs as described later in this section.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, specify the name as `db_name.view_name` when you create it:

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Within a database, base tables and views share the same namespace, so a base table and a view cannot have the same name.

Columns retrieved by the `SELECT` statement can be simple references to table columns. They can also be expressions that use functions, constant values, operators, and so forth.

Views must have unique column names with no duplicates, just like base tables. By default, the names of the columns retrieved by the `SELECT` statement are used for the view column names. To define explicit names for the view columns, the optional `column_list` clause can be given as a list of comma-separated identifiers. The number of names in `column_list` must be the same as the number of columns retrieved by the `SELECT` statement.

Unqualified table or view names in the `SELECT` statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the proper database name.

A view can be created from many kinds of `SELECT` statements. It can refer to base tables or other views. It can use joins, `UNION`, and subqueries. The `SELECT` need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+-----+-----+-----+
| qty  | price | value |
+-----+-----+-----+
| 3    | 50    | 150   |
+-----+-----+-----+
```

A view definition is subject to the following restrictions:

- The `SELECT` statement cannot contain a subquery in the `FROM` clause.
- The `SELECT` statement cannot refer to system or user variables.
- Within a stored program, the definition cannot refer to program parameters or local variables.
- The `SELECT` statement cannot refer to prepared statement parameters.
- Any table or view referred to in the definition must exist. However, after a view has been created, it is possible to drop a table or view that the definition refers to. In this case, use of the view results in an error. To check a view definition for problems of this kind, use the `CHECK TABLE` statement.
- The definition cannot refer to a `TEMPORARY` table, and you cannot create a `TEMPORARY` view.
- Any tables named in the view definition must exist at definition time.
- You cannot associate a trigger with a view.
- Aliases for column names in the `SELECT` statement are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

`ORDER BY` is permitted in a view definition, but it is ignored if you select from a view using a statement that has its own `ORDER BY`.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a `LIMIT` clause, and you select from the view using a statement that has its own `LIMIT` clause, it is undefined which limit applies. This same principle applies to options such as `ALL`, `DISTINCT`, or `SQL_SMALL_RESULT` that

follow the `SELECT` keyword, and to clauses such as `INTO`, `FOR UPDATE`, `LOCK IN SHARE MODE`, and `PROCEDURE`.

If you create a view and then change the query processing environment by changing system variables, that may affect the results that you get from the view:

```
mysql> CREATE VIEW v (mycol) AS SELECT 'abc';
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-----+
| mycol |
+-----+
| mycol |
+-----+
1 row in set (0.01 sec)

mysql> SET sql_mode = 'ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-----+
| mycol |
+-----+
| abc   |
+-----+
1 row in set (0.00 sec)
```

The `DEFINER` and `SQL SECURITY` clauses determine which MySQL account to use when checking access privileges for the view when a statement is executed that references the view. The valid `SQL SECURITY` characteristic values are `DEFINER` and `INVOKER`. These indicate that the required privileges must be held by the user who defined or invoked the view, respectively. The default `SQL SECURITY` value is `DEFINER`.

If a `user` value is given for the `DEFINER` clause, it should be a MySQL account specified as `'user_name'@'host_name'` (the same format used in the `GRANT` statement), `CURRENT_USER` [1192], or `CURRENT_USER()` [1192]. The default `DEFINER` value is the user who executes the `CREATE VIEW` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the valid `DEFINER` user values:

- If you do not have the `SUPER` [697] privilege, the only valid `user` value is your own account, either specified literally or by using `CURRENT_USER` [1192]. You cannot set the definer to some other account.
- If you have the `SUPER` [697] privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.
- Although it is possible to create a view with a nonexistent `DEFINER` account, an error occurs when the view is referenced if the `SQL SECURITY` value is `DEFINER` but the definer account does not exist.

For more information about view security, see [Section 19.6, “Access Control for Stored Programs and Views”](#).

Within a view definition, `CURRENT_USER` [1192] returns the view's `DEFINER` value by default. For views defined with the `SQL SECURITY INVOKER` characteristic, `CURRENT_USER` [1192] returns the account for the view's invoker. For information about user auditing within views, see [Section 6.3.11, “SQL-Based MySQL Account Activity Auditing”](#).

Within a stored routine that is defined with the `SQL SECURITY DEFINER` characteristic, `CURRENT_USER [1192]` returns the routine's `DEFINER` value. This also affects a view defined within such a routine, if the view definition contains a `DEFINER` value of `CURRENT_USER [1192]`.

View privileges are checked like this:

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to table columns, the creator must have some privilege for each column in the select list of the definition, and the `SELECT [697]` privilege for each column used elsewhere in the definition. If the definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required at function invocation time can be checked only as it executes: For different invocations, different execution paths within the function might be taken.
- The user who references a view must have appropriate privileges to access it (`SELECT [697]` to select from it, `INSERT [696]` to insert into it, and so forth.)
- When a view has been referenced, privileges for objects accessed by the view are checked against the privileges held by the view `DEFINER` account or invoker, depending on whether the `SQL SECURITY` characteristic is `DEFINER` or `INVOKER`, respectively.
- If reference to a view causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function `SQL SECURITY` characteristic is `DEFINER` or `INVOKER`. If the security characteristic is `DEFINER`, the function runs with the privileges of the `DEFINER` account. If the characteristic is `INVOKER`, the function runs with the privileges determined by the view's `SQL SECURITY` characteristic.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function `f()`:

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

Suppose that `f()` contains a statement such as this:

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

The privileges required for executing statements within `f()` need to be checked when `f()` executes. This might mean that privileges are needed for `p1()` or `p2()`, depending on the execution path within `f()`. Those privileges must be checked at runtime, and the user who must possess the privileges is determined by the `SQL SECURITY` values of the view `v` and the function `f()`.

The `DEFINER` and `SQL SECURITY` clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for `SQL SECURITY DEFINER`. The standard says that the definer of the view, which is the same as the owner of the view's schema, gets applicable privileges on the view (for example, `SELECT [697]`) and may grant them. MySQL has no concept of a schema “owner”, so MySQL adds a clause to identify the definer. The `DEFINER` clause is an extension where the intent is to have what the standard has; that is, a permanent record of who defined the view. This is why the default `DEFINER` value is the account of the view creator.

The optional `ALGORITHM` clause is a MySQL extension to standard SQL. It affects how MySQL processes the view. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`. The default algorithm is `UNDEFINED` if no `ALGORITHM` clause is present. For more information, see [Section 19.5.2, “View Processing Algorithms”](#).

Some views are updatable. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable.

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts or updates to rows except those for which the `WHERE` clause in the `select_statement` is true.

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. The `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is `CASCADED`.

For more information about updatable views and the `WITH CHECK OPTION` clause, see [Section 19.5.3, “Updatable and Insertable Views”](#).

13.1.21. DROP DATABASE Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

`DROP DATABASE` drops all tables in the database and deletes the database. Be very careful with this statement! To use `DROP DATABASE`, you need the `DROP [696]` privilege on the database. `DROP SCHEMA` is a synonym for `DROP DATABASE`.



Important

When a database is dropped, user privileges on the database are *not* automatically dropped. See [Section 13.7.1.3, “GRANT Syntax”](#).

`IF EXISTS` is used to prevent an error from occurring if the database does not exist.

If the default database is dropped, the default database is unset (the `DATABASE() [1193]` function returns `NULL`).

If you use `DROP DATABASE` on a symbolically linked database, both the link and the original database are deleted.

`DROP DATABASE` returns the number of tables that were removed. This corresponds to the number of `.frm` files removed.

The `DROP DATABASE` statement removes from the given database directory those files and directories that MySQL itself may create during normal operation:

- All files with the following extensions.

<code>.BAK</code>	<code>.DAT</code>	<code>.HSH</code>	<code>.MRG</code>
<code>.MYD</code>	<code>.MYI</code>	<code>.TRG</code>	<code>.TRN</code>
<code>.db</code>	<code>.frm</code>	<code>.ibd</code>	<code>.ndb</code>
<code>.par</code>			

- The `db.opt` file, if it exists.

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the `DROP DATABASE` statement again.

You can also drop databases with `mysqladmin`. See [Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”](#).

13.1.22. DROP EVENT Syntax

```
DROP EVENT [IF EXISTS] event_name
```

This statement drops the event named *event_name*. The event immediately ceases being active, and is deleted completely from the server.

If the event does not exist, the error `ERROR 1517 (HY000): Unknown event 'event_name'` results. You can override this and cause the statement to generate a warning for nonexistent events instead using `IF EXISTS`.

This statement requires the [EVENT \[696\]](#) privilege for the schema to which the event to be dropped belongs.

13.1.23. DROP FUNCTION Syntax

The `DROP FUNCTION` statement is used to drop stored functions and user-defined functions (UDFs):

- For information about dropping stored functions, see [Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Syntax”](#).
- For information about dropping user-defined functions, see [Section 13.7.3.2, “DROP FUNCTION Syntax”](#).

13.1.24. DROP INDEX Syntax

```
DROP [ONLINE|OFFLINE] INDEX index_name ON tbl_name
```

`DROP INDEX` drops the index named *index_name* from the table *tbl_name*. This statement is mapped to an `ALTER TABLE` statement to drop the index. See [Section 13.1.7, “ALTER TABLE Syntax”](#).

To drop a primary key, the index name is always `PRIMARY`, which must be specified as a quoted identifier because `PRIMARY` is a reserved word:

```
DROP INDEX `PRIMARY` ON t;
```

Indexes on variable-width columns of `NDBCLUSTER` tables are dropped online; that is, without any table copying. The table is not locked against access from other MySQL Cluster API nodes, although it is locked against other operations on the *same* API node for the duration of the operation. This is done automatically by the server whenever it determines that it is possible to do so; you do not have to use any special SQL syntax or server options to cause it to happen.

In standard MySQL 5.5 releases, it is not possible to override the server when it determines that an index is to be dropped without table copying. In MySQL Cluster, you can drop indexes offline (which causes the table to be locked for all API nodes in the cluster) using the `OFFLINE` keyword. The rules and limitations governing `DROP OFFLINE INDEX` and `DROP ONLINE INDEX` are the same as for `ALTER OFFLINE TABLE ... DROP INDEX` and `ALTER ONLINE TABLE ... DROP INDEX`. You cannot cause the noncopying dropping of an index that would normally be dropped offline by using the `ONLINE` keyword: if it is not possible to perform the `DROP` operation without table copying, the server ignores the `ONLINE` keyword. For more information, see [Section 13.1.7.2, “ALTER TABLE Online Operations”](#).



Note

The `ONLINE` and `OFFLINE` keywords are available only in MySQL Cluster; attempting to use these keywords in standard MySQL Server 5.5 releases results in a syntax error.

13.1.25. DROP LOGFILE GROUP Syntax

```
DROP LOGFILE GROUP logfile_group
ENGINE [=] engine_name
```

This statement drops the log file group named *logfile_group*. The log file group must already exist or an error results. (For information on creating log file groups, see [Section 13.1.14](#), “CREATE LOGFILE GROUP Syntax”.)



Important

Before dropping a log file group, you must drop all tablespaces that use that log file group for UNDO logging.

The required `ENGINE` clause provides the name of the storage engine used by the log file group to be dropped. Currently, the only permitted values for *engine_name* are `NDB` and `NDBCLUSTER`.

`DROP LOGFILE GROUP` is useful only with Disk Data storage for MySQL Cluster. See [Section 17.5.12](#), “MySQL Cluster Disk Data Tables”.

13.1.26. DROP PROCEDURE and DROP FUNCTION Syntax

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

This statement is used to drop a stored procedure or function. That is, the specified routine is removed from the server. You must have the `ALTER ROUTINE [695]` privilege for the routine. (If the `automatic_sp_privileges` system variable is enabled, that privilege and `EXECUTE [696]` are granted automatically to the routine creator when the routine is created and dropped from the creator when the routine is dropped. See [Section 19.2.2](#), “Stored Routines and MySQL Privileges”.)

The `IF EXISTS` clause is a MySQL extension. It prevents an error from occurring if the procedure or function does not exist. A warning is produced that can be viewed with `SHOW WARNINGS`.

`DROP FUNCTION` is also used to drop user-defined functions (see [Section 13.7.3.2](#), “DROP FUNCTION Syntax”).

13.1.27. DROP SERVER Syntax

```
DROP SERVER [ IF EXISTS ] server_name
```

Drops the server definition for the server named *server_name*. The corresponding row in the `mysql.servers` table is deleted. This statement requires the `SUPER [697]` privilege.

Dropping a server for a table does not affect any `FEDERATED` tables that used this connection information when they were created. See [Section 13.1.16](#), “CREATE SERVER Syntax”.

`DROP SERVER` does not cause an automatic commit.

In MySQL 5.5, `DROP SERVER` is not written to the binary log, regardless of the logging format that is in use.

13.1.28. DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

`DROP TABLE` removes one or more tables. You must have the `DROP` [696] privilege for each table. All table data and the table definition are *removed*, so *be careful* with this statement! If any of the tables named in the argument list do not exist, MySQL returns an error indicating by name which nonexisting tables it was unable to drop, but it also drops all of the tables in the list that do exist.



Important

When a table is dropped, user privileges on the table are *not* automatically dropped. See [Section 13.7.1.3, “GRANT Syntax”](#).

Note that for a partitioned table, `DROP TABLE` permanently removes the table definition, all of its partitions, and all of the data which was stored in those partitions. It also removes the partitioning definition (`.par`) file associated with the dropped table.

Use `IF EXISTS` to prevent an error from occurring for tables that do not exist. A `NOTE` is generated for each nonexistent table when using `IF EXISTS`. See [Section 13.7.5.41, “SHOW WARNINGS Syntax”](#).

`RESTRICT` and `CASCADE` are permitted to make porting easier. In MySQL 5.5, they do nothing.



Note

`DROP TABLE` automatically commits the current active transaction, unless you use the `TEMPORARY` keyword.

The `TEMPORARY` keyword has the following effects:

- The statement drops only `TEMPORARY` tables.
- The statement does not end an ongoing transaction.
- No access rights are checked. (A `TEMPORARY` table is visible only to the session that created it, so no check is necessary.)

Using `TEMPORARY` is a good way to ensure that you do not accidentally drop a non-`TEMPORARY` table.

13.1.29. DROP TABLESPACE Syntax

```
DROP TABLESPACE tablespace_name
ENGINE [=] engine_name
```

This statement drops a tablespace that was previously created using `CREATE TABLESPACE` (see [Section 13.1.18, “CREATE TABLESPACE Syntax”](#)).



Important

The tablespace to be dropped must not contain any data files; in other words, before you can drop a tablespace, you must first drop each of its data files using `ALTER TABLESPACE ... DROP DATAFILE` (see [Section 13.1.8, “ALTER TABLESPACE Syntax”](#)).

The `ENGINE` clause (required) specifies the storage engine used by the tablespace. Currently, the only accepted values for `engine_name` are `NDB` and `NDBCLUSTER`.

`DROP TABLESPACE` is useful only with Disk Data storage for MySQL Cluster. See [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#).

13.1.30. DROP TRIGGER Syntax

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. `DROP TRIGGER` requires the `TRIGGER [697]` privilege for the table associated with the trigger.

Use `IF EXISTS` to prevent an error from occurring for a trigger that does not exist. A `NOTE` is generated for a nonexistent trigger when using `IF EXISTS`. See [Section 13.7.5.41, “SHOW WARNINGS Syntax”](#).

Triggers for a table are also dropped if you drop the table.



Note

When upgrading from a version of MySQL older than MySQL 5.0.10 to 5.0.10 or newer—including all MySQL 5.5 releases—you must drop all triggers and re-create them. Otherwise, `DROP TRIGGER` does not work for older triggers after the upgrade. See [Upgrading from MySQL 4.1 to 5.0](#), for a suggested upgrade procedure.

13.1.31. DROP VIEW Syntax

```
DROP VIEW [IF EXISTS]
  view_name [, view_name] ...
  [RESTRICT | CASCADE]
```

`DROP VIEW` removes one or more views. You must have the `DROP [696]` privilege for each view. If any of the views named in the argument list do not exist, MySQL returns an error indicating by name which nonexistent views it was unable to drop, but it also drops all of the views in the list that do exist.

The `IF EXISTS` clause prevents an error from occurring for views that don't exist. When this clause is given, a `NOTE` is generated for each nonexistent view. See [Section 13.7.5.41, “SHOW WARNINGS Syntax”](#).

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

13.1.32. RENAME TABLE Syntax

```
RENAME TABLE tbl_name TO new_tbl_name
  [, tbl_name2 TO new_tbl_name2] ...
```

This statement renames one or more tables.

The rename operation is done atomically, which means that no other session can access any of the tables while the rename is running. For example, if you have an existing table `old_table`, you can create another table `new_table` that has the same structure but is empty, and then replace the existing table with the empty one as follows (assuming that `backup_table` does not already exist):

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

If the statement renames more than one table, renaming operations are done from left to right. If you want to swap two table names, you can do so like this (assuming that `tmp_table` does not already exist):

```
RENAME TABLE old_table TO tmp_table,
  new_table TO old_table,
  tmp_table TO new_table;
```

As long as two databases are on the same file system, you can use `RENAME TABLE` to move a table from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

If there are any triggers associated with a table which is moved to a different database using `RENAME TABLE`, then the statement fails with the error `Trigger in wrong schema`.

`RENAME TABLE` also works for views, as long as you do not try to rename a view into a different database.

Any privileges granted specifically for the renamed table or view are not migrated to the new name. They must be changed manually.

When you execute `RENAME`, you cannot have any locked tables or active transactions. You must also have the `ALTER` [695] and `DROP` [696] privileges on the original table, and the `CREATE` [695] and `INSERT` [696] privileges on the new table.

If MySQL encounters any errors in a multiple-table rename, it does a reverse rename for all renamed tables to return everything to its original state.

You cannot use `RENAME` to rename a `TEMPORARY` table. However, you can use `ALTER TABLE` instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

13.1.33. TRUNCATE TABLE Syntax

```
TRUNCATE [TABLE] tbl_name
```

`TRUNCATE TABLE` empties a table completely. It requires the `DROP` [696] privilege.

Logically, `TRUNCATE TABLE` is similar to a `DELETE` statement that deletes all rows, or a sequence of `DROP TABLE` and `CREATE TABLE` statements. To achieve high performance, it bypasses the DML method of deleting data. Thus, it cannot be rolled back, it does not cause `ON DELETE` triggers to fire, and it cannot be performed for `InnoDB` tables with parent-child foreign key relationships.

Although `TRUNCATE TABLE` is similar to `DELETE`, it is classified as a DDL statement rather than a DML statement. It differs from `DELETE` in the following ways in MySQL 5.5:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- Truncate operations cause an implicit commit, and so cannot be rolled back.
- Truncation operations cannot be performed if the session holds an active table lock.
- `TRUNCATE TABLE` fails for an `InnoDB` table if there are any `FOREIGN KEY` constraints from other tables that reference the table. Foreign key constraints between columns of the same table are permitted.
- Truncation operations do not return a meaningful value for the number of deleted rows. The usual result is “0 rows affected,” which should be interpreted as “no information.”
- As long as the table format file `tbl_name.frm` is valid, the table can be re-created as an empty table with `TRUNCATE TABLE`, even if the data or index files have become corrupted.
- Any `AUTO_INCREMENT` value is reset to its start value. This is true even for `MyISAM` and `InnoDB`, which normally do not reuse sequence values.
- When used with partitioned tables, `TRUNCATE TABLE` preserves the partitioning; that is, the data and index files are dropped and re-created, while the partition definitions (`.par`) file is unaffected.

- The `TRUNCATE TABLE` statement does not invoke `ON DELETE` triggers.

`TRUNCATE TABLE` for a table closes all handlers for the table that were opened with `HANDLER OPEN`.

`TRUNCATE TABLE` is treated for purposes of binary logging and replication as `DROP TABLE` followed by `CREATE TABLE`—that is, as DDL rather than DML. This is due to the fact that, when using `InnoDB` and other transactional storage engines where the transaction isolation level does not permit statement-based logging (`READ COMMITTED` or `READ UNCOMMITTED`), the statement was not logged and replicated when using `STATEMENT` or `MIXED` logging mode. (Bug #36763) However, it is still applied on replication slaves using `InnoDB` in the manner described previously.

`TRUNCATE TABLE` can be used with Performance Schema summary tables, but the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. See [Section 21.7.4, “Performance Schema Summary Tables”](#).

13.2. Data Manipulation Statements

13.2.1. `CALL` Syntax

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

The `CALL` statement invokes a stored procedure that was defined previously with `CREATE PROCEDURE`.

Stored procedures that take no arguments can be invoked without parentheses. That is, `CALL p()` and `CALL p` are equivalent.

`CALL` can pass back values to its caller using parameters that are declared as `OUT` or `INOUT` parameters. When the procedure returns, a client program can also obtain the number of rows affected for the final statement executed within the routine: At the SQL level, call the `ROW_COUNT()` [1196] function; from the C API, call the `mysql_affected_rows()` function.

To get back a value from a procedure using an `OUT` or `INOUT` parameter, pass the parameter by means of a user variable, and then check the value of the variable after the procedure returns. (If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.) For an `INOUT` parameter, initialize its value before passing it to the procedure. The following procedure has an `OUT` parameter that the procedure sets to the current server version, and an `INOUT` value that the procedure increments by one from its current value:

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;
```

Before calling the procedure, initialize the variable to be passed as the `INOUT` parameter. After calling the procedure, the values of the two variables will have been set or modified:

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+-----+-----+
| @version | @increment |
+-----+-----+
| 5.5.3-m3-log | 11 |
```

In prepared `CALL` statements used with `PREPARE` and `EXECUTE`, placeholders can be used for `IN` parameters. For `OUT` and `INOUT` parameters, placeholder support is available as of MySQL 5.5.3. These types of parameters can be used as follows:

```
mysql> SET @increment = 10;
mysql> PREPARE s FROM 'CALL p(?, ?)';
mysql> EXECUTE s USING @version, @increment;
mysql> SELECT @version, @increment;
+-----+-----+
| @version | @increment |
+-----+-----+
| 5.5.3-m3-log | 11 |
+-----+-----+
```

Before MySQL 5.5.3, placeholder support is not available for `OUT` or `INOUT` parameters. To work around this limitation for `OUT` and `INOUT` parameters, forego the use of placeholders; instead, refer to user variables in the `CALL` statement itself and do not specify them in the `EXECUTE` statement:

```
mysql> SET @increment = 10;
mysql> PREPARE s FROM 'CALL p(@version, @increment)';
mysql> EXECUTE s;
mysql> SELECT @version, @increment;
+-----+-----+
| @version | @increment |
+-----+-----+
| 5.5.0-m2-log | 11 |
+-----+-----+
```

To write C programs that use the `CALL` SQL statement to execute stored procedures that produce result sets, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. `CLIENT_MULTI_RESULTS` must also be enabled if `CALL` is used to execute any stored procedure that contains prepared statements. It cannot be determined when such a procedure is loaded whether those statements will produce result sets, so it is necessary to assume that they will.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). As of MySQL 5.5.3, `CLIENT_MULTI_RESULTS` is enabled by default.

To process the result of a `CALL` statement executed using `mysql_query()` or `mysql_real_query()`, use a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 22.8.13, "C API Support for Multiple Statement Execution"](#).

For programs written in a language that provides a MySQL interface, there is no native method prior to MySQL 5.5.3 for directly retrieving the results of `OUT` or `INOUT` parameters from `CALL` statements. To get the parameter values, pass user-defined variables to the procedure in the `CALL` statement and then execute a `SELECT` statement to produce a result set containing the variable values. To handle an `INOUT` parameter, execute a statement prior to the `CALL` that sets the corresponding user variable to the value to be passed to the procedure.

The following example illustrates the technique (without error checking) for the stored procedure `p` described earlier that has an `OUT` parameter and an `INOUT` parameter:

```
mysql_query(mysql, "SET @increment = 10");
mysql_query(mysql, "CALL p(@version, @increment)");
mysql_query(mysql, "SELECT @version, @increment");
```

```
result = mysql_store_result(mysql);
row = mysql_fetch_row(result);
mysql_free_result(result);
```

After the preceding code executes, `row[0]` and `row[1]` contain the values of `@version` and `@increment`, respectively.

As of MySQL 5.5.3, C programs can use the prepared-statement interface to execute `CALL` statements and access `OUT` and `INOUT` parameters. This is done by processing the result of a `CALL` statement using a loop that calls `mysql_stmt_next_result()` to determine whether there are more results. For an example, see [Section 22.8.16, “C API Support for Prepared `CALL` Statements”](#). Languages that provide a MySQL interface can use prepared `CALL` statements to directly retrieve `OUT` and `INOUT` procedure parameters.

13.2.2. DELETE Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
  tbl_name[.*] [, tbl_name[.*]] ...
FROM table_references
  [WHERE where_condition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
  FROM tbl_name[.*] [, tbl_name[.*]] ...
  USING table_references
  [WHERE where_condition]
```

For the single-table syntax, the `DELETE` statement deletes rows from `tbl_name` and returns a count of the number of deleted rows. This count can be obtained by calling the `ROW_COUNT()` [1196] function (see [Section 12.14, “Information Functions”](#)). The `WHERE` clause, if given, specifies the conditions that identify which rows to delete. With no `WHERE` clause, all rows are deleted. If the `ORDER BY` clause is specified, the rows are deleted in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be deleted.

For the multiple-table syntax, `DELETE` deletes from each `tbl_name` the rows that satisfy the conditions. In this case, `ORDER BY` and `LIMIT` cannot be used.

`where_condition` is an expression that evaluates to true for each row to be deleted. It is specified as described in [Section 13.2.9, “SELECT Syntax”](#).

Currently, you cannot delete from a table and select from the same table in a subquery.

You need the `DELETE` [696] privilege on a table to delete rows from it. You need only the `SELECT` [697] privilege for any columns that are only read, such as those named in the `WHERE` clause.

As stated, a `DELETE` statement with no `WHERE` clause deletes all rows. A faster way to do this, when you do not need to know the number of deleted rows, is to use `TRUNCATE TABLE`. However, within a transaction or if you have a lock on the table, `TRUNCATE TABLE` cannot be used whereas `DELETE` can.

See [Section 13.1.33, “TRUNCATE TABLE Syntax”](#), and [Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”](#).

If you delete the row containing the maximum value for an `AUTO_INCREMENT` column, the value is not reused for a `MyISAM` or `InnoDB` table. If you delete all rows in the table with `DELETE FROM tbl_name` (without a `WHERE` clause) in `autocommit [484]` mode, the sequence starts over for all storage engines except `InnoDB` and `MyISAM`. There are some exceptions to this behavior for `InnoDB` tables, as discussed in [Section 14.3.5.3, “AUTO_INCREMENT Handling in InnoDB”](#).

For `MyISAM` tables, you can specify an `AUTO_INCREMENT` secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for `MyISAM` tables. See [Section 3.6.9, “Using AUTO_INCREMENT”](#).

The `DELETE` statement supports the following modifiers:

- If you specify `LOW_PRIORITY`, the server delays execution of the `DELETE` until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).
- For `MyISAM` tables, if you use the `QUICK` keyword, the storage engine does not merge index leaves during delete, which may speed up some kinds of delete operations.
- The `IGNORE` keyword causes MySQL to ignore all errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of `IGNORE` are returned as warnings.

The speed of delete operations may also be affected by factors discussed in [Section 8.2.2.3, “Speed of DELETE Statements”](#).

In `MyISAM` tables, deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions. To reclaim unused space and reduce file sizes, use the `OPTIMIZE TABLE` statement or the `myisamchk` utility to reorganize tables. `OPTIMIZE TABLE` is easier to use, but `myisamchk` is faster. See [Section 13.7.2.4, “OPTIMIZE TABLE Syntax”](#), and [Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

The `QUICK` modifier affects whether index leaves are merged for delete operations. `DELETE QUICK` is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

`DELETE QUICK` is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of `QUICK` can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed `AUTO_INCREMENT` column.
2. Insert many rows into the table. Each insert results in an index value that is added to the high end of the index.
3. Delete a block of rows at the low end of the column range using `DELETE QUICK`.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of `QUICK`. They remain underfilled when new inserts occur, because new rows do not have index values in the deleted range. Furthermore, they remain underfilled even if you later use `DELETE` without `QUICK`, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, use `OPTIMIZE TABLE`.

If you are going to delete many rows from a table, it might be faster to use [DELETE QUICK](#) followed by [OPTIMIZE TABLE](#). This rebuilds the index rather than performing many index block merge operations.

The MySQL-specific [LIMIT *row_count*](#) option to [DELETE](#) tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to ensure that a given [DELETE](#) statement does not take too much time. You can simply repeat the [DELETE](#) statement until the number of affected rows is less than the [LIMIT](#) value.

If the [DELETE](#) statement includes an [ORDER BY](#) clause, rows are deleted in the order specified by the clause. This is useful primarily in conjunction with [LIMIT](#). For example, the following statement finds rows matching the [WHERE](#) clause, sorts them by [timestamp_column](#), and deletes the first (oldest) one:

```
DELETE FROM somelog WHERE user = 'jcole'
ORDER BY timestamp_column LIMIT 1;
```

[ORDER BY](#) may also be useful in some cases to delete rows in an order required to avoid referential integrity violations.

If you are deleting many rows from a large table, you may exceed the lock table size for an [InnoDB](#) table. To avoid this problem, or simply to minimize the time that the table remains locked, the following strategy (which does not use [DELETE](#) at all) might be helpful:

1. Select the rows *not* to be deleted into an empty table that has the same structure as the original table:

```
INSERT INTO t_copy SELECT * FROM t WHERE ... ;
```

2. Use [RENAME TABLE](#) to atomically move the original table out of the way and rename the copy to the original name:

```
RENAME TABLE t TO t_old, t_copy TO t;
```

3. Drop the original table:

```
DROP TABLE t_old;
```

No other sessions can access the tables involved while [RENAME TABLE](#) executes, so the rename operation is not subject to concurrency problems. See [Section 13.1.32, “RENAME TABLE Syntax”](#).

You can specify multiple tables in a [DELETE](#) statement to delete rows from one or more tables depending on the particular condition in the [WHERE](#) clause. However, you cannot use [ORDER BY](#) or [LIMIT](#) in a multiple-table [DELETE](#). The [table_references](#) clause lists the tables involved in the join. Its syntax is described in [Section 13.2.9.2, “JOIN Syntax”](#).

For the first multiple-table syntax, only matching rows from the tables listed before the [FROM](#) clause are deleted. For the second multiple-table syntax, only matching rows from the tables listed in the [FROM](#) clause (before the [USING](#) clause) are deleted. The effect is that you can delete rows from many tables at the same time and have additional tables that are used only for searching:

```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three tables when searching for rows to delete, but delete matching rows only from tables [t1](#) and [t2](#).

The preceding examples use `INNER JOIN`, but multiple-table `DELETE` statements can use other types of join permitted in `SELECT` statements, such as `LEFT JOIN`. For example, to delete rows that exist in `t1` that have no match in `t2`, use a `LEFT JOIN`:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
```

The syntax permits `. *` after each `tbl_name` for compatibility with `Access`.

If you use a multiple-table `DELETE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, you should delete from a single table and rely on the `ON DELETE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly.



Note

If you declare an alias for a table, you must use the alias when referring to the table:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

Table aliases in a multiple-table `DELETE` should be declared only in the `table_references` part of the statement.

Correct:

```
DELETE a1, a2 FROM t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;

DELETE FROM a1, a2 USING t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;
```

Incorrect:

```
DELETE t1 AS a1, t2 AS a2 FROM t1 INNER JOIN t2
WHERE a1.id=a2.id;

DELETE FROM t1 AS a1, t2 AS a2 USING t1 INNER JOIN t2
WHERE a1.id=a2.id;
```

Declaration of aliases other than in the `table_references` part of the statement should be avoided because that can lead to ambiguous statements that have unexpected results such as deleting rows from the wrong table. This is such a statement:

```
DELETE t1 AS a2 FROM t1 AS a1 INNER JOIN t2 AS a2;
```

Before MySQL 5.5.3, alias declarations outside the `table_references` part of the statement are disallowed for the `USING` variant of multiple-table `DELETE` syntax. As of MySQL 5.5.3, alias declarations outside `table_references` are disallowed for all multiple-table `DELETE` statements.

Before MySQL 5.5.3, for alias references in the list of tables from which to delete rows in a multiple-table delete, the default database is used unless one is specified explicitly. For example, if the default database is `db1`, the following statement does not work because the unqualified alias reference `a2` is interpreted as having a database of `db1`:

```
DELETE a1, a2 FROM db1.t1 AS a1 INNER JOIN db2.t2 AS a2
WHERE a1.id=a2.id;
```

To correctly match an alias that refers to a table outside the default database, you must explicitly qualify the reference with the name of the proper database:

```
DELETE a1, db2.a2 FROM db1.t1 AS a1 INNER JOIN db2.t2 AS a2
WHERE a1.id=a2.id;
```

As of MySQL 5.5.3, alias resolution does not require qualification and alias references should not be qualified with the database name. Qualified names are interpreted as referring to tables, not aliases.

13.2.3. DO Syntax

```
DO expr [, expr] ...
```

DO executes the expressions but does not return any results. In most respects, DO is shorthand for SELECT *expr*, ..., but has the advantage that it is slightly faster when you do not care about the result.

DO is useful primarily with functions that have side effects, such as RELEASE_LOCK() [1201].

13.2.4. HANDLER Syntax

```
HANDLER tbl_name OPEN [ [AS] alias ]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
  [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
  [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
  [ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

The HANDLER statement provides direct access to table storage engine interfaces. It is available for InnoDB and MyISAM tables.

The HANDLER ... OPEN statement opens a table, making it accessible using subsequent HANDLER ... READ statements. This table object is not shared by other sessions and is not closed until the session calls HANDLER ... CLOSE or the session terminates. If you open the table using an alias, further references to the open table with other HANDLER statements must use the alias rather than the table name.

The first HANDLER ... READ syntax fetches a row where the index specified satisfies the given values and the WHERE condition is met. If you have a multiple-column index, specify the index column values as a comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index *my_idx* includes three columns named *col_a*, *col_b*, and *col_c*, in that order. The HANDLER statement can specify values for all three columns in the index, or for the columns in a leftmost prefix. For example:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

To employ the HANDLER interface to refer to a table's PRIMARY KEY, use the quoted identifier `PRIMARY`:

```
HANDLER tbl_name READ `PRIMARY` ...
```

The second HANDLER ... READ syntax fetches a row from the table in index order that matches the WHERE condition.

The third HANDLER ... READ syntax fetches a row from the table in natural row order that matches the WHERE condition. It is faster than HANDLER *tbl_name* READ *index_name* when a full table scan is

desired. Natural row order is the order in which rows are stored in a `MyISAM` table data file. This statement works for `InnoDB` tables as well, but there is no such concept because there is no separate data file.

Without a `LIMIT` clause, all forms of `HANDLER ... READ` fetch a single row if one is available. To return a specific number of rows, include a `LIMIT` clause. It has the same syntax as for the `SELECT` statement. See [Section 13.2.9, “SELECT Syntax”](#).

`HANDLER ... CLOSE` closes a table that was opened with `HANDLER ... OPEN`.

There are several reasons to use the `HANDLER` interface instead of normal `SELECT` statements:

- `HANDLER` is faster than `SELECT`:
 - A designated storage engine handler object is allocated for the `HANDLER ... OPEN`. The object is reused for subsequent `HANDLER` statements for that table; it need not be reinitialized for each one.
 - There is less parsing involved.
 - There is no optimizer or query-checking overhead.
 - The handler interface does not have to provide a consistent look of the data (for example, `dirty reads` are permitted), so the storage engine can use optimizations that `SELECT` does not normally permit.
- `HANDLER` makes it easier to port to MySQL applications that use a low-level `ISAM`-like interface.
- `HANDLER` enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with `SELECT`. The `HANDLER` interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

`HANDLER` is a somewhat low-level statement. For example, it does not provide consistency. That is, `HANDLER ... OPEN` does *not* take a snapshot of the table, and does *not* lock the table. This means that after a `HANDLER ... OPEN` statement is issued, table data can be modified (by the current session or other sessions) and these modifications might be only partially visible to `HANDLER ... NEXT` or `HANDLER ... PREV` scans.

An open handler can be closed and marked for reopen, in which case the handler loses its position in the table. This occurs when both of the following circumstances are true:

- Any session executes `FLUSH TABLES` or DDL statements on the handler's table.
- The session in which the handler is open executes non-`HANDLER` statements that use tables.

`TRUNCATE TABLE` for a table closes all handlers for the table that were opened with `HANDLER OPEN`.

If a table is flushed with `FLUSH TABLES tbl_name WITH READ LOCK` was opened with `HANDLER`, the handler is implicitly flushed and loses its position.

13.2.5. INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
{VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE
  col_name=expr
  [, col_name=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
  [INTO] tbl_name
  SET col_name={expr | DEFAULT}, ...
  [ ON DUPLICATE KEY UPDATE
    col_name=expr
    [, col_name=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
  [INTO] tbl_name [(col_name,...)]
  SELECT ...
  [ ON DUPLICATE KEY UPDATE
    col_name=expr
    [, col_name=expr] ... ]
```

INSERT inserts new rows into an existing table. The **INSERT ... VALUES** and **INSERT ... SET** forms of the statement insert rows based on explicitly specified values. The **INSERT ... SELECT** form inserts rows selected from another table or tables. **INSERT ... SELECT** is discussed further in [Section 13.2.5.1, “INSERT ... SELECT Syntax”](#).

You can use **REPLACE** instead of **INSERT** to overwrite old rows. **REPLACE** is the counterpart to **INSERT IGNORE** in the treatment of new rows that contain unique key values that duplicate old rows: The new rows are used to replace the old rows rather than being discarded. See [Section 13.2.8, “REPLACE Syntax”](#).

tbl_name is the table into which rows should be inserted. The columns for which the statement provides values can be specified as follows:

- You can provide a comma-separated list of column names following the table name. In this case, a value for each named column must be provided by the **VALUES** list or the **SELECT** statement.
- If you do not specify a list of column names for **INSERT ... VALUES** or **INSERT ... SELECT**, values for every column in the table must be provided by the **VALUES** list or the **SELECT** statement. If you do not know the order of the columns in the table, use **DESCRIBE *tbl_name*** to find out.
- The **SET** clause indicates the column names explicitly.

Column values can be given in several ways:

- If you are not running in strict SQL mode, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in [Section 11.5, “Data Type Default Values”](#). See also [Section 1.8.6.2, “Constraints on Invalid Data”](#).

If you want an **INSERT** statement to generate an error unless you explicitly specify values for all columns that do not have a default value, you should use strict mode. See [Section 5.1.7, “Server SQL Modes”](#).

- Use the keyword **DEFAULT** to set a column explicitly to its default value. This makes it easier to write **INSERT** statements that assign values to all but a few columns, because it enables you to avoid writing an incomplete **VALUES** list that does not include a value for each column in the table. Otherwise, you would have to write out the list of column names corresponding to each value in the **VALUES** list.

You can also use **DEFAULT(*col_name*)** [\[1198\]](#) as a more general form that can be used in expressions to produce a given column's default value.

- If both the column list and the **VALUES** list are empty, **INSERT** creates a row with each column set to its default value:

```
INSERT INTO tbl_name () VALUES();
```

In strict mode, an error occurs if any column doesn't have a default value. Otherwise, MySQL uses the implicit default value for any column that does not have an explicitly defined default.

- You can specify an expression *expr* to provide a column value. This might involve type conversion if the type of the expression does not match the type of the column, and conversion of a given value can result in different inserted values depending on the data type. For example, inserting the string '1999.0e-2' into an `INT`, `FLOAT`, `DECIMAL(10,6)`, or `YEAR` column results in the values 1999, 19.9921, 19.992100, and 1999 being inserted, respectively. The reason the value stored in the `INT` and `YEAR` columns is 1999 is that the string-to-integer conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the floating-point and fixed-point columns, the string-to-floating-point conversion considers the entire string a valid floating-point value.

An expression *expr* can refer to any column that was set earlier in a value list. For example, you can do this because the value for `col2` refers to `col1`, which has previously been assigned:

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

But the following is not legal, because the value for `col1` refers to `col2`, which is assigned after `col1`:

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

One exception involves columns that contain `AUTO_INCREMENT` values. Because the `AUTO_INCREMENT` value is generated after other value assignments, any reference to an `AUTO_INCREMENT` column in the assignment returns a 0.

`INSERT` statements that use `VALUES` syntax can insert multiple rows. To do this, include multiple lists of column values, each enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

The values list for each row must be enclosed within parentheses. The following statement is illegal because the number of values in the list does not match the number of column names:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

`VALUE` is a synonym for `VALUES` in this context. Neither implies anything about the number of values lists, and either may be used whether there is a single values list or multiple lists.

The affected-rows value for an `INSERT` can be obtained using the `ROW_COUNT()` [1196] function (see Section 12.14, "Information Functions"), or the `mysql_affected_rows()` C API function (see Section 22.8.3.1, "mysql_affected_rows()").

If you use an `INSERT ... VALUES` statement with multiple value lists or `INSERT ... SELECT`, the statement returns an information string in this format:

```
Records: 100 Duplicates: 0 Warnings: 0
```

`Records` indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because `Duplicates` can be nonzero.) `Duplicates` indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. For multiple-row `INSERT` statements or `INSERT INTO ... SELECT` statements, the column is set to the implicit default value for the column data type. This is 0 for numeric types, the empty string (' ') for string types, and the "zero" value for date and time types. `INSERT INTO ... SELECT` statements are handled the same way as multiple-row

inserts because the server does not examine the result set from the `SELECT` to see whether it returns a single row. (For a single-row `INSERT`, no warning occurs when `NULL` is inserted into a `NOT NULL` column. Instead, the statement fails with an error.)

- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.
- Assigning a value such as `'10.34 a'` to a numeric column. The trailing nonnumeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to `0`.
- Inserting a string into a string column (`CHAR`, `VARCHAR`, `TEXT`, or `BLOB`) that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

If you are using the C API, the information string can be obtained by invoking the `mysql_info()` function. See [Section 22.8.3.35](#), “`mysql_info()`”.

If `INSERT` inserts a row into a table that has an `AUTO_INCREMENT` column, you can find the value used for that column by using the SQL `LAST_INSERT_ID()` [1194] function. From within the C API, use the `mysql_insert_id()` function. However, you should note that the two functions do not always behave identically. The behavior of `INSERT` statements with respect to `AUTO_INCREMENT` columns is discussed further in [Section 12.14](#), “Information Functions”, and [Section 22.8.3.37](#), “`mysql_insert_id()`”.

The `INSERT` statement supports the following modifiers:

- If you use the `DELAYED` keyword, the server puts the row or rows to be inserted into a buffer, and the client issuing the `INSERT DELAYED` statement can then continue immediately. If the table is in use, the server holds the rows. When the table is free, the server begins inserting rows, checking periodically to see whether there are any new read requests for the table. If there are, the delayed row queue is suspended until the table becomes free again. See [Section 13.2.5.2](#), “`INSERT DELAYED` Syntax”.

`DELAYED` is ignored with `INSERT ... SELECT` or `INSERT ... ON DUPLICATE KEY UPDATE`.

`DELAYED` is also disregarded for an `INSERT` that uses functions accessing tables or triggers, or that is called from a function or a trigger.

- If you use the `LOW_PRIORITY` keyword, execution of the `INSERT` is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the `INSERT LOW_PRIORITY` statement is waiting. It is possible, therefore, for a client that issues an `INSERT LOW_PRIORITY` statement to wait for a very long time (or even forever) in a read-heavy environment. (This is in contrast to `INSERT DELAYED`, which lets the client continue at once. Note that `LOW_PRIORITY` should normally not be used with `MyISAM` tables because doing so disables concurrent inserts. See [Section 8.10.3](#), “Concurrent Inserts”.

If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` [452] option if the server was started with that option. It also causes concurrent inserts not to be used. See [Section 8.10.3](#), “Concurrent Inserts”.

`LOW_PRIORITY` and `HIGH_PRIORITY` affect only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- If you use the `IGNORE` keyword, errors that occur while executing the `INSERT` statement are treated as warnings instead. For example, without `IGNORE`, a row that duplicates an existing `UNIQUE` index

or **PRIMARY KEY** value in the table causes a duplicate-key error and the statement is aborted. With **IGNORE**, the row still is not inserted, but no error is issued.

IGNORE has a similar effect on inserts into partitioned tables where no partition matching a given value is found. Without **IGNORE**, such **INSERT** statements are aborted with an error; however, when **INSERT IGNORE** is used, the insert operation fails silently for the row containing the unmatched value, but any rows that are matched are inserted. For an example, see [Section 18.2.2, “LIST Partitioning”](#).

Data conversions that would trigger errors abort the statement if **IGNORE** is not specified. With **IGNORE**, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the `mysql_info()` C API function how many rows were actually inserted into the table.

- If you specify **ON DUPLICATE KEY UPDATE**, and a row is inserted that would cause a duplicate value in a **UNIQUE** index or **PRIMARY KEY**, an **UPDATE** of the old row is performed. The affected-rows value per row is 1 if the row is inserted as a new row and 2 if an existing row is updated. See [Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

Inserting into a table requires the **INSERT [696]** privilege for the table. If the **ON DUPLICATE KEY UPDATE** clause is used and a duplicate key causes an **UPDATE** to be performed instead, the statement requires the **UPDATE [697]** privilege for the columns to be updated. For columns that are read but not modified you need only the **SELECT [697]** privilege (such as for a column referenced only on the right hand side of an `col_name=expr` assignment in an **ON DUPLICATE KEY UPDATE** clause).

An **INSERT** statement that acts on a partitioned table using a storage engine such as **MyISAM** that employs table-level locks locks all partitions of the table. This does not occur with tables using storage engines such as **InnoDB** that employ row-level locking. This issue is resolved in MySQL 5.6. See [Section 18.5.4, “Partitioning and Table-Level Locking”](#), for more information.

13.2.5.1. INSERT ... SELECT Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
 [INTO] tbl_name [(col_name,...)]
SELECT ...
 [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

With **INSERT ... SELECT**, you can quickly insert many rows into a table from one or many tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
SELECT tbl_temp1.fld_order_id
FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

The following conditions hold for a **INSERT ... SELECT** statements:

- Specify **IGNORE** to ignore rows that would cause duplicate-key violations.
- **DELAYED** is ignored with **INSERT ... SELECT**.
- The target table of the **INSERT** statement may appear in the **FROM** clause of the **SELECT** part of the query. (This was not possible in some older versions of MySQL.) However, you cannot insert into a table and select from the same table in a subquery.

When selecting from and inserting into a table at the same time, MySQL creates a temporary table to hold the rows from the **SELECT** and then inserts those rows into the target table. However, it remains true that you cannot use **INSERT INTO t ... SELECT ... FROM t** when **t** is a **TEMPORARY** table, because **TEMPORARY** tables cannot be referred to twice in the same statement (see [Section C.5.7.2, “TEMPORARY Table Problems”](#)).

- `AUTO_INCREMENT` columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts for `INSERT ... SELECT` statements.
- To avoid ambiguous column reference problems when the `SELECT` and the `INSERT` refer to the same table, provide a unique alias for each table used in the `SELECT` part, and qualify column names in that part with the appropriate alias.

In the values part of `ON DUPLICATE KEY UPDATE`, you can refer to columns in other tables, as long as you do not use `GROUP BY` in the `SELECT` part. One side effect is that you must qualify nonunique column names in the values part.

The order in which rows are returned by a `SELECT` statement with no `ORDER BY` clause is not determined. This means that, when using replication, there is no guarantee that such a `SELECT` returns rows in the same order on the master and the slave; this can lead to inconsistencies between them. To prevent this from occurring, you should always write `INSERT ... SELECT` statements that are to be replicated as `INSERT ... SELECT ... ORDER BY column`. The choice of `column` does not matter as long as the same order for returning the rows is enforced on both the master and the slave. See also [Section 16.4.1.14, “Replication and LIMIT”](#).

Due to this issue, beginning with MySQL 5.5.18, `INSERT ... SELECT ON DUPLICATE KEY UPDATE` and `INSERT IGNORE ... SELECT` statements are flagged as unsafe for statement-based replication. With this change, such statements produce a warning in the log when using statement-based mode and are logged using the row-based format when using `MIXED` mode. (Bug #11758262, Bug #50439)

See also [Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”](#).

An `INSERT ... SELECT` statement that acts on partitioned tables using a storage engine such as `MyISAM` that employs table-level locks locks all partitions of the source and target tables. This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking. This issue is resolved in MySQL 5.6. See [Section 18.5.4, “Partitioning and Table-Level Locking”](#), for more information.

13.2.5.2. INSERT DELAYED Syntax

```
INSERT DELAYED ...
```

The `DELAYED` option for the `INSERT` statement is a MySQL extension to standard SQL that is very useful if you have clients that cannot or need not wait for the `INSERT` to complete. This is a common situation when you use MySQL for logging and you also periodically run `SELECT` and `UPDATE` statements that take a long time to complete.

When a client uses `INSERT DELAYED`, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Another major benefit of using `INSERT DELAYED` is that inserts from many clients are bundled together and written in one block. This is much faster than performing many separate inserts.

Note that `INSERT DELAYED` is slower than a normal `INSERT` if the table is not otherwise in use. There is also the additional overhead for the server to handle a separate thread for each table for which there are delayed rows. This means that you should use `INSERT DELAYED` only when you are really sure that you need it.

The queued rows are held only in memory until they are inserted into the table. This means that if you terminate `mysqld` forcibly (for example, with `kill -9`) or if `mysqld` dies unexpectedly, *any queued rows that have not been written to disk are lost*.

There are some constraints on the use of `DELAYED`:

- `INSERT DELAYED` works only with `MyISAM`, `MEMORY`, `ARCHIVE`, and `BLACKHOLE` tables. For engines that do not support `DELAYED`, an error occurs.
- An error occurs for `INSERT DELAYED` if used with a table that has been locked with `LOCK TABLES` because the insert must be handled by a separate thread, not by the session that holds the lock.
- For `MyISAM` tables, if there are no free blocks in the middle of the data file, concurrent `SELECT` and `INSERT` statements are supported. Under these circumstances, you very seldom need to use `INSERT DELAYED` with `MyISAM`.
- `INSERT DELAYED` should be used only for `INSERT` statements that specify value lists. The server ignores `DELAYED` for `INSERT ... SELECT` or `INSERT ... ON DUPLICATE KEY UPDATE` statements.
- Because the `INSERT DELAYED` statement returns immediately, before the rows are inserted, you cannot use `LAST_INSERT_ID()` [1194] to get the `AUTO_INCREMENT` value that the statement might generate.
- `DELAYED` rows are not visible to `SELECT` statements until they actually have been inserted.
- Prior to MySQL 5.5.7, `INSERT DELAYED` was treated as a normal `INSERT` if the statement inserted multiple rows, binary logging was enabled, and the global logging format was statement-based (that is, whenever `binlog_format` [1899] was set to `STATEMENT`). Beginning with MySQL 5.5.7, `INSERT DELAYED` is always handled as a simple `INSERT` (that is, without the `DELAYED` option) whenever the value of `binlog_format` [1899] is `STATEMENT` or `MIXED`. (In the latter case, the statement no longer triggers a switch to row-based logging, and so is logged using the statement-based format.)

This does not apply when using row-based binary logging mode (`binlog_format` [1899] set to `ROW`), in which `INSERT DELAYED` statements are always executed using the `DELAYED` option as specified, and logged as row-update events.

- `DELAYED` is ignored on slave replication servers, so that `INSERT DELAYED` is treated as a normal `INSERT` on slaves. This is because `DELAYED` could cause the slave to have different data than the master.
- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.
- `INSERT DELAYED` is not supported for views.
- `INSERT DELAYED` is not supported for partitioned tables.

The following describes in detail what happens when you use the `DELAYED` option to `INSERT` or `REPLACE`. In this description, the “thread” is the thread that received an `INSERT DELAYED` statement and “handler” is the thread that handles all `INSERT DELAYED` statements for a particular table.

- When a thread executes a `DELAYED` statement for a table, a handler thread is created to process all `DELAYED` statements for the table, if no such handler already exists.
- The thread checks whether the handler has previously acquired a `DELAYED` lock; if not, it tells the handler thread to do so. The `DELAYED` lock can be obtained even if other threads have a `READ` or `WRITE` lock on the table. However, the handler waits for all `ALTER TABLE` locks or `FLUSH TABLES` statements to finish, to ensure that the table structure is up to date.
- The thread executes the `INSERT` statement, but instead of writing the row to the table, it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported to the client program.

- The client cannot obtain from the server the number of duplicate rows or the `AUTO_INCREMENT` value for the resulting row, because the `INSERT` returns before the insert operation has been completed. (If you use the C API, the `mysql_info()` function does not return anything meaningful, for the same reason.)
- The binary log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the binary log is updated when the first row is inserted.
- Each time that `delayed_insert_limit` [495] rows are written, the handler checks whether any `SELECT` statements are still pending. If so, it permits these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new `INSERT DELAYED` statements are received within `delayed_insert_timeout` [496] seconds, the handler terminates.
- If more than `delayed_queue_size` [496] rows are pending in a specific handler queue, the thread requesting `INSERT DELAYED` waits until there is room in the queue. This is done to ensure that `mysqld` does not use all memory for the delayed memory queue.
- The handler thread shows up in the MySQL process list with `delayed_insert` in the `Command` column. It is killed if you execute a `FLUSH TABLES` statement or kill it with `KILL thread_id`. However, before exiting, it first stores all queued rows into the table. During this time it does not accept any new `INSERT` statements from other threads. If you execute an `INSERT DELAYED` statement after this, a new handler thread is created.

Note that this means that `INSERT DELAYED` statements have higher priority than normal `INSERT` statements if there is an `INSERT DELAYED` handler running. Other update statements have to wait until the `INSERT DELAYED` queue is empty, someone terminates the handler thread (with `KILL thread_id`), or someone executes a `FLUSH TABLES`.

- The following status variables provide information about `INSERT DELAYED` statements.

Status Variable	Meaning
<code>Delayed_insert_threads</code> [598]	Number of handler threads
<code>Delayed_writes</code> [598]	Number of rows written with <code>INSERT DELAYED</code>
<code>Not_flushed_delayed_rows</code> [604]	Number of rows waiting to be written

You can view these variables by issuing a `SHOW STATUS` statement or by executing a `mysqladmin extended-status` command.

13.2.5.3. INSERT ... ON DUPLICATE KEY UPDATE Syntax

If you specify `ON DUPLICATE KEY UPDATE`, and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, MySQL performs an `UPDATE` of the old row. For example, if column `a` is declared as `UNIQUE` and contains the value `1`, the following two statements have similar effect:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=c+1;

UPDATE table SET c=c+1 WHERE a=1;
```

(The effects are not identical for an `InnoDB` table where `a` is an auto-increment column. With an auto-increment column, an `INSERT` statement increases the auto-increment value but `UPDATE` does not.)

The `ON DUPLICATE KEY UPDATE` clause can contain multiple column assignments, separated by commas.

With `ON DUPLICATE KEY UPDATE`, the affected-rows value per row is 1 if the row is inserted as a new row, and 2 if an existing row is updated.

If column `b` is also unique, the `INSERT` is equivalent to this `UPDATE` statement instead:

```
UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

If `a=1 OR b=2` matches several rows, only *one* row is updated. In general, you should try to avoid using an `ON DUPLICATE KEY UPDATE` clause on tables with multiple unique indexes.

You can use the `VALUES(col_name)` [1202] function in the `UPDATE` clause to refer to column values from the `INSERT ... ON DUPLICATE KEY UPDATE` statement. In other words, `VALUES(col_name)` [1202] in the `ON DUPLICATE KEY UPDATE` clause refers to the value of *col_name* that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` [1202] function is meaningful only in `INSERT ... UPDATE` statements and returns `NULL` otherwise. Example:

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=3;
INSERT INTO table (a,b,c) VALUES (4,5,6)
ON DUPLICATE KEY UPDATE c=9;
```

If a table contains an `AUTO_INCREMENT` column and `INSERT ... ON DUPLICATE KEY UPDATE` inserts or updates a row, the `LAST_INSERT_ID()` [1194] function returns the `AUTO_INCREMENT` value.

The `DELAYED` option is ignored when you use `ON DUPLICATE KEY UPDATE`.

Because the results of `INSERT ... SELECT` statements depend on the ordering of rows from the `SELECT` and this order cannot always be guaranteed, it is possible when logging `INSERT ... SELECT ON DUPLICATE KEY UPDATE` statements for the master and the slave to diverge. Thus, in MySQL 5.5.18 and later, `INSERT ... SELECT ON DUPLICATE KEY UPDATE` statements are flagged as unsafe for statement-based replication. With this change, such statements produce a warning in the log when using statement-based mode and are logged using the row-based format when using `MIXED` mode. In addition, beginning with MySQL 5.5.24, an `INSERT ... ON DUPLICATE KEY UPDATE` statement against a table having more than one unique or primary key is also marked as unsafe. (Bug #11765650, Bug #58637) See also Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

An `INSERT ... ON DUPLICATE KEY UPDATE` on a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks all partitions of the table. This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking. This issue is resolved in MySQL 5.6. See Section 18.5.4, “Partitioning and Table-Level Locking”, for more information.

13.2.6. LOAD DATA INFILE Syntax

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[CHARACTER SET charset_name]
[{FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
```

```
[IGNORE number {LINES | ROWS}]
[(col_name_or_user_var,...)]
[SET col_name = expr,...]
```

The `LOAD DATA INFILE` statement reads rows from a text file into a table at a very high speed. The file name must be given as a literal string.

`LOAD DATA INFILE` is the complement of `SELECT ... INTO OUTFILE`. (See [Section 13.2.9.1, “SELECT ... INTO Syntax”](#).) To write data from a table to a file, use `SELECT ... INTO OUTFILE`. To read the file back into a table, use `LOAD DATA INFILE`. The syntax of the `FIELDS` and `LINES` clauses is the same for both statements. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

For more information about the efficiency of `INSERT` versus `LOAD DATA INFILE` and speeding up `LOAD DATA INFILE`, see [Section 8.2.2.1, “Speed of INSERT Statements”](#).

The character set indicated by the `character_set_database` [\[487\]](#) system variable is used to interpret the information in the file. `SET NAMES` and the setting of `character_set_client` [\[486\]](#) do not affect interpretation of input. If the contents of the input file use a character set that differs from the default, it is usually preferable to specify the character set of the file by using the `CHARACTER SET` clause. A character set of `binary` specifies “no conversion.”

`LOAD DATA INFILE` interprets all fields in the file as having the same character set, regardless of the data types of the columns into which field values are loaded. For proper interpretation of file contents, you must ensure that it was written with the correct character set. For example, if you write a data file with `mysqldump -T` or by issuing a `SELECT ... INTO OUTFILE` statement in `mysql`, be sure to use a `--default-character-set` option with `mysqldump` or `mysql` so that output is written in the character set to be used when the file is loaded with `LOAD DATA INFILE`.



Note

It is not possible to load data files that use the `ucs2`, `utf16`, or `utf32` character set.

The `character_set_filesystem` [\[487\]](#) system variable controls the interpretation of the file name.

You can also load data files by using the `mysqlimport` utility; it operates by sending a `LOAD DATA INFILE` statement to the server. The `--local` [\[328\]](#) option causes `mysqlimport` to read data files from the client host. You can specify the `--compress` [\[327\]](#) option to get better performance over slow networks if the client and server support the compressed protocol. See [Section 4.5.5, “mysqlimport — A Data Import Program”](#).

If you use `LOW_PRIORITY`, execution of the `LOAD DATA` statement is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

If you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while `LOAD DATA` is executing. Using this option affects the performance of `LOAD DATA` a bit, even if no other thread is using the table at the same time.

Prior to MySQL 5.5.1, `CONCURRENT` was not replicated when using statement-based replication (see [Bug #34628](#)). However, it is replicated when using row-based replication, regardless of the version. See [Section 16.4.1.15, “Replication and LOAD DATA INFILE”](#), for more information.

The `LOCAL` keyword, if specified, is interpreted with respect to the client end of the connection:

- If `LOCAL` is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full path name to specify its exact location. If given as a relative path name, the name is interpreted relative to the directory in which the client program was started.

When using `LOCAL` with `LOAD DATA`, a copy of the file is created in the server's temporary directory. This is *not* the directory determined by the value of `tmpdir` [568] or `slave_load_tmpdir` [1887], but rather the operating system's temporary directory, and is not configurable in the MySQL Server. (Typically the system temporary directory is `/tmp` on Linux systems and `C:\WINDOWS\TEMP` on Windows.) Lack of sufficient space for the copy in this directory can cause the `LOAD DATA LOCAL` statement to fail.

- If `LOCAL` is not specified, the file must be located on the server host and is read directly by the server. The server uses the following rules to locate the file:
 - If the file name is an absolute path name, the server uses it as given.
 - If the file name is a relative path name with one or more leading components, the server searches for the file relative to the server's data directory.
 - If a file name with no leading components is given, the server looks for the file in the database directory of the default database.

Note that, in the non-`LOCAL` case, these rules mean that a file named as `./myfile.txt` is read from the server's data directory, whereas the file named as `myfile.txt` is read from the database directory of the default database. For example, if `db1` is the default database, the following `LOAD DATA` statement reads the file `data.txt` from the database directory for `db1`, even though the statement explicitly loads the file into a table in the `db2` database:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Windows path names are specified using forward slashes rather than backslashes. If you do use backslashes, you must double them.

For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by all. Also, to use `LOAD DATA INFILE` on server files, you must have the `FILE` [696] privilege. See [Section 6.2.1, "Privileges Provided by MySQL"](#). For non-`LOCAL` load operations, if the `secure_file_priv` [550] system variable is set to a nonempty directory name, the file to be loaded must be located in that directory.

Using `LOCAL` is a bit slower than letting the server access the files directly, because the contents of the file must be sent over the connection by the client to the server. On the other hand, you do not need the `FILE` [696] privilege to load local files.

With `LOCAL`, the default duplicate-key handling behavior is the same as if `IGNORE` is specified; this is because the server has no way to stop transmission of the file in the middle of the operation. `IGNORE` is explained further later in this section.

`LOCAL` works only if your server and your client both have been configured to permit it. For example, if `mysqld` was started with `--local-infile=0` [513], `LOCAL` does not work. See [Section 6.1.6, "Security Issues with LOAD DATA LOCAL"](#).

On Unix, if you need `LOAD DATA` to read from a pipe, you can use the following technique (the example loads a listing of the `/` directory into the table `db1.t1`):

```
mkfifo /mysql/data/db1/ls.dat
chmod 666 /mysql/data/db1/ls.dat
find / -ls > /mysql/data/db1/ls.dat &
mysql -e "LOAD DATA INFILE 'ls.dat' INTO TABLE t1" db1
```

Note that you must run the command that generates the data to be loaded and the `mysql` commands either on separate terminals, or run the data generation process in the background (as shown in the preceding example). If you do not do this, the pipe will block until data is read by the `mysql` process.

The `REPLACE` and `IGNORE` keywords control handling of input rows that duplicate existing rows on unique key values:

- If you specify `REPLACE`, input rows replace existing rows. In other words, rows that have the same value for a primary key or unique index as an existing row. See [Section 13.2.8, “REPLACE Syntax”](#).
- If you specify `IGNORE`, input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, the behavior depends on whether the `LOCAL` keyword is specified. Without `LOCAL`, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With `LOCAL`, the default behavior is the same as if `IGNORE` is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

If you want to ignore foreign key constraints during the load operation, you can issue a `SET foreign_key_checks = 0` statement before executing `LOAD DATA`.

If you use `LOAD DATA INFILE` on an empty `MyISAM` table, all nonunique indexes are created in a separate batch (as for `REPAIR TABLE`). Normally, this makes `LOAD DATA INFILE` much faster when you have many indexes. In some extreme cases, you can create the indexes even faster by turning them off with `ALTER TABLE ... DISABLE KEYS` before loading the file into the table and using `ALTER TABLE ... ENABLE KEYS` to re-create the indexes after loading the file. See [Section 8.2.2.1, “Speed of INSERT Statements”](#).

For both the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements, the syntax of the `FIELDS` and `LINES` clauses is the same. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

If you specify a `FIELDS` clause, each of its subclasses (`TERMINATED BY`, `[OPTIONALLY] ENCLOSED BY`, and `ESCAPED BY`) is also optional, except that you must specify at least one of them.

If you specify no `FIELDS` or `LINES` clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'  
LINES TERMINATED BY '\n' STARTING BY ''
```

(Backslash is the MySQL escape character within strings in SQL statements, so to specify a literal backslash, you must specify two backslashes for the value to be interpreted as a single backslash. The escape sequences `'\t'` and `'\n'` specify tab and newline characters, respectively.)

In other words, the defaults cause `LOAD DATA INFILE` to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip over any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret characters preceded by the escape character “\” as escape sequences. For example, `“\t”`, `“\n”`, and `“\”` signify tab, newline, and backslash, respectively. See the discussion of `FIELDS ESCAPED BY` later for the full list of escape sequences.

Conversely, the defaults cause `SELECT ... INTO OUTFILE` to act as follows when writing output:

- Write tabs between fields.

- Do not enclose fields within any quoting characters.
- Use “\” to escape instances of tab, newline, or “\” that occur within field values.
- Write newlines at the ends of lines.

**Note**

If you have generated the text file on a Windows system, you might have to use `LINES TERMINATED BY '\r\n'` to read the file properly, because Windows programs typically use two characters as a line terminator. Some programs, such as `WordPad`, might use `\r` as a line terminator when writing files. To read such files, use `LINES TERMINATED BY '\r'`.

If all the lines you want to read in have a common prefix that you want to ignore, you can use `LINES STARTING BY 'prefix_string'` to skip over the prefix, *and anything before it*. If a line does not include the prefix, the entire line is skipped. Suppose that you issue the following statement:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
  FIELDS TERMINATED BY ',' LINES STARTING BY 'xxx';
```

If the data file looks like this:

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

The resulting rows will be (`"abc",1`) and (`"def",2`). The third row in the file is skipped because it does not contain the prefix.

The `IGNORE number LINES` option can be used to ignore lines at the start of the file. For example, you can use `IGNORE 1 LINES` to skip over an initial header line containing column names:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA INFILE` to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, `LOAD DATA INFILE` will not interpret the contents of the file properly. Suppose that you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
SELECT * INTO OUTFILE 'data.txt'
  FIELDS TERMINATED BY ','
  FROM table2;
```

To read the comma-delimited file back in, the correct statement would be:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown following, it wouldn't work because it instructs `LOAD DATA INFILE` to look for tabs between fields:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

`LOAD DATA INFILE` can be used to read files obtained from external sources. For example, many programs can export data in comma-separated values (CSV) format, such that lines have fields separated

by commas and enclosed within double quotation marks, with an initial line of column names. If the lines in such a file are terminated by carriage return/newline pairs, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

If the input values are not necessarily enclosed within quotation marks, use `OPTIONALLY` before the `ENCLOSED BY` keywords.

Any of the field- or line-handling options can specify an empty string (' '). If not empty, the `FIELDS [OPTIONALLY] ENCLOSED BY` and `FIELDS ESCAPED BY` values must be a single character. The `FIELDS TERMINATED BY`, `LINES STARTING BY`, and `LINES TERMINATED BY` values can be more than one character. For example, to write lines that are terminated by carriage return/linefeed pairs, or to read a file containing such lines, specify a `LINES TERMINATED BY '\r\n'` clause.

To read a file containing jokes that are separated by lines consisting of `%%`, you can do this

```
CREATE TABLE jokes
  (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
  FIELDS TERMINATED BY ' '
  LINES TERMINATED BY '\n%%\n' (joke);
```

`FIELDS [OPTIONALLY] ENCLOSED BY` controls quoting of fields. For output (`SELECT ... INTO OUTFILE`), if you omit the word `OPTIONALLY`, all fields are enclosed by the `ENCLOSED BY` character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string",100.20"
"2","a string containing a , comma",102.20"
"3","a string containing a \" quote",102.20"
"4","a string containing a \", quote and comma",102.20"
```

If you specify `OPTIONALLY`, the `ENCLOSED BY` character is used only to enclose values from columns that have a string data type (such as `CHAR`, `BINARY`, `TEXT`, or `ENUM`):

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Note that occurrences of the `ENCLOSED BY` character within a field value are escaped by prefixing them with the `ESCAPED BY` character. Also note that if you specify an empty `ESCAPED BY` value, it is possible to inadvertently generate output that cannot be read properly by `LOAD DATA INFILE`. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the `ENCLOSED BY` character, if present, is stripped from the ends of field values. (This is true regardless of whether `OPTIONALLY` is specified; `OPTIONALLY` has no effect on input interpretation.) Occurrences of the `ENCLOSED BY` character preceded by the `ESCAPED BY` character are interpreted as part of the current field value.

If the field begins with the `ENCLOSED BY` character, instances of that character are recognized as terminating a field value only if followed by the field or line `TERMINATED BY` sequence. To avoid ambiguity, occurrences of the `ENCLOSED BY` character within a field value can be doubled and are interpreted as a single instance of the character. For example, if `ENCLOSED BY ' '` is specified, quotation marks are handled as shown here:

```
"The ""BIG"" boss" -> The "BIG" boss
The "BIG" boss     -> The "BIG" boss
The ""BIG"" boss  -> The ""BIG"" boss
```

`FIELDS ESCAPED BY` controls how to read or write special characters:

- For input, if the `FIELDS ESCAPED BY` character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. Some two-character sequences that are exceptions, where the first character is the escape character. These sequences are shown in the following table (using “\” for the escape character). The rules for `NULL` handling are described later in this section.

Character	Escape Sequence
\0	An ASCII NUL (0x00) character
\b	A backspace character
\n	A newline (linefeed) character
\r	A carriage return character
\t	A tab character.
\z	ASCII 26 (Control+Z)
\N	NULL

For more information about “\”-escape syntax, see [Section 9.1.1, “String Literals”](#).

If the `FIELDS ESCAPED BY` character is empty, escape-sequence interpretation does not occur.

- For output, if the `FIELDS ESCAPED BY` character is not empty, it is used to prefix the following characters on output:
 - The `FIELDS ESCAPED BY` character
 - The `FIELDS [OPTIONALLY] ENCLOSED BY` character
 - The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
 - ASCII 0 (what is actually written following the escape character is ASCII “0”, not a zero-valued byte)

If the `FIELDS ESCAPED BY` character is empty, no characters are escaped and `NULL` is output as `NULL`, not `\N`. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

In certain cases, field- and line-handling options interact:

- If `LINES TERMINATED BY` is an empty string and `FIELDS TERMINATED BY` is nonempty, lines are also terminated with `FIELDS TERMINATED BY`.
- If the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values are both empty (' '), a fixed-row (nondelimited) format is used. With fixed-row format, no delimiters are used between fields (but you can still have a line terminator). Instead, column values are read and written using a field width wide enough

to hold all values in the field. For `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, and `BIGINT`, the field widths are 4, 6, 8, 11, and 20, respectively, no matter what the declared display width is.

`LINES TERMINATED BY` is still used to separate lines. If a line does not contain all fields, the rest of the columns are set to their default values. If you do not have a line terminator, you should set this to `' '`. In this case, the text file must contain all fields for each row.

Fixed-row format also affects handling of `NULL` values, as described later. Note that fixed-size format does not work if you are using a multi-byte character set.

Handling of `NULL` values varies according to the `FIELDS` and `LINES` options in use:

- For the default `FIELDS` and `LINES` values, `NULL` is written as a field value of `\N` for output, and a field value of `\N` is read as `NULL` for input (assuming that the `ESCAPED BY` character is `"\"`).
- If `FIELDS ENCLOSED BY` is not empty, a field containing the literal word `NULL` as its value is read as a `NULL` value. This differs from the word `NULL` enclosed within `FIELDS ENCLOSED BY` characters, which is read as the string `'NULL'`.
- If `FIELDS ESCAPED BY` is empty, `NULL` is written as the word `NULL`.
- With fixed-row format (which is used when `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` are both empty), `NULL` is written as an empty string. Note that this causes both `NULL` values and empty strings in the table to be indistinguishable when written to the file because both are written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

An attempt to load `NULL` into a `NOT NULL` column causes assignment of the implicit default value for the column's data type and a warning, or an error in strict SQL mode. Implicit default values are discussed in [Section 11.5, "Data Type Default Values"](#).

Some cases are not supported by `LOAD DATA INFILE`:

- Fixed-size rows (`FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` both empty) and `BLOB` or `TEXT` columns.
- If you specify one separator that is the same as or a prefix of another, `LOAD DATA INFILE` cannot interpret the input properly. For example, the following `FIELDS` clause would cause problems:

```
FIELDS TERMINATED BY '' ENCLOSED BY ''
```

- If `FIELDS ESCAPED BY` is empty, a field value that contains an occurrence of `FIELDS ENCLOSED BY` or `LINES TERMINATED BY` followed by the `FIELDS TERMINATED BY` value causes `LOAD DATA INFILE` to stop reading a field or line too early. This happens because `LOAD DATA INFILE` cannot properly determine where the field or line value ends.

The following example loads all columns of the `persondata` table:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

By default, when no column list is provided at the end of the `LOAD DATA INFILE` statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata (col1,col2,...);
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match input fields with table columns.

The column list can contain either column names or user variables. With user variables, the `SET` clause enables you to perform transformations on their values before assigning the result to columns.

User variables in the `SET` clause can be used in several ways. The following example uses the first input column directly for the value of `t1.column1`, and assigns the second input column to a user variable that is subjected to a division operation before being used for the value of `t1.column2`:

```
LOAD DATA INFILE 'file.txt'
INTO TABLE t1
(column1, @var1)
SET column2 = @var1/100;
```

The `SET` clause can be used to supply values not derived from the input file. The following statement sets `column3` to the current date and time:

```
LOAD DATA INFILE 'file.txt'
INTO TABLE t1
(column1, column2)
SET column3 = CURRENT_TIMESTAMP;
```

You can also discard an input value by assigning it to a user variable and not assigning the variable to a table column:

```
LOAD DATA INFILE 'file.txt'
INTO TABLE t1
(column1, @dummy, column2, @dummy, column3);
```

Use of the column/variable list and `SET` clause is subject to the following restrictions:

- Assignments in the `SET` clause should have only column names on the left hand side of assignment operators.
- You can use subqueries in the right hand side of `SET` assignments. A subquery that returns a value to be assigned to a column may be a scalar subquery only. Also, you cannot use a subquery to select from the table that is being loaded.
- Lines ignored by an `IGNORE` clause are not processed for the column/variable list or `SET` clause.
- User variables cannot be used when loading data with fixed-row format because user variables do not have a display width.

When processing an input line, `LOAD DATA` splits it into fields and uses the values according to the column/variable list and the `SET` clause, if they are present. Then the resulting row is inserted into the table. If there are `BEFORE INSERT` or `AFTER INSERT` triggers for the table, they are activated before or after inserting the row, respectively.

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in [Section 11.5, “Data Type Default Values”](#).

An empty field value is interpreted differently than if the field value is missing:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to `0`.
- For date and time types, the column is set to the appropriate “zero” value for the type. See [Section 11.3, “Date and Time Types”](#).

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an `INSERT` or `UPDATE` statement.

`TIMESTAMP` columns are set to the current date and time only if there is a `NULL` value for the column (that is, `\N`) and the column is not declared to permit `NULL` values, or if the `TIMESTAMP` column's default value is the current timestamp and it is omitted from the field list when a field list is specified.

`LOAD DATA INFILE` regards all input as strings, so you cannot use numeric values for `ENUM` or `SET` columns the way you can with `INSERT` statements. All `ENUM` and `SET` values must be specified as strings.

`BIT` values cannot be loaded using binary notation (for example, `b'011010'`). To work around this, specify the values as regular integers and use the `SET` clause to convert them so that MySQL performs a numeric type conversion and loads them into the `BIT` column properly:

```
shell> cat /tmp/bit_test.txt
2
127
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
      -> INTO TABLE bit_test (@var1) SET b= CAST(@var1 AS UNSIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Deleted: 0 Skipped: 0 Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+-----+
| bin(b+0) |
+-----+
| 10       |
| 1111111 |
+-----+
2 rows in set (0.00 sec)
```

When the `LOAD DATA INFILE` statement finishes, it returns an information string in the following format:

```
Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
```

If you are using the C API, you can get information about the statement by calling the `mysql_info()` function. See [Section 22.8.3.35](#), “`mysql_info()`”.

Warnings occur under the same circumstances as when values are inserted using the `INSERT` statement (see [Section 13.2.5](#), “`INSERT Syntax`”), except that `LOAD DATA INFILE` also generates warnings when there are too few or too many fields in the input row. The warnings are not stored anywhere; the number of warnings can be used only as an indication of whether everything went well.

You can use `SHOW WARNINGS` to get a list of the first `max_error_count` [520] warnings as information about what went wrong. See [Section 13.7.5.41](#), “`SHOW WARNINGS Syntax`”.

For partitioned tables using storage engines that employ table locks, such as `MyISAM`, any locks caused by `LOAD DATA` perform locks on all partitions of the table. This does not apply to tables using storage engines which employ row-level locking, such as `InnoDB`. For more information, see [Section 18.5.4](#), “`Partitioning and Table-Level Locking`”.

13.2.7. LOAD XML Syntax

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
  [REPLACE | IGNORE]
  INTO TABLE [db_name.]tbl_name
  [CHARACTER SET charset_name]
  [ROWS IDENTIFIED BY '<tagname>']
```

```
[IGNORE number {LINES | ROWS}]
[(column_or_user_var,...)]
[SET col_name = expr,...]
```

The `LOAD XML` statement reads data from an XML file into a table. The *file_name* must be given as a literal string. The *tagname* in the optional `ROWS IDENTIFIED BY` clause must also be given as a literal string, and must be surrounded by angle brackets (< and >).

`LOAD XML` acts as the complement of running the `mysql` client in XML output mode (that is, starting the client with the `--xml [282]` option). To write data from a table to an XML file, use a command such as the following one from the system shell:

```
shell> mysql --xml -e 'SELECT * FROM mytable' > file.xml
```

To read the file back into a table, use `LOAD XML INFILE`. By default, the `<row>` element is considered to be the equivalent of a database table row; this can be changed using the `ROWS IDENTIFIED BY` clause.

This statement supports three different XML formats:

- Column names as attributes and column values as attribute values:

```
<row column1="value1" column2="value2" .../>
```

- Column names as tags and column values as the content of these tags:

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

- Column names are the *name* attributes of `<field>` tags, and values are the contents of these tags:

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

This is the format used by other MySQL tools, such as `mysqldump`.

All 3 formats can be used in the same XML file; the import routine automatically detects the format for each row and interprets it correctly. Tags are matched based on the tag or attribute name and the column name.

The following clauses work essentially the same way for `LOAD XML` as they do for `LOAD DATA`:

- `LOW_PRIORITY` or `CONCURRENT`
- `LOCAL`
- `REPLACE` or `IGNORE`
- `CHARACTER SET`
- (*column_or_user_var*,...)
- `SET`

See [Section 13.2.6, “LOAD DATA INFILE Syntax”](#), for more information about these clauses.

The `IGNORE number LINES` or `IGNORE number ROWS` clause causes the first *number* rows in the XML file to be skipped. It is analogous to the `LOAD DATA` statement's `IGNORE ... LINES` clause.

To illustrate how this statement is used, suppose that we have a table created as follows:

```
USE test;

CREATE TABLE person (
  person_id INT NOT NULL PRIMARY KEY,
  fname VARCHAR(40) NULL,
  lname VARCHAR(40) NULL,
  created TIMESTAMP
);
```

Suppose further that this table is initially empty.

Now suppose that we have a simple XML file `person.xml`, whose contents are as shown here:

```
<?xml version="1.0"?>
<list>
  <person person_id="1" fname="Pekka" lname="Nousiainen"/>
  <person person_id="2" fname="Jonas" lname="Oreland"/>
  <person person_id="3"><fname>Mikael</fname><lname>Ronström</lname></person>
  <person person_id="4"><fname>Lars</fname><lname>Thalmann</lname></person>
  <person><field name="person_id">5</field><field name="fname">Tomas</field><field name="lname">Ulin</field>
  <person><field name="person_id">6</field><field name="fname">Martin</field><field name="lname">Sköld</field>
</list>
```

Each of the permissible XML formats discussed previously is represented in this example file.

To import the data in `person.xml` into the `person` table, you can use this statement:

```
mysql> LOAD XML LOCAL INFILE 'person.xml'
-> INTO TABLE person
-> ROWS IDENTIFIED BY '<person>';

Query OK, 6 rows affected (0.00 sec)
Records: 6 Deleted: 0 Skipped: 0 Warnings: 0
```

Here, we assume that `person.xml` is located in the MySQL data directory. If the file cannot be found, the following error results:

```
ERROR 2 (HY000): File '/person.xml' not found (Errcode: 2)
```

The `ROWS IDENTIFIED BY '<person>'` clause means that each `<person>` element in the XML file is considered equivalent to a row in the table into which the data is to be imported. In this case, this is the `person` table in the `test` database.

As can be seen by the response from the server, 6 rows were imported into the `test.person` table. This can be verified by a simple `SELECT` statement:

```
mysql> SELECT * FROM person;
+-----+-----+-----+-----+
| person_id | fname | lname | created |
+-----+-----+-----+-----+
| 1 | Pekka | Nousiainen | 2007-07-13 16:18:47 |
| 2 | Jonas | Oreland | 2007-07-13 16:18:47 |
| 3 | Mikael | Ronström | 2007-07-13 16:18:47 |
| 4 | Lars | Thalmann | 2007-07-13 16:18:47 |
| 5 | Tomas | Ulin | 2007-07-13 16:18:47 |
| 6 | Martin | Sköld | 2007-07-13 16:18:47 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

This shows, as stated earlier in this section, that any or all of the 3 permitted XML formats may appear in a single file and be read in using `LOAD XML`.

The inverse of the above operation—that is, dumping MySQL table data into an XML file—can be accomplished using the `mysql` client from the system shell, as shown here:



Note

The `--xml` [282] option causes the `mysql` client to use XML formatting for its output; the `-e` option causes the client to execute the SQL statement immediately following the option.

```
shell> mysql --xml -e "SELECT * FROM test.person" > person-dump.xml
shell> cat person-dump.xml
<?xml version="1.0"?>

<resultset statement="SELECT * FROM test.person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="person_id">1</field>
    <field name="fname">Pekka</field>
    <field name="lname">Nousiainen</field>
    <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
    <field name="person_id">2</field>
    <field name="fname">Jonas</field>
    <field name="lname">Oreland</field>
    <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
    <field name="person_id">3</field>
    <field name="fname">Mikael</field>
    <field name="lname">Ronström</field>
    <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
    <field name="person_id">4</field>
    <field name="fname">Lars</field>
    <field name="lname">Thalmann</field>
    <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
    <field name="person_id">5</field>
    <field name="fname">Tomas</field>
    <field name="lname">Ulin</field>
    <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
    <field name="person_id">6</field>
    <field name="fname">Martin</field>
    <field name="lname">Sköld</field>
    <field name="created">2007-07-13 16:18:47</field>
  </row>
</resultset>
```

You can verify that the dump is valid by creating a copy of the `person` and then importing the dump file into the new table, like this:

```
mysql> USE test;
mysql> CREATE TABLE person2 LIKE person;
Query OK, 0 rows affected (0.00 sec)

mysql> LOAD XML LOCAL INFILE 'person-dump.xml'
```

```

-> INTO TABLE person2;
Query OK, 6 rows affected (0.01 sec)
Records: 6 Deleted: 0 Skipped: 0 Warnings: 0

mysql> SELECT * FROM person2;
+-----+-----+-----+-----+
| person_id | fname | lname | created |
+-----+-----+-----+-----+
|          1 | Pekka | Nousiainen | 2007-07-13 16:18:47 |
|          2 | Jonas | Oreland | 2007-07-13 16:18:47 |
|          3 | Mikael | Ronström | 2007-07-13 16:18:47 |
|          4 | Lars | Thalmann | 2007-07-13 16:18:47 |
|          5 | Tomas | Ulin | 2007-07-13 16:18:47 |
|          6 | Martin | Sköld | 2007-07-13 16:18:47 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)

```

Using a `ROWS IDENTIFIED BY '<tagname>'` clause, it is possible to import data from the same XML file into database tables with different definitions. For this example, suppose that you have a file named `address.xml` which contains the following XML:

```

<?xml version="1.0"?>
<list>
  <person person_id="1">
    <fname>Robert</fname>
    <lname>Jones</lname>
    <address address_id="1" street="Mill Creek Road" zip="45365" city="Sidney"/>
    <address address_id="2" street="Main Street" zip="28681" city="Taylorsville"/>
  </person>

  <person person_id="2">
    <fname>Mary</fname>
    <lname>Smith</lname>
    <address address_id="3" street="River Road" zip="80239" city="Denver"/>
    <!-- <address address_id="4" street="North Street" zip="37920" city="Knoxville"/> -->
  </person>
</list>

```

You can again use the `test.person` table as defined previously in this section, after clearing all the existing records from the table and then showing its structure as shown here:

```

mysql< TRUNCATE person;
Query OK, 0 rows affected (0.04 sec)

mysql< SHOW CREATE TABLE person\G
***** 1. row *****
      Table: person
Create Table: CREATE TABLE `person` (
  `person_id` int(11) NOT NULL,
  `fname` varchar(40) DEFAULT NULL,
  `lname` varchar(40) DEFAULT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`person_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

```

Now create an `address` table in the `test` database using the following `CREATE TABLE` statement:

```

CREATE TABLE address (
  address_id INT NOT NULL PRIMARY KEY,
  person_id INT NULL,
  street VARCHAR(40) NULL,
  zip INT NULL,
  city VARCHAR(40) NULL,

```

```

    created TIMESTAMP
);

```

To import the data from the XML file into the `person` table, execute the following `LOAD XML` statement, which specifies that rows are to be specified by the `<person>` element, as shown here:

```

mysql> LOAD XML LOCAL INFILE 'address.xml'
-> INTO TABLE person
-> ROWS IDENTIFIED BY '<person>';
Query OK, 2 rows affected (0.00 sec)
Records: 2 Deleted: 0 Skipped: 0 Warnings: 0

```

You can verify that the records were imported using a `SELECT` statement:

```

mysql> SELECT * FROM person;
+-----+-----+-----+-----+
| person_id | fname | lname | created          |
+-----+-----+-----+-----+
|          1 | Robert | Jones | 2007-07-24 17:37:06 |
|          2 | Mary  | Smith | 2007-07-24 17:37:06 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

Since the `<address>` elements in the XML file have no corresponding columns in the `person` table, they are skipped.

To import the data from the `<address>` elements into the `address` table, use the `LOAD XML` statement shown here:

```

mysql> LOAD XML LOCAL INFILE 'address.xml'
-> INTO TABLE address
-> ROWS IDENTIFIED BY '<address>';
Query OK, 3 rows affected (0.00 sec)
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0

```

You can see that the data was imported using a `SELECT` statement such as this one:

```

mysql> SELECT * FROM address;
+-----+-----+-----+-----+-----+-----+
| address_id | person_id | street          | zip   | city          | created          |
+-----+-----+-----+-----+-----+-----+
|          1 |          1 | Mill Creek Road | 45365 | Sidney        | 2007-07-24 17:37:37 |
|          2 |          1 | Main Street     | 28681 | Taylorsville  | 2007-07-24 17:37:37 |
|          3 |          2 | River Road     | 80239 | Denver        | 2007-07-24 17:37:37 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

The data from the `<address>` element that is enclosed in XML comments is not imported. However, since there is a `person_id` column in the `address` table, the value of the `person_id` attribute from the parent `<person>` element for each `<address>` is imported into the `address` table.

Security Considerations. As with the `LOAD DATA` statement, the transfer of the XML file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD XML` statement. Such a server could access any file on the client host to which the client user has read access.

In a Web environment, clients usually connect to MySQL from a Web server. A user that can run any command against the MySQL server can use `LOAD XML LOCAL` to read any files to which the Web server process has read access. In this environment, the client with respect to the MySQL server is actually the Web server, not the remote program being run by the user who connects to the Web server.

You can disable loading of XML files from clients by starting the server with `--local-infile=0` [513] or `--local-infile=OFF` [513]. This option can also be used when starting the `mysql` client to disable `LOAD XML` for the duration of the client session.

To prevent a client from loading XML files from the server, do not grant the `FILE` [696] privilege to the corresponding MySQL user account, or revoke this privilege if the client user account already has it.



Important

Revoking the `FILE` [696] privilege (or not granting it in the first place) keeps the user only from executing the `LOAD XML INFILE` statement (as well as the `LOAD_FILE()` [1105] function; it does *not* prevent the user from executing `LOAD XML LOCAL INFILE`. To disallow this statement, you must start the server or the client with `--local-infile=OFF`.

In other words, the `FILE` [696] privilege affects only whether the client can read files on the server; it has no bearing on whether the client can read files on the local file system.

For partitioned tables using storage engines that employ table locks, such as `MyISAM`, any locks caused by `LOAD XML` perform locks on all partitions of the table. This does not apply to tables using storage engines which employ row-level locking, such as `InnoDB`. For more information, see [Section 18.5.4, “Partitioning and Table-Level Locking”](#).

13.2.8. REPLACE Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
  [INTO] tbl_name [(col_name,...)]
  {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
  [INTO] tbl_name
  SET col_name={expr | DEFAULT}, ...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
  [INTO] tbl_name [(col_name,...)]
  SELECT ...
```

`REPLACE` works exactly like `INSERT`, except that if an old row in the table has the same value as a new row for a `PRIMARY KEY` or a `UNIQUE` index, the old row is deleted before the new row is inserted. See [Section 13.2.5, “INSERT Syntax”](#).

`REPLACE` is a MySQL extension to the SQL standard. It either inserts, or *deletes* and inserts. For another MySQL extension to standard SQL—that either inserts or *updates*—see [Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

Note that unless the table has a `PRIMARY KEY` or `UNIQUE` index, using a `REPLACE` statement makes no sense. It becomes equivalent to `INSERT`, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the `REPLACE` statement. Any missing columns are set to their default values, just as happens for `INSERT`. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as `SET col_name = col_name + 1`, the

reference to the column name on the right hand side is treated as `DEFAULT(col_name)` [1198], so the assignment is equivalent to `SET col_name = DEFAULT(col_name) + 1`.

To use `REPLACE`, you must have both the `INSERT` [696] and `DELETE` [696] privileges for the table.

The `REPLACE` statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row `REPLACE`, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether `REPLACE` only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the `mysql_affected_rows()` function.

Currently, you cannot replace into a table and select from the same table in a subquery.

MySQL uses the following algorithm for `REPLACE` (and `LOAD DATA ... REPLACE`):

1. Try to insert the new row into the table
2. While the insertion fails because a duplicate-key error occurs for a primary key or unique index:
 - a. Delete from the table the conflicting row that has the duplicate key value
 - b. Try again to insert the new row into the table

It is possible that in the case of a duplicate-key error, a storage engine may perform the `REPLACE` as an update rather than a delete plus insert, but the semantics are the same. There are no user-visible effects other than a possible difference in how the storage engine increments `Handler_xxx` status variables.

Because the results of `REPLACE ... SELECT` statements depend on the ordering of rows from the `SELECT` and this order cannot always be guaranteed, it is possible when logging these statements for the master and the slave to diverge. For this reason, in MySQL 5.5.18 and later, `REPLACE ... SELECT` statements are flagged as unsafe for statement-based replication. With this change, such statements produce a warning in the log when using the `STATEMENT` binary logging mode, and are logged using the row-based format when using `MIXED` mode. See also [Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”](#).

A `REPLACE` statement that acts on a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks all partitions of the table. This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking. This issue is resolved in MySQL 5.6. See [Section 18.5.4, “Partitioning and Table-Level Locking”](#), for more information.

13.2.9. SELECT Syntax

```
SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr [, select_expr ...]
  [FROM table_references]
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}]
```

```

[ASC | DESC], ... [WITH ROLLUP]]
[HAVING where_condition]
[ORDER BY {col_name | expr | position}
  [ASC | DESC], ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
[PROCEDURE procedure_name(argument_list)]
[INTO OUTFILE 'file_name'
  [CHARACTER SET charset_name]
  export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name]]
[FOR UPDATE | LOCK IN SHARE MODE]]

```

SELECT is used to retrieve rows selected from one or more tables, and can include **UNION** statements and subqueries. See [Section 13.2.9.4, “UNION Syntax”](#), and [Section 13.2.10, “Subquery Syntax”](#).

The most commonly used clauses of **SELECT** statements are these:

- Each *select_expr* indicates a column that you want to retrieve. There must be at least one *select_expr*.
- *table_references* indicates the table or tables from which to retrieve rows. Its syntax is described in [Section 13.2.9.2, “JOIN Syntax”](#).
- The **WHERE** clause, if given, indicates the condition or conditions that rows must satisfy to be selected. *where_condition* is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no **WHERE** clause.

In the **WHERE** expression, you can use any of the functions and operators that MySQL supports, except for aggregate (summary) functions. See [Section 9.5, “Expression Syntax”](#), and [Chapter 12, Functions and Operators](#).

SELECT can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;
-> 2
```

You are permitted to specify **DUAL** as a dummy table name in situations where no tables are referenced:

```
mysql> SELECT 1 + 1 FROM DUAL;
-> 2
```

DUAL is purely for the convenience of people who require that all **SELECT** statements should have **FROM** and possibly other clauses. MySQL may ignore the clauses. MySQL does not require **FROM DUAL** if no tables are referenced.

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a **HAVING** clause must come after any **GROUP BY** clause and before any **ORDER BY** clause. The exception is that the **INTO** clause can appear either as shown in the syntax description or immediately following the *select_expr* list. For more information about **INTO**, see [Section 13.2.9.1, “SELECT ... INTO Syntax”](#).

The list of *select_expr* terms comprises the select list that indicates which columns to retrieve. Terms specify a column or expression or can use *****-shorthand:

- A select list consisting only of a single unqualified ***** can be used as shorthand to select all columns from all tables:

```
SELECT * FROM t1 INNER JOIN t2 ...
```

- `tbl_name.*` can be used as a qualified shorthand to select all columns from the named table:

```
SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
```

- Use of an unqualified `*` with other items in the select list may produce a parse error. To avoid this problem, use a qualified `tbl_name.*` reference

```
SELECT AVG(score), t1.* FROM t1 ...
```

The following list provides additional information about other `SELECT` clauses:

- A `select_expr` can be given an alias using `AS alias_name`. The alias is used as the expression's column name and can be used in `GROUP BY`, `ORDER BY`, or `HAVING` clauses. For example:

```
SELECT CONCAT(last_name, ' ', first_name) AS full_name
FROM mytable ORDER BY full_name;
```

The `AS` keyword is optional when aliasing a `select_expr` with an identifier. The preceding example could have been written like this:

```
SELECT CONCAT(last_name, ' ', first_name) full_name
FROM mytable ORDER BY full_name;
```

However, because the `AS` is optional, a subtle problem can occur if you forget the comma between two `select_expr` expressions: MySQL interprets the second as an alias name. For example, in the following statement, `columnb` is treated as an alias name:

```
SELECT columna columnb FROM mytable;
```

For this reason, it is good practice to be in the habit of using `AS` explicitly when specifying column aliases.

It is not permissible to refer to a column alias in a `WHERE` clause, because the column value might not yet be determined when the `WHERE` clause is executed. See [Section C.5.5.4, “Problems with Column Aliases”](#).

- The `FROM table_references` clause indicates the table or tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see [Section 13.2.9.2, “JOIN Syntax”](#). For each table specified, you can optionally specify an alias.

```
tbl_name [[AS] alias] [index_hint]
```

The use of index hints provides the optimizer with information about how to choose indexes during query processing. For a description of the syntax for specifying these hints, see [Section 13.2.9.3, “Index Hint Syntax”](#).

You can use `SET max_seeks_for_key=value` as an alternative way to force MySQL to prefer key scans instead of table scans. See [Section 5.1.4, “Server System Variables”](#).

- You can refer to a table within the default database as `tbl_name`, or as `db_name.tbl_name` to specify a database explicitly. You can refer to a column as `col_name`, `tbl_name.col_name`, or `db_name.tbl_name.col_name`. You need not specify a `tbl_name` or `db_name.tbl_name` prefix for a column reference unless the reference would be ambiguous. See [Section 9.2.1, “Identifier Qualifiers”](#), for examples of ambiguity that require the more explicit column reference forms.
- A table reference can be aliased using `tbl_name AS alias_name` or `tbl_name alias_name`:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
WHERE t1.name = t2.name;
```

```
SELECT t1.name, t2.salary FROM employee t1, info t2
WHERE t1.name = t2.name;
```

- Columns selected for output can be referred to in [ORDER BY](#) and [GROUP BY](#) clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

```
SELECT college, region, seed FROM tournament
ORDER BY region, seed;

SELECT college, region AS r, seed AS s FROM tournament
ORDER BY r, s;

SELECT college, region, seed FROM tournament
ORDER BY 2, 3;
```

To sort in reverse order, add the [DESC](#) (descending) keyword to the name of the column in the [ORDER BY](#) clause that you are sorting by. The default is ascending order; this can be specified explicitly using the [ASC](#) keyword.

If [ORDER BY](#) occurs within a subquery and also is applied in the outer query, the outermost [ORDER BY](#) takes precedence. For example, results for the following statement are sorted in descending order, not ascending order:

```
(SELECT ... ORDER BY a) ORDER BY a DESC;
```

Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- If you use [GROUP BY](#), output rows are sorted according to the [GROUP BY](#) columns as if you had an [ORDER BY](#) for the same columns. To avoid the overhead of sorting that [GROUP BY](#) produces, add [ORDER BY NULL](#):

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

- MySQL extends the [GROUP BY](#) clause so that you can also specify [ASC](#) and [DESC](#) after columns named in the clause:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;
```

- MySQL extends the use of [GROUP BY](#) to permit selecting fields that are not mentioned in the [GROUP BY](#) clause. If you are not getting the results that you expect from your query, please read the description of [GROUP BY](#) found in [Section 12.16, "Functions and Modifiers for Use with GROUP BY Clauses"](#).
- [GROUP BY](#) permits a [WITH ROLLUP](#) modifier. See [Section 12.16.2, "GROUP BY Modifiers"](#).
- The [HAVING](#) clause is applied nearly last, just before items are sent to the client, with no optimization. ([LIMIT](#) is applied after [HAVING](#).)

The SQL standard requires that [HAVING](#) must reference only columns in the [GROUP BY](#) clause or columns used in aggregate functions. However, MySQL supports an extension to this behavior, and permits [HAVING](#) to refer to columns in the [SELECT](#) list and columns in outer subqueries as well.

If the [HAVING](#) clause refers to a column that is ambiguous, a warning occurs. In the following statement, `col2` is ambiguous because it is used as both an alias and a column name:

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

Preference is given to standard SQL behavior, so if a [HAVING](#) column name is used both in [GROUP BY](#) and as an aliased column in the output column list, preference is given to the column in the [GROUP BY](#) column.

- Do not use `HAVING` for items that should be in the `WHERE` clause. For example, do not write the following:

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

Write this instead:

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- The `HAVING` clause can refer to aggregate functions, which the `WHERE` clause cannot:

```
SELECT user, MAX(salary) FROM users
GROUP BY user HAVING MAX(salary) > 10;
```

(This did not work in some older versions of MySQL.)

- MySQL permits duplicate column names. That is, there can be more than one `select_expr` with the same name. This is an extension to standard SQL. Because MySQL also permits `GROUP BY` and `HAVING` to refer to `select_expr` values, this can result in an ambiguity:

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

In that statement, both columns have the name `a`. To ensure that the correct column is used for grouping, use different names for each `select_expr`.

- MySQL resolves unqualified column or alias references in `ORDER BY` clauses by searching in the `select_expr` values, then in the columns of the tables in the `FROM` clause. For `GROUP BY` or `HAVING` clauses, it searches the `FROM` clause before searching in the `select_expr` values. (For `GROUP BY` and `HAVING`, this differs from the pre-MySQL 5.0 behavior that used the same rules as for `ORDER BY`.)
- The `LIMIT` clause can be used to constrain the number of rows returned by the `SELECT` statement. `LIMIT` takes one or two numeric arguments, which must both be nonnegative integer constants, with these exceptions:
 - Within prepared statements, `LIMIT` parameters can be specified using `?` placeholder markers.
 - Within stored programs, `LIMIT` parameters can be specified using integer-valued routine parameters or local variables as of MySQL 5.5.6.

With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

```
SELECT * FROM tbl LIMIT 5,10; # Retrieve rows 6-15
```

To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

With one argument, the value specifies the number of rows to return from the beginning of the result set:

```
SELECT * FROM tbl LIMIT 5; # Retrieve first 5 rows
```

In other words, `LIMIT row_count` is equivalent to `LIMIT 0, row_count`.

For prepared statements, you can use placeholders. The following statements will return one row from the `tbl` table:

```
SET @a=1;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
```

```
EXECUTE STMT USING @a;
```

The following statements will return the second to sixth row from the `tbl` table:

```
SET @skip=1; SET @numrows=5;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
EXECUTE STMT USING @skip, @numrows;
```

For compatibility with PostgreSQL, MySQL also supports the `LIMIT row_count OFFSET offset` syntax.

If `LIMIT` occurs within a subquery and also is applied in the outer query, the outermost `LIMIT` takes precedence. For example, the following statement produces two rows, not one:

```
(SELECT ... LIMIT 1) LIMIT 2;
```

- A `PROCEDURE` clause names a procedure that should process the data in the result set. For an example, see [Section 8.4.2.4, “Using PROCEDURE ANALYSE”](#), which describes `ANALYSE`, a procedure that can be used to obtain suggestions for optimal column data types that may help reduce table sizes.
- The `SELECT ... INTO` form of `SELECT` enables the query result to be written to a file or stored in variables. For more information, see [Section 13.2.9.1, “SELECT ... INTO Syntax”](#).
- If you use `FOR UPDATE` with a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction. Using `LOCK IN SHARE MODE` sets a shared lock that permits other transactions to read the examined rows but not to update or delete them. See [Section 14.3.9.3, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”](#).

Following the `SELECT` keyword, you can use a number of options that affect the operation of the statement. `HIGH_PRIORITY`, `STRAIGHT_JOIN`, and options beginning with `SQL_` are MySQL extensions to standard SQL.

- The `ALL` and `DISTINCT` options specify whether duplicate rows should be returned. `ALL` (the default) specifies that all matching rows should be returned, including duplicates. `DISTINCT` specifies removal of duplicate rows from the result set. It is an error to specify both options. `DISTINCTROW` is a synonym for `DISTINCT`.
- `HIGH_PRIORITY` gives the `SELECT` higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A `SELECT HIGH_PRIORITY` query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

`HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`.

- `STRAIGHT_JOIN` forces the optimizer to join the tables in the order in which they are listed in the `FROM` clause. You can use this to speed up a query if the optimizer joins the tables in nonoptimal order. `STRAIGHT_JOIN` also can be used in the `table_references` list. See [Section 13.2.9.2, “JOIN Syntax”](#).

`STRAIGHT_JOIN` does not apply to any table that the optimizer treats as a `const` [837] or `system` [837] table. Such a table produces a single row, is read during the optimization phase of query execution, and references to its columns are replaced with the appropriate column values before query execution proceeds. These tables will appear first in the query plan displayed by `EXPLAIN`. See [Section 8.8.1, “Optimizing Queries with EXPLAIN”](#). This exception may not apply to `const` [837] or `system` [837] tables that are used on the `NULL`-complemented side of an outer join (that is, the right-side table of a `LEFT JOIN` or the left-side table of a `RIGHT JOIN`).

- `SQL_BIG_RESULT` or `SQL_SMALL_RESULT` can be used with `GROUP BY` or `DISTINCT` to tell the optimizer that the result set has many rows or is small, respectively. For `SQL_BIG_RESULT`, MySQL directly uses disk-based temporary tables if needed, and prefers sorting to using a temporary table with a key on the `GROUP BY` elements. For `SQL_SMALL_RESULT`, MySQL uses fast temporary tables to store the resulting table instead of using sorting. This should not normally be needed.
- `SQL_BUFFER_RESULT` forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client. This option can be used only for top-level `SELECT` statements, not for subqueries or following `UNION`.
- `SQL_CALC_FOUND_ROWS` tells MySQL to calculate how many rows there would be in the result set, disregarding any `LIMIT` clause. The number of rows can then be retrieved with `SELECT FOUND_ROWS()`. See [Section 12.14, “Information Functions”](#).
- The `SQL_CACHE` and `SQL_NO_CACHE` options affect caching of query results in the query cache (see [Section 8.9.3, “The MySQL Query Cache”](#)). `SQL_CACHE` tells MySQL to store the result in the query cache if it is cacheable and the value of the `query_cache_type` [542] system variable is 2 or `DEMAND`. `SQL_NO_CACHE` tells MySQL not to store the result in the query cache.

For views, `SQL_NO_CACHE` applies if it appears in any `SELECT` in the query. For a cacheable query, `SQL_CACHE` applies if it appears in the first `SELECT` of a view referred to by the query.

As of MySQL 5.5.3, these two options are mutually exclusive and an error occurs if they are both specified. Also, these options are not permitted in subqueries (including subqueries in the `FROM` clause), and `SELECT` statements in unions other than the first `SELECT`.

Before MySQL 5.5.3, for a query that uses `UNION` or subqueries, the following rules apply:

- `SQL_NO_CACHE` applies if it appears in any `SELECT` in the query.
- For a cacheable query, `SQL_CACHE` applies if it appears in the first `SELECT` of the query.

A `SELECT` from a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks all partitions of the table. This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking. This issue is resolved in MySQL 5.6. See [Section 18.5.4, “Partitioning and Table-Level Locking”](#), for more information.

13.2.9.1. `SELECT ... INTO` Syntax

The `SELECT ... INTO` form of `SELECT` enables a query result to be written to a file or stored in variables:

- `SELECT ... INTO OUTFILE` writes the selected rows to a file. Column and line terminators can be specified to produce a specific output format.
- `SELECT ... INTO DUMPFILE` writes a single row to a file without any formatting.
- `SELECT ... INTO var_list` selects column values and into variables.

The `SELECT` syntax description (see [Section 13.2.9, “SELECT Syntax”](#)) shows the `INTO` clause near the end of the statement. It is also possible to use `INTO` immediately following the `select_expr` list.

The `SELECT ... INTO OUTFILE 'file_name'` form of `SELECT` writes the selected rows to a file. The file is created on the server host, so you must have the `FILE` [696] privilege to use this syntax. `file_name` cannot be an existing file, which among other things prevents files such as `/etc/passwd` and database tables from being destroyed. The `character_set_filesystem` [487] system variable controls the interpretation of the file name.

The `SELECT ... INTO OUTFILE` statement is intended primarily to let you very quickly dump a table to a text file on the server machine. If you want to create the resulting file on some other host than the server host, you normally cannot use `SELECT ... INTO OUTFILE` since there is no way to write a path to the file relative to the server host's file system.

However, if the MySQL client software is installed on the remote machine, you can instead use a client command such as `mysql -e "SELECT ..." > file_name` to generate the file on the client host.

It is also possible to create the resulting file on a different host other than the server host, if the location of the file on the remote host can be accessed using a network-mapped path on the server's file system. In this case, the presence of `mysql` (or some other MySQL client program) is not required on the target host.

`SELECT ... INTO OUTFILE` is the complement of `LOAD DATA INFILE`. Column values are written converted to the character set specified in the `CHARACTER SET` clause. If no such clause is present, values are dumped using the `binary` character set. In effect, there is no character set conversion. If a result set contains columns in several character sets, the output data file will as well and you may not be able to reload the file correctly.

The syntax for the `export_options` part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA INFILE` statement. See [Section 13.2.6, "LOAD DATA INFILE Syntax"](#), for information about the `FIELDS` and `LINES` clauses, including their default values and permissible values.

`FIELDS ESCAPED BY` controls how to write special characters. If the `FIELDS ESCAPED BY` character is not empty, it is used when necessary to avoid ambiguity as a prefix that precedes following characters on output:

- The `FIELDS ESCAPED BY` character
- The `FIELDS [OPTIONALLY] ENCLOSED BY` character
- The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
- ASCII `NUL` (the zero-valued byte; what is actually written following the escape character is ASCII "0", not a zero-valued byte)

The `FIELDS TERMINATED BY`, `ENCLOSED BY`, `ESCAPED BY`, or `LINES TERMINATED BY` characters *must* be escaped so that you can read the file back in reliably. ASCII `NUL` is escaped to make it easier to view with some paggers.

The resulting file does not have to conform to SQL syntax, so nothing else need be escaped.

If the `FIELDS ESCAPED BY` character is empty, no characters are escaped and `NULL` is output as `NULL`, not `\N`. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

Here is an example that produces a file in the comma-separated values (CSV) format used by many programs:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n'
FROM test_table;
```

If you use `INTO DUMPFILE` instead of `INTO OUTFILE`, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a `BLOB` value in a file.

**Note**

Any file created by `INTO outfile` or `INTO outfile` is writable by all users on the server host. The reason for this is that the MySQL server cannot create a file that is owned by anyone other than the user under whose account it is running. (You should *never* run `mysqld` as `root` for this and other reasons.) The file thus must be world-writable so that you can manipulate its contents.

If the `secure_file_priv` [550] system variable is set to a nonempty directory name, the file to be written must be located in that directory.

The `INTO` clause can name a list of one or more variables, which can be user-defined variables, stored procedure or function parameters, or stored program local variables (see [Section 13.6.4, “Variables in Stored Programs”](#)). The selected values are assigned to the variables. The number of variables must match the number of columns. The query should return a single row. If the query returns no rows, a warning with error code 1329 occurs (`No data`), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (`Result consisted of more than one row`). If it is possible that the statement may retrieve multiple rows, you can use `LIMIT 1` to limit the result set to a single row.

```
SELECT id, data INTO @x, @y FROM test.t1 LIMIT 1;
```

User variable names are not case sensitive. See [Section 9.4, “User-Defined Variables”](#).

In the context of such statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For additional information, see [Section 19.4.5, “Event Scheduler Status”](#).

An `INTO` clause should not be used in a nested `SELECT` because such a `SELECT` must return its result to the outer context.

13.2.9.2. JOIN Syntax

MySQL supports the following `JOIN` syntaxes for the `table_references` part of `SELECT` statements and multiple-table `DELETE` and `UPDATE` statements:

```
table_references:
    table_reference [, table_reference] ...

table_reference:
    table_factor
  | join_table

table_factor:
    tbl_name [[AS] alias] [index_hint_list]
  | table_subquery [AS] alias
  | ( table_references )
  | { OJ table_reference LEFT OUTER JOIN table_reference
      ON conditional_expr }

join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor

join_condition:
    ON conditional_expr
  | USING (column_list)
```

```

index_hint_list:
    index_hint [, index_hint] ...

index_hint:
    USE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
    | IGNORE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
    | FORCE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...

```

A table reference is also known as a join expression.

The syntax of *table_factor* is extended in comparison with the SQL Standard. The latter accepts only *table_reference*, not a list of them inside a pair of parentheses.

This is a conservative extension if we consider each comma in a list of *table_reference* items as equivalent to an inner join. For example:

```

SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
    ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

```

is equivalent to:

```

SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
    ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

```

In MySQL, **JOIN**, **CROSS JOIN**, and **INNER JOIN** are syntactic equivalents (they can replace each other). In standard SQL, they are not equivalent. **INNER JOIN** is used with an **ON** clause, **CROSS JOIN** is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. MySQL also supports nested joins (see [Section 8.13.7, “Nested Join Optimization”](#)).

Index hints can be specified to affect how the MySQL optimizer makes use of indexes. For more information, see [Section 13.2.9.3, “Index Hint Syntax”](#).

The following list describes general factors to take into account when writing joins.

- A table reference can be aliased using *tbl_name AS alias_name* or *tbl_name alias_name*:

```

SELECT t1.name, t2.salary
    FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

SELECT t1.name, t2.salary
    FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;

```

- A *table_subquery* is also known as a subquery in the **FROM** clause. Such subqueries *must* include an alias to give the subquery result a table name. A trivial example follows; see also [Section 13.2.10.8, “Subqueries in the FROM Clause”](#).

```

SELECT * FROM (SELECT 1, 2, 3) AS t1;

```

- **INNER JOIN** and **,** (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table is joined to each and every row in the second table).

However, the precedence of the comma operator is less than of **INNER JOIN**, **CROSS JOIN**, **LEFT JOIN**, and so on. If you mix comma joins with the other join types when there is a join condition, an error

of the form `Unknown column 'col_name' in 'on clause'` may occur. Information about dealing with this problem is given later in this section.

- The `conditional_expr` used with `ON` is any conditional expression of the form that can be used in a `WHERE` clause. Generally, you should use the `ON` clause for conditions that specify how to join tables, and the `WHERE` clause to restrict which rows you want in the result set.
- If there is no matching row for the right table in the `ON` or `USING` part in a `LEFT JOIN`, a row with all columns set to `NULL` is used for the right table. You can use this fact to find rows in a table that have no counterpart in another table:

```
SELECT left_tbl.*
   FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
   WHERE right_tbl.id IS NULL;
```

This example finds all rows in `left_tbl` with an `id` value that is not present in `right_tbl` (that is, all rows in `left_tbl` with no corresponding row in `right_tbl`). This assumes that `right_tbl.id` is declared `NOT NULL`. See [Section 8.13.5, “LEFT JOIN and RIGHT JOIN Optimization”](#).

- The `USING(column_list)` clause names a list of columns that must exist in both tables. If tables `a` and `b` both contain columns `c1`, `c2`, and `c3`, the following join compares corresponding columns from the two tables:

```
a LEFT JOIN b USING (c1,c2,c3)
```

- The `NATURAL [LEFT] JOIN` of two tables is defined to be semantically equivalent to an `INNER JOIN` or a `LEFT JOIN` with a `USING` clause that names all columns that exist in both tables.
- `RIGHT JOIN` works analogously to `LEFT JOIN`. To keep code portable across databases, it is recommended that you use `LEFT JOIN` instead of `RIGHT JOIN`.
- The `{ OJ ... LEFT OUTER JOIN ... }` syntax shown in the join syntax description exists only for compatibility with ODBC. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

```
SELECT left_tbl.*
   FROM { OJ left_tbl LEFT OUTER JOIN right_tbl ON left_tbl.id = right_tbl.id }
   WHERE right_tbl.id IS NULL;
```

You can use other types of joins within `{ OJ ... }`, such as `INNER JOIN` or `RIGHT OUTER JOIN`. This helps with compatibility with some third-party applications, but is not official ODBC syntax.

The parser does not permit nested `{ OJ ... }` constructs (which are not legal ODBC syntax, anyway). Queries that use such constructs should be rewritten. For an example, see [JOIN Syntax](#).

- `STRAIGHT_JOIN` is similar to `JOIN`, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order.

Some join examples:

```
SELECT * FROM table1, table2;

SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 USING (id);

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
   LEFT JOIN table3 ON table2.id=table3.id;
```

Join Processing Changes in MySQL 5.0.12

**Note**

Natural joins and joins with `USING`, including outer join variants, are processed according to the SQL:2003 standard. The goal was to align the syntax and semantics of MySQL with respect to `NATURAL JOIN` and `JOIN ... USING` according to SQL:2003. However, these changes in join processing can result in different output columns for some joins. Also, some queries that appeared to work correctly in older versions (prior to 5.0.12) must be rewritten to comply with the standard.

These changes have five main aspects:

- The way that MySQL determines the result columns of `NATURAL` or `USING` join operations (and thus the result of the entire `FROM` clause).
- Expansion of `SELECT *` and `SELECT tbl_name.*` into a list of selected columns.
- Resolution of column names in `NATURAL` or `USING` joins.
- Transformation of `NATURAL` or `USING` joins into `JOIN ... ON`.
- Resolution of column names in the `ON` condition of a `JOIN ... ON`.

The following list provides more detail about several effects of current join processing versus join processing in older versions. The term “previously” means “prior to MySQL 5.0.12.”

- The columns of a `NATURAL` join or a `USING` join may be different from previously. Specifically, redundant output columns no longer appear, and the order of columns for `SELECT *` expansion may be different from before.

Consider this set of statements:

```
CREATE TABLE t1 (i INT, j INT);
CREATE TABLE t2 (k INT, j INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 JOIN t2 USING (j);
```

Previously, the statements produced this output:

```
+-----+-----+-----+-----+
| i     | j     | k     | j     |
+-----+-----+-----+-----+
|  1   |  1   |  1   |  1   |
+-----+-----+-----+-----+
| i     | j     | k     | j     |
+-----+-----+-----+-----+
|  1   |  1   |  1   |  1   |
+-----+-----+-----+-----+
```

In the first `SELECT` statement, column `j` appears in both tables and thus becomes a join column, so, according to standard SQL, it should appear only once in the output, not twice. Similarly, in the second `SELECT` statement, column `j` is named in the `USING` clause and should appear only once in the output, not twice. But in both cases, the redundant column is not eliminated. Also, the order of the columns is not correct according to standard SQL.

Now the statements produce this output:

```

+-----+-----+-----+
| j     | i     | k     |
+-----+-----+-----+
| 1     | 1     | 1     |
+-----+-----+-----+
| j     | i     | k     |
+-----+-----+-----+
| 1     | 1     | 1     |
+-----+-----+-----+

```

The redundant column is eliminated and the column order is correct according to standard SQL:

- First, coalesced common columns of the two joined tables, in the order in which they occur in the first table
- Second, columns unique to the first table, in order in which they occur in that table
- Third, columns unique to the second table, in order in which they occur in that table

The single result column that replaces two common columns is defined using the coalesce operation. That is, for two `t1.a` and `t2.a` the resulting single join column `a` is defined as `a = COALESCE(t1.a, t2.a)`, where:

```
COALESCE(x, y) = (CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END)
```

If the join operation is any other join, the result columns of the join consists of the concatenation of all columns of the joined tables. This is the same as previously.

A consequence of the definition of coalesced columns is that, for outer joins, the coalesced column contains the value of the non-`NULL` column if one of the two columns is always `NULL`. If neither or both columns are `NULL`, both common columns have the same value, so it doesn't matter which one is chosen as the value of the coalesced column. A simple way to interpret this is to consider that a coalesced column of an outer join is represented by the common column of the inner table of a `JOIN`. Suppose that the tables `t1(a,b)` and `t2(a,c)` have the following contents:

```

t1    t2
----  ---
1 x   2 z
2 y   3 w

```

Then:

```

mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
+-----+-----+-----+
| a     | b     | c     |
+-----+-----+-----+
| 1     | x     | NULL  |
| 2     | y     | z     |
+-----+-----+-----+

```

Here column `a` contains the values of `t1.a`.

```

mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
+-----+-----+-----+
| a     | c     | b     |
+-----+-----+-----+
| 2     | z     | y     |
| 3     | w     | NULL  |
+-----+-----+-----+

```

Here column `a` contains the values of `t2.a`.

Compare these results to the otherwise equivalent queries with `JOIN ... ON`:

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
```

a	b	a	c
1	x	NULL	NULL
2	y	2	z

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
```

a	b	a	c
2	y	2	z
NULL	NULL	3	w

- Previously, a `USING` clause could be rewritten as an `ON` clause that compares corresponding columns. For example, the following two clauses were semantically identical:

```
a LEFT JOIN b USING (c1,c2,c3)
a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
```

Now the two clauses no longer are quite the same:

- With respect to determining which rows satisfy the join condition, both joins remain semantically identical.
- With respect to determining which columns to display for `SELECT *` expansion, the two joins are not semantically identical. The `USING` join selects the coalesced value of corresponding columns, whereas the `ON` join selects all columns from all tables. For the preceding `USING` join, `SELECT *` selects these values:

```
COALESCE(a.c1,b.c1), COALESCE(a.c2,b.c2), COALESCE(a.c3,b.c3)
```

For the `ON` join, `SELECT *` selects these values:

```
a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
```

With an inner join, `COALESCE(a.c1,b.c1)` [1092] is the same as either `a.c1` or `b.c1` because both columns will have the same value. With an outer join (such as `LEFT JOIN`), one of the two columns can be `NULL`. That column will be omitted from the result.

- The evaluation of multi-way natural joins differs in a very important way that affects the result of `NATURAL` or `USING` joins and that can require query rewriting. Suppose that you have three tables `t1(a,b)`, `t2(c,b)`, and `t3(a,c)` that each have one row: `t1(1,2)`, `t2(10,2)`, and `t3(7,10)`. Suppose also that you have this `NATURAL JOIN` on the three tables:

```
SELECT ... FROM t1 NATURAL JOIN t2 NATURAL JOIN t3;
```

Previously, the left operand of the second join was considered to be `t2`, whereas it should be the nested join (`t1 NATURAL JOIN t2`). As a result, the columns of `t3` are checked for common columns only in `t2`, and, if `t3` has common columns with `t1`, these columns are not used as equi-join columns. Thus, previously, the preceding query was transformed to the following equi-join:

```
SELECT ... FROM t1, t2, t3
WHERE t1.b = t2.b AND t2.c = t3.c;
```

That join is missing one more equi-join predicate (`t1.a = t3.a`). As a result, it produces one row, not the empty result that it should. The correct equivalent query is this:

```
SELECT ... FROM t1, t2, t3
WHERE t1.b = t2.b AND t2.c = t3.c AND t1.a = t3.a;
```

If you require the same query result in current versions of MySQL as in older versions, rewrite the natural join as the first equi-join.

- Previously, the comma operator (`,`) and `JOIN` both had the same precedence, so the join expression `t1, t2 JOIN t3` was interpreted as `((t1, t2) JOIN t3)`. Now `JOIN` has higher precedence, so the expression is interpreted as `(t1, (t2 JOIN t3))`. This change affects statements that use an `ON` clause, because that clause can refer only to columns in the operands of the join, and the change in precedence changes interpretation of what those operands are.

Example:

```
CREATE TABLE t1 (i1 INT, j1 INT);
CREATE TABLE t2 (i2 INT, j2 INT);
CREATE TABLE t3 (i3 INT, j3 INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
INSERT INTO t3 VALUES(1,1);
SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
```

Previously, the `SELECT` was legal due to the implicit grouping of `t1, t2` as `(t1, t2)`. Now the `JOIN` takes precedence, so the operands for the `ON` clause are `t2` and `t3`. Because `t1.i1` is not a column in either of the operands, the result is an `Unknown column 't1.i1' in 'on clause'` error. To allow the join to be processed, group the first two tables explicitly with parentheses so that the operands for the `ON` clause are `(t1, t2)` and `t3`:

```
SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
```

Alternatively, avoid the use of the comma operator and use `JOIN` instead:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
```

This change also applies to statements that mix the comma operator with `INNER JOIN`, `CROSS JOIN`, `LEFT JOIN`, and `RIGHT JOIN`, all of which now have higher precedence than the comma operator.

- Previously, the `ON` clause could refer to columns in tables named to its right. Now an `ON` clause can refer only to its operands.

Example:

```
CREATE TABLE t1 (i1 INT);
CREATE TABLE t2 (i2 INT);
CREATE TABLE t3 (i3 INT);
SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
```

Previously, the `SELECT` statement was legal. Now the statement fails with an `Unknown column 'i3' in 'on clause'` error because `i3` is a column in `t3`, which is not an operand of the `ON` clause. The statement should be rewritten as follows:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
```

- Resolution of column names in `NATURAL` or `USING` joins is different than previously. For column names that are outside the `FROM` clause, MySQL now handles a superset of the queries compared to previously. That is, in cases when MySQL formerly issued an error that some column is ambiguous, the query now

is handled correctly. This is due to the fact that MySQL now treats the common columns of [NATURAL](#) or [USING](#) joins as a single column, so when a query refers to such columns, the query compiler does not consider them as ambiguous.

Example:

```
SELECT * FROM t1 NATURAL JOIN t2 WHERE b > 1;
```

Previously, this query would produce an error `ERROR 1052 (23000): Column 'b' in where clause is ambiguous`. Now the query produces the correct result:

```
+-----+-----+-----+
| b     | c     | y     |
+-----+-----+-----+
|      4 |      2 |      3 |
+-----+-----+-----+
```

One extension of MySQL compared to the SQL:2003 standard is that MySQL enables you to qualify the common (coalesced) columns of [NATURAL](#) or [USING](#) joins (just as previously), while the standard disallows that.

13.2.9.3. Index Hint Syntax

You can provide hints to give the optimizer information about how to choose indexes during query processing. [Section 13.2.9.2, "JOIN Syntax"](#), describes the general syntax for specifying tables in a [SELECT](#) statement. The syntax for an individual table, including that for index hints, looks like this:

```
tbl_name [[AS] alias] [index_hint_list]

index_hint_list:
    index_hint [, index_hint] ...

index_hint:
    USE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
    | IGNORE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
    | FORCE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...
```

By specifying `USE INDEX (index_list)`, you can tell MySQL to use only one of the named indexes to find rows in the table. The alternative syntax `IGNORE INDEX (index_list)` can be used to tell MySQL to not use some particular index or indexes. These hints are useful if `EXPLAIN` shows that MySQL is using the wrong index from the list of possible indexes.

You can also use `FORCE INDEX`, which acts like `USE INDEX (index_list)` but with the addition that a table scan is assumed to be *very* expensive. In other words, a table scan is used only if there is no way to use one of the given indexes to find rows in the table.

Each hint requires the names of *indexes*, not the names of columns. The name of a [PRIMARY KEY](#) is `PRIMARY`. To see the index names for a table, use `SHOW INDEX`.

An `index_name` value need not be a full index name. It can be an unambiguous prefix of an index name. If a prefix is ambiguous, an error occurs.

Examples:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
```

```
WHERE col1=1 AND col2=2 AND col3=3;

SELECT * FROM table1 IGNORE INDEX (col3_index)
WHERE col1=1 AND col2=2 AND col3=3;
```

The syntax for index hints has the following characteristics:

- It is syntactically valid to specify an empty *index_list* for `USE INDEX`, which means “use no indexes.” Specifying an empty *index_list* for `FORCE INDEX` or `IGNORE INDEX` is a syntax error.
- You can specify the scope of a index hint by adding a `FOR` clause to the hint. This provides more fine-grained control over the optimizer's selection of an execution plan for various phases of query processing. To affect only the indexes used when MySQL decides how to find rows in the table and how to process joins, use `FOR JOIN`. To influence index usage for sorting or grouping rows, use `FOR ORDER BY` or `FOR GROUP BY`. (However, if there is a covering index for the table and it is used to access the table, the optimizer will ignore `IGNORE INDEX FOR {ORDER BY|GROUP BY}` hints that disable that index.)
- You can specify multiple index hints:

```
SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX FOR ORDER BY (i2) ORDER BY a;
```

It is not a error to name the same index in several hints (even within the same hint):

```
SELECT * FROM t1 USE INDEX (i1) USE INDEX (i1,i1);
```

However, it is an error to mix `USE INDEX` and `FORCE INDEX` for the same table:

```
SELECT * FROM t1 USE INDEX FOR JOIN (i1) FORCE INDEX FOR JOIN (i2);
```

if you specify no `FOR` clause for an index hint, the hint by default applies to all parts of the statement. For example, this hint:

```
IGNORE INDEX (i1)
```

is equivalent to this combination of hints:

```
IGNORE INDEX FOR JOIN (i1)
IGNORE INDEX FOR ORDER BY (i1)
IGNORE INDEX FOR GROUP BY (i1)
```

To cause the server to use the older behavior for hint scope when no `FOR` clause is present (so that hints apply only to row retrieval), enable the `old [534]` system variable at server startup. Take care about enabling this variable in a replication setup. With statement-based binary logging, having different modes for the master and slaves might lead to replication errors.

When index hints are processed, they are collected in a single list by type (`USE`, `FORCE`, `IGNORE`) and by scope (`FOR JOIN`, `FOR ORDER BY`, `FOR GROUP BY`). For example:

```
SELECT * FROM t1
USE INDEX () IGNORE INDEX (i2) USE INDEX (i1) USE INDEX (i2);
```

is equivalent to:

```
SELECT * FROM t1
USE INDEX (i1,i2) IGNORE INDEX (i2);
```

The index hints then are applied for each scope in the following order:

1. `{USE|FORCE} INDEX` is applied if present. (If not, the optimizer-determined set of indexes is used.)

2. `IGNORE INDEX` is applied over the result of the previous step. For example, the following two queries are equivalent:

```
SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX (i2) USE INDEX (i2);
SELECT * FROM t1 USE INDEX (i1);
```

For `FULLTEXT` searches, index hints work as follows:

- For natural language mode searches, index hints are silently ignored. For example, `IGNORE INDEX(i)` is ignored with no warning and the index is still used.

For boolean mode searches, index hints with `FOR ORDER BY` or `FOR GROUP BY` are silently ignored. Index hints with `FOR JOIN` or no `FOR` modifier are honored. In contrast to how hints apply for non-`FULLTEXT` searches, the hint is used for all phases of query execution (finding rows and retrieval, grouping, and ordering). This is true even if the hint is given for a non-`FULLTEXT` index.

For example, the following two queries are equivalent:

```
SELECT * FROM t
  USE INDEX (index1)
  IGNORE INDEX (index1) FOR ORDER BY
  IGNORE INDEX (index1) FOR GROUP BY
  WHERE ... IN BOOLEAN MODE ... ;

SELECT * FROM t
  USE INDEX (index1)
  WHERE ... IN BOOLEAN MODE ... ;
```

13.2.9.4. UNION Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

`UNION` is used to combine the result from multiple `SELECT` statements into a single result set.

The column names from the first `SELECT` statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each `SELECT` statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If the data types of corresponding `SELECT` columns do not match, the types and lengths of the columns in the `UNION` result take into account the values retrieved by all of the `SELECT` statements. For example, consider the following:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+-----+
| REPEAT('a',1) |
+-----+
| a             |
| bbbbbbbbbb   |
+-----+
```

The `SELECT` statements are normal select statements, but with the following restrictions:

- Only the last `SELECT` statement can use `INTO OUTFILE`. (However, the entire `UNION` result is written to the file.)
- `HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`. If you specify it for the first `SELECT`, it has no effect. If you specify it for any subsequent `SELECT` statements, a syntax error results.

The default behavior for `UNION` is that duplicate rows are removed from the result. The optional `DISTINCT` keyword has no effect other than the default because it also specifies duplicate-row removal. With the optional `ALL` keyword, duplicate-row removal does not occur and the result includes all matching rows from all the `SELECT` statements.

You can mix `UNION ALL` and `UNION DISTINCT` in the same query. Mixed `UNION` types are treated such that a `DISTINCT` union overrides any `ALL` union to its left. A `DISTINCT` union can be produced explicitly by using `UNION DISTINCT` or implicitly by using `UNION` with no following `DISTINCT` or `ALL` keyword.

To apply `ORDER BY` or `LIMIT` to an individual `SELECT`, place the clause inside the parentheses that enclose the `SELECT`:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

However, use of `ORDER BY` for individual `SELECT` statements implies nothing about the order in which the rows appear in the final result because `UNION` by default produces an unordered set of rows. Therefore, the use of `ORDER BY` in this context is typically in conjunction with `LIMIT`, so that it is used to determine the subset of the selected rows to retrieve for the `SELECT`, even though it does not necessarily affect the order of those rows in the final `UNION` result. If `ORDER BY` appears without `LIMIT` in a `SELECT`, it is optimized away because it will have no effect anyway.

To use an `ORDER BY` or `LIMIT` clause to sort or limit the entire `UNION` result, parenthesize the individual `SELECT` statements and place the `ORDER BY` or `LIMIT` after the last one. The following example uses both clauses:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

A statement without parentheses is equivalent to one parenthesized as just shown.

This kind of `ORDER BY` cannot use column references that include a table name (that is, names in `tbl_name.col_name` format). Instead, provide a column alias in the first `SELECT` statement and refer to the alias in the `ORDER BY`. (Alternatively, refer to the column in the `ORDER BY` using its column position. However, use of column positions is deprecated.)

Also, if a column to be sorted is aliased, the `ORDER BY` clause *must* refer to the alias, not the column name. The first of the following statements will work, but the second will fail with an `Unknown column 'a' in 'order clause'` error:

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

To cause rows in a `UNION` result to consist of the sets of rows retrieved by each `SELECT` one after the other, select an additional column in each `SELECT` to use as a sort column and add an `ORDER BY` following the last `SELECT`:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

To additionally maintain sort order within individual `SELECT` results, add a secondary column to the `ORDER BY` clause:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

Use of an additional column also enables you to determine which `SELECT` each row comes from. Extra columns can provide other identifying information as well, such as a string that indicates a table name.

13.2.10. Subquery Syntax

A subquery is a `SELECT` statement within another statement.

Starting with MySQL 4.1, all subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, `SELECT * FROM t1 ...` is the *outer query* (or *outer statement*), and `(SELECT column1 FROM t2)` is the *subquery*. We say that the subquery is *nested* within the outer query, and in fact it is possible to nest subqueries within other subqueries, to a considerable depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- Many people find subqueries more readable than complex joins or unions. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
  (SELECT COUNT(*) /* no hint */ FROM t2
   WHERE NOT EXISTS
    (SELECT * FROM t3
     WHERE ROW(5*t2.s1,77)=
      (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
       (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used. A subquery can contain many of the keywords or clauses that an ordinary `SELECT` can contain: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, index hints, `UNION` constructs, comments, functions, and so on.

A subquery's outer statement can be any one of: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, or `DO`.

In MySQL, you cannot modify a table and select from the same table in a subquery. This applies to statements such as `DELETE`, `INSERT`, `REPLACE`, `UPDATE`, and (because subqueries can be used in the `SET` clause) `LOAD DATA INFILE`.

For information about how the optimizer handles subqueries, see [Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”](#). For a discussion of restrictions on subquery use, including performance issues for certain forms of subquery syntax, see [Section E.4, “Restrictions on Subqueries”](#).

13.2.10.1. The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication that it can be `NULL`, and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

The subquery in this `SELECT` returns a single value ('abcde') that has a data type of `CHAR`, a length of 5, a character set and collation equal to the defaults in effect at `CREATE TABLE` time, and an indication that the value in the column can be `NULL`. Nullability of the value selected by a scalar subquery is not copied because if the subquery result is empty, the result is `NULL`. For the subquery just shown, if `t1` were empty, the result would be `NULL` even though `s2` is `NOT NULL`.

There are a few contexts in which a scalar subquery cannot be used. If a statement permits only a literal value, you cannot use a subquery. For example, `LIMIT` requires literal integer arguments, and `LOAD DATA INFILE` requires a literal string file name. You cannot use subqueries to supply these values.

When you see examples in the following sections that contain the rather spartan construct (`SELECT column1 FROM t1`), imagine that your own code contains much more diverse and complex constructions.

Suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Then perform a `SELECT`:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is `2` because there is a row in `t2` containing a column `s1` that has a value of `2`.

A scalar subquery can be part of an expression, but remember the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.2.10.2. Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where *comparison_operator* is one of these operators:

```
= > < >= <= <> != <=>
```

For example:

```
... WHERE 'a' = (SELECT column1 FROM t1)
```

MySQL also permits this construct:

```
non_subquery_operand LIKE (subquery)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the rows in table `t1` for which the `column1` value is equal to a maximum value in table `t2`:

```
SELECT * FROM t1
WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one of the tables. It finds all rows in table `t1` containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison of the subquery to a scalar, the subquery must return a scalar. For a comparison of the subquery to a row constructor, the subquery must be a row subquery that returns a row with the same number of values as the row constructor. See [Section 13.2.10.5, “Row Subqueries”](#).

13.2.10.3. Subqueries with **ANY**, **IN**, or **SOME**

Syntax:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

Where *comparison_operator* is one of these operators:

```
= > < >= <= <> !=
```

The **ANY** keyword, which must follow a comparison operator, means “return **TRUE** if the comparison is **TRUE** for **ANY** of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is **TRUE** if table `t2` contains `(21,14,7)` because there is a value `7` in `t2` that is less than `10`. The expression is **FALSE** if table `t2` contains `(20,10)`, or if table `t2` is empty. The expression is *unknown* (that is, **NULL**) if table `t2` contains `(NULL,NULL,NULL)`.

When used with a subquery, the word **IN** is an alias for `= ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

IN and `= ANY` are not synonyms when used with an expression list. **IN** can take an expression list, but `= ANY` cannot. See [Section 12.3.2, “Comparison Functions and Operators”](#).

NOT IN is not an alias for `<> ANY`, but for `<> ALL`. See [Section 13.2.10.4, “Subqueries with ALL”](#).

The word **SOME** is an alias for **ANY**. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word **SOME** is rare, but this example shows why it might be useful. To most people, the English phrase “a is not equal to any b” means “there is no b which is equal to a,” but that is not what is meant by the SQL syntax. The syntax means “there is some b to which a is not equal.” Using `<> SOME` instead helps ensure that everyone understands the true meaning of the query.

13.2.10.4. Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word **ALL**, which must follow a comparison operator, means “return **TRUE** if the comparison is **TRUE** for **ALL** of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table **t1** containing (10). The expression is **TRUE** if table **t2** contains (-5, 0, +5) because 10 is greater than all three values in **t2**. The expression is **FALSE** if table **t2** contains (12, 6, NULL, -100) because there is a single value 12 in table **t2** that is greater than 10. The expression is *unknown* (that is, **NULL**) if table **t2** contains (0, NULL, 1).

Finally, the expression is **TRUE** if table **t2** is empty. So, the following expression is **TRUE** when table **t2** is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this expression is **NULL** when table **t2** is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following expression is **NULL** when table **t2** is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables containing NULL values* and *empty tables* are “edge cases.” When writing subqueries, always consider whether you have taken those two possibilities into account.

NOT IN is an alias for **<> ALL**. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.2.10.5. Row Subqueries

The discussion to this point has been of scalar or column subqueries; that is, subqueries that return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Legal operators for row subquery comparisons are:

```
= > < >= <= <> != <=>
```

Here are two examples:

```
SELECT * FROM t1
  WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
SELECT * FROM t1
  WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

For both queries, if the table **t2** contains a single row with **id = 10**, the subquery returns a single row. If this row has **col3** and **col4** values equal to the **col1** and **col2** values of any rows in **t1**, the **WHERE** expression is **TRUE** and each query returns those **t1** rows. If the **t2** row **col3** and **col4** values are not equal the **col1** and **col2** values of any **t1** row, the expression is **FALSE** and the query returns an empty result set. The expression is *unknown* (that is, **NULL**) if the subquery produces no rows. An error occurs if the subquery produces multiple rows because a row subquery can return at most one row.

The expressions `(1, 2)` and `ROW(1, 2)` are sometimes called *row constructors*. The two are equivalent. The row constructor and the row returned by the subquery must contain the same number of values.

A row constructor is used for comparisons with subqueries that return two or more columns. When a subquery returns a single column, this is regarded as a scalar value and not as a row, so a row constructor cannot be used with a subquery that does not return at least two columns. Thus, the following query fails with a syntax error:

```
SELECT * FROM t1 WHERE ROW(1) = (SELECT column1 FROM t2)
```

Row constructors are legal in other contexts. For example, the following two statements are semantically equivalent (and are handled in the same way by the optimizer):

```
SELECT * FROM t1 WHERE (column1, column2) = (1, 1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

The following query answers the request, “find all rows in table `t1` that also exist in table `t2`”:

```
SELECT column1, column2, column3
FROM t1
WHERE (column1, column2, column3) IN
      (SELECT column1, column2, column3 FROM t2);
```

13.2.10.6. Subqueries with `EXISTS` or `NOT EXISTS`

If a subquery returns any rows at all, `EXISTS subquery` is `TRUE`, and `NOT EXISTS subquery` is `FALSE`. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an `EXISTS` subquery starts with `SELECT *`, but it could begin with `SELECT 5` or `SELECT column1` or anything at all. MySQL ignores the `SELECT` list in such a subquery, so it makes no difference.

For the preceding example, if `t2` contains any rows, even rows with nothing but `NULL` values, the `EXISTS` condition is `TRUE`. This is actually an unlikely example because a `[NOT] EXISTS` subquery almost always contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

```
SELECT DISTINCT store_type FROM stores
WHERE EXISTS (SELECT * FROM cities_stores
              WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in no cities?

```
SELECT DISTINCT store_type FROM stores
WHERE NOT EXISTS (SELECT * FROM cities_stores
                  WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in all cities?

```
SELECT DISTINCT store_type FROM stores s1
WHERE NOT EXISTS (
  SELECT * FROM cities WHERE NOT EXISTS (
    SELECT * FROM cities_stores
    WHERE cities_stores.city = cities.city
    AND cities_stores.store_type = stores.store_type));
```

The last example is a double-nested `NOT EXISTS` query. That is, it has a `NOT EXISTS` clause within a `NOT EXISTS` clause. Formally, it answers the question “does a city exist with a store that is not in `Stores`”? But it is easier to say that a nested `NOT EXISTS` answers the question “is `x TRUE` for all `y`?”

13.2.10.7. Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1
WHERE column1 = ANY (SELECT column1 FROM t2
                     WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of `t1`, even though the subquery's `FROM` clause does not mention a table `t1`. So, MySQL looks outside the subquery, and finds `t1` in the outer query.

Suppose that table `t1` contains a row where `column1 = 5` and `column2 = 6`; meanwhile, table `t2` contains a row where `column1 = 5` and `column2 = 7`. The simple expression `... WHERE column1 = ANY (SELECT column1 FROM t2)` would be `TRUE`, but in this example, the `WHERE` clause within the subquery is `FALSE` (because `(5,6)` is not equal to `(5,7)`), so the expression as a whole is `FALSE`.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
WHERE x.column1 = (SELECT column1 FROM t2 AS x
                  WHERE x.column1 = (SELECT column1 FROM t3
                                    WHERE x.column2 = t3.column1));
```

In this statement, `x.column2` must be a column in table `t2` because `SELECT column1 FROM t2 AS x ...` renames `t2`. It is not a column in table `t1` because `SELECT column1 FROM t1 ...` is an outer query that is *farther out*.

For subqueries in `HAVING` or `ORDER BY` clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Aggregate functions in correlated subqueries may contain outer references, provided the function contains nothing but outer references, and provided the function is not contained in another function or expression.

13.2.10.8. Subqueries in the `FROM` Clause

Subqueries are legal in a `SELECT` statement's `FROM` clause. The actual syntax is:

```
SELECT ... FROM (subquery) [AS] name ...
```

The `[AS] name` clause is mandatory, because every table in a `FROM` clause must have a name. Any columns in the `subquery` select list must have unique names.

For the sake of illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here is how to use a subquery in the `FROM` clause, using the example table:

```
INSERT INTO t1 VALUES (1, '1', 1.0);
INSERT INTO t1 VALUES (2, '2', 2.0);
SELECT sb1, sb2, sb3
FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
WHERE sb1 > 1;
```

Result: 2, '2', 4.0.

Here is another example: Suppose that you want to know the average of a set of sums for a grouped table. This does not work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

However, this query provides the desired information:

```
SELECT AVG(sum_column1)
FROM (SELECT SUM(column1) AS sum_column1
      FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery (`sum_column1`) is recognized in the outer query.

Subqueries in the `FROM` clause can return a scalar, column, row, or table. Subqueries in the `FROM` clause cannot be correlated subqueries, unless used within the `ON` clause of a `JOIN` operation.

Subqueries in the `FROM` clause are executed even for the `EXPLAIN` statement (that is, derived temporary tables are materialized). This occurs because upper-level queries need information about all tables during the optimization phase, and the table represented by a subquery in the `FROM` clause is unavailable unless the subquery is executed.

It is possible under certain circumstances to modify table data using `EXPLAIN SELECT`. This can occur if the outer query accesses any tables and an inner query invokes a stored function that changes one or more rows of a table. Suppose that there are two tables `t1` and `t2` in database `d1`, created as shown here:

```
mysql> CREATE DATABASE d1;
Query OK, 1 row affected (0.00 sec)

mysql> USE d1;
Database changed

mysql> CREATE TABLE t1 (c1 INT);
Query OK, 0 rows affected (0.15 sec)

mysql> CREATE TABLE t2 (c1 INT);
Query OK, 0 rows affected (0.08 sec)
```

Now we create a stored function `f1` which modifies `t2`:

```
mysql> DELIMITER //
mysql> CREATE FUNCTION f1(p1 INT) RETURNS INT
mysql> BEGIN
mysql>     INSERT INTO t2 VALUES (p1);
mysql>     RETURN p1;
mysql> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

Referencing the function directly in an `EXPLAIN SELECT` does not have any effect on `t2`, as shown here:

```
mysql> SELECT * FROM t2;
Empty set (0.00 sec)

mysql> EXPLAIN SELECT f1(5);
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | NULL | No tables used |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

This is because the `SELECT` statement did not reference any tables, as can be seen in the `table` and `Extra` columns of the output. This is also true of the following nested `SELECT`:

```
mysql> EXPLAIN SELECT NOW() AS a1, (SELECT f1(5)) AS a2;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY    | NULL | NULL | NULL          | NULL | NULL    | NULL | NULL | No tables used |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note  | 1249 | Select 2 was reduced during optimization |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

However, if the outer `SELECT` references any tables, the optimizer executes the statement in the subquery as well:

```
mysql> EXPLAIN SELECT * FROM t1 AS a1, (SELECT f1(5)) AS a2;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table      | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | PRIMARY    | a1         | system | NULL          | NULL | NULL    | NULL | 0   | const row not found |
| 1  | PRIMARY    | <derived2> | system | NULL          | NULL | NULL    | NULL | 1   | |
| 2  | DERIVED    | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL | No tables used |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
+-----+
| c1 |
+-----+
| 5 |
+-----+
1 row in set (0.00 sec)
```

This also means that an `EXPLAIN SELECT` statement such as the one shown here may take a long time to execute because the `BENCHMARK()` [1190] function is executed once for each row in `t1`:

```
EXPLAIN SELECT * FROM t1 AS a1, (SELECT BENCHMARK(1000000, MD5(NOW())));
```

13.2.10.9. Subquery Errors

There are some errors that apply only to subqueries. This section describes them.

- Unsupported subquery syntax:

```
ERROR 1235 (ER_NOT_SUPPORTED_YET)
SQLSTATE = 42000
Message = "This version of MySQL doesn't yet support
'LIMIT & IN/ALL/ANY/SOME subquery'"
```

This means that MySQL does not support statements of the following form:

```
SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
```

- Incorrect number of columns from subquery:

```
ERROR 1241 (ER_OPERAND_COL)
SQLSTATE = 21000
Message = "Operand should contain 1 column(s)"
```

This error occurs in cases like this:

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

You may use a subquery that returns multiple columns, if the purpose is row comparison. In other contexts, the subquery must be a scalar operand. See [Section 13.2.10.5, "Row Subqueries"](#).

- Incorrect number of rows from subquery:

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

This error occurs for statements where the subquery must return at most one row but returns multiple rows. Consider the following example:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

If `SELECT column1 FROM t2` returns just one row, the previous query will work. If the subquery returns more than one row, error 1242 will occur. In that case, the query should be rewritten as:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Incorrectly used table in subquery:

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

This error occurs in cases such as the following, which attempts to modify a table and select from the same table in the subquery:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

You can use a subquery for assignment within an `UPDATE` statement because subqueries are legal in `UPDATE` and `DELETE` statements as well as in `SELECT` statements. However, you cannot use the same table (in this case, table `t1`) for both the subquery `FROM` clause and the update target.

For transactional storage engines, the failure of a subquery causes the entire statement to fail. For nontransactional storage engines, data modifications made before the error was encountered are preserved.

13.2.10.10. Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. The following list provides some interesting tricks that you might want to play with:

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

```
SELECT * FROM t1 WHERE t1.column1 IN
(SELECT column1 FROM t2 ORDER BY column1);
```

```
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
  (SELECT * FROM t2 LIMIT 1);
```

- Replace a join with a subquery. For example, try this:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
  SELECT column1 FROM t2);
```

Instead of this:

```
SELECT DISTINCT t1.column1 FROM t1, t2
  WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL that do not support subqueries. However, in some cases, converting a subquery to a join may improve performance. See [Section 13.2.10.11, “Rewriting Subqueries as Joins”](#).
- Move clauses from outside to inside the subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

For another example, use this query:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

Instead of this query:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use a row subquery instead of a correlated subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
              AND t2.column2=t1.column2);
```

- Use `NOT (a = ANY (...))` rather than `a <> ALL (...)`.
- Use `x = ANY (table containing (1,2))` rather than `x=1 OR x=2`.
- Use `= ANY` rather than `EXISTS`.
- For uncorrelated subqueries that always return one row, `IN` is always slower than `=`. For example, use this query:

```
SELECT * FROM t1
  WHERE t1.col_name = (SELECT a FROM t2 WHERE b = some_const);
```

Instead of this query:

```
SELECT * FROM t1
```

```
WHERE t1.col_name IN (SELECT a FROM t2 WHERE b = some_const);
```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the [BENCHMARK\(\)](#) [1190] function, you can get an idea about what helps in your own situation. See [Section 12.14, “Information Functions”](#).

Some optimizations that MySQL itself makes are:

- MySQL executes uncorrelated subqueries only once. Use [EXPLAIN](#) to make sure that a given subquery really is uncorrelated.
- MySQL rewrites [IN](#), [ALL](#), [ANY](#), and [SOME](#) subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which [EXPLAIN](#) describes as a special join type ([unique_subquery](#) [838] or [index_subquery](#) [838]):

```
... IN (SELECT indexed_column FROM single_table ...)
```

- MySQL enhances expressions of the following form with an expression involving [MIN\(\)](#) [1206] or [MAX\(\)](#) [1205], unless [NULL](#) values or empty sets are involved:

```
value {ALL|ANY|SOME} {> | < | >= | <=} (uncorrelated subquery)
```

For example, this [WHERE](#) clause:

```
WHERE 5 > ALL (SELECT x FROM t)
```

might be treated by the optimizer like this:

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

See also [MySQL Internals: How MySQL Transforms Subqueries](#).

13.2.10.11. Rewriting Subqueries as Joins

Sometimes there are other ways to test membership in a set of values than by using a subquery. Also, on some occasions, it is not only possible to rewrite a query without a subquery, but it can be more efficient to make use of some of these techniques rather than to use subqueries. One of these is the [IN\(\)](#) construct:

For example, this query:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten as:

```
SELECT table1.*
FROM table1 LEFT JOIN table2 ON table1.id=table2.id
WHERE table2.id IS NULL;
```

A [LEFT \[OUTER\] JOIN](#) can be faster than an equivalent subquery because the server might be able to optimize it better—a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not

exist, so subqueries were the only way to do certain things. Today, MySQL Server and many other modern database systems offer a wide range of outer join types.

MySQL Server supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported. See [Section 13.2.2, “DELETE Syntax”](#), and [Section 13.2.11, “UPDATE Syntax”](#).

13.2.11. UPDATE Syntax

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
  SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
  SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
  [WHERE where_condition]
```

For the single-table syntax, the `UPDATE` statement updates columns of existing rows in the named table with new values. The `SET` clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword `DEFAULT` to set a column explicitly to its default value. The `WHERE` clause, if given, specifies the conditions that identify which rows to update. With no `WHERE` clause, all rows are updated. If the `ORDER BY` clause is specified, the rows are updated in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, `UPDATE` updates rows in each table named in `table_references` that satisfy the conditions. In this case, `ORDER BY` and `LIMIT` cannot be used.

`where_condition` is an expression that evaluates to true for each row to be updated. For expression syntax, see [Section 9.5, “Expression Syntax”](#).

`table_references` and `where_condition` are specified as described in [Section 13.2.9, “SELECT Syntax”](#).

You need the `UPDATE [697]` privilege only for columns referenced in an `UPDATE` that are actually updated. You need only the `SELECT [697]` privilege for any columns that are read but not modified.

The `UPDATE` statement supports the following modifiers:

- With the `LOW_PRIORITY` keyword, execution of the `UPDATE` is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).
- With the `IGNORE` keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur are not updated. Rows for which columns are updated to values that would cause data conversion errors are updated to the closest valid values instead.

In MySQL 5.5.18 and later, `UPDATE IGNORE` statements, including those having an `ORDER BY` clause, are flagged as unsafe for statement-based replication. (This is because the order in which the rows are updated determines which rows are ignored.) With this change, such statements produce a warning in the log when using statement-based mode and are logged using the row-based format when using `MIXED` mode. (Bug #11758262, Bug #50439) See [Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”](#), for more information.

If you access a column from the table to be updated in an expression, `UPDATE` uses the current value of the column. For example, the following statement sets `col1` to one more than its current value:

```
UPDATE t1 SET col1 = col1 + 1;
```

The second assignment in the following statement sets `col2` to the current (updated) `col1` value, not the original `col1` value. The result is that `col1` and `col2` have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

Single-table `UPDATE` assignments are generally evaluated from left to right. For multiple-table updates, there is no guarantee that assignments are carried out in any particular order.

If you set a column to the value it currently has, MySQL notices this and does not update it.

If you update a column that has been declared `NOT NULL` by setting to `NULL`, an error occurs if strict SQL mode is enabled; otherwise, the column is set to the implicit default value for the column data type and the warning count is incremented. The implicit default value is `0` for numeric types, the empty string (`' '`) for string types, and the “zero” value for date and time types. See [Section 11.5, “Data Type Default Values”](#).

`UPDATE` returns the number of rows that were actually changed. The `mysql_info()` C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the `UPDATE`.

You can use `LIMIT row_count` to restrict the scope of the `UPDATE`. A `LIMIT` clause is a rows-matched restriction. The statement stops as soon as it has found `row_count` rows that satisfy the `WHERE` clause, whether or not they actually were changed.

If an `UPDATE` statement includes an `ORDER BY` clause, the rows are updated in the order specified by the clause. This can be useful in certain situations that might otherwise result in an error. Suppose that a table `t` contains a column `id` that has a unique index. The following statement could fail with a duplicate-key error, depending on the order in which rows are updated:

```
UPDATE t SET id = id + 1;
```

For example, if the table contains 1 and 2 in the `id` column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an `ORDER BY` clause to cause the rows with larger `id` values to be updated before those with smaller values:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

You can also perform `UPDATE` operations covering multiple tables. However, you cannot use `ORDER BY` or `LIMIT` with a multiple-table `UPDATE`. The `table_references` clause lists the tables involved in the join. Its syntax is described in [Section 13.2.9.2, “JOIN Syntax”](#). Here is an example:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The preceding example shows an inner join that uses the comma operator, but multiple-table `UPDATE` statements can use any type of join permitted in `SELECT` statements, such as `LEFT JOIN`.

If you use a multiple-table `UPDATE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the `ON UPDATE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly. See [Section 14.3.5.4, “FOREIGN KEY Constraints”](#).

Currently, you cannot update a table and select from the same table in a subquery.

Index hints (see [Section 13.2.9.3, “Index Hint Syntax”](#)) are accepted but ignored for `UPDATE` statements.

An `UPDATE` on a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks all partitions of the table. This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking. This issue is resolved in MySQL 5.6. See [Section 18.5.4, “Partitioning and Table-Level Locking”](#), for more information.

13.3. MySQL Transactional and Locking Statements

MySQL supports local transactions (within a given client session) through statements such as `SET autocommit`, `START TRANSACTION`, `COMMIT`, and `ROLLBACK`. See [Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#). XA transaction support enables MySQL to participate in distributed transactions as well. See [Section 13.3.7, “XA Transactions”](#).

13.3.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax

```
START TRANSACTION [WITH CONSISTENT SNAPSHOT]
BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

These statements provide control over use of [transactions](#):

- `START TRANSACTION` or `BEGIN` start a new transaction.
- `COMMIT` commits the current transaction, making its changes permanent.
- `ROLLBACK` rolls back the current transaction, canceling its changes.
- `SET autocommit` disables or enables the default autocommit mode for the current session.

By default, MySQL runs with `autocommit` mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent. The change cannot be rolled back.

To disable autocommit mode implicitly for a single series of statements, use the `START TRANSACTION` statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With `START TRANSACTION`, autocommit remains disabled until you end the transaction with `COMMIT` or `ROLLBACK`. The autocommit mode then reverts to its previous state.

You can also begin a transaction like this:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

The `WITH CONSISTENT SNAPSHOT` option starts a [consistent read](#) for storage engines that are capable of it. This applies only to `InnoDB`. The effect is the same as issuing a `START TRANSACTION` followed by a `SELECT` from any `InnoDB` table. See [Section 14.3.9.2, “Consistent Nonlocking Reads”](#). The `WITH CONSISTENT SNAPSHOT` option does not change the current transaction [isolation level](#), so it provides a consistent snapshot only if the current isolation level is one that permits consistent read (`REPEATABLE READ [1401]` or `SERIALIZABLE [1402]`).

**Important**

Many APIs used for writing MySQL client applications (such as JDBC) provide their own methods for starting transactions that can (and sometimes should) be used instead of sending a `START TRANSACTION` statement from the client. See [Chapter 22, Connectors and APIs](#), or the documentation for your API, for more information.

To disable autocommit mode explicitly, use the following statement:

```
SET autocommit=0;
```

After disabling autocommit mode by setting the `autocommit` [484] variable to zero, changes to transaction-safe tables (such as those for `InnoDB` or `NDBCLUSTER`) are not made permanent immediately. You must use `COMMIT` to store your changes to disk or `ROLLBACK` to ignore the changes.

`autocommit` [484] is a session variable and must be set for each session. To disable autocommit mode for each new connection, see the description of the `autocommit` [484] system variable at [Section 5.1.4, “Server System Variables”](#).

`BEGIN` and `BEGIN WORK` are supported as aliases of `START TRANSACTION` for initiating a transaction. `START TRANSACTION` is standard SQL syntax and is the recommended way to start an ad-hoc transaction.

The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN ... END` compound statement. The latter does not begin a transaction. See [Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”](#).

**Note**

Within all stored programs (stored procedures and functions, triggers, and events), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. Begin a transaction in this context with `START TRANSACTION` instead.

The optional `WORK` keyword is supported for `COMMIT` and `ROLLBACK`, as are the `CHAIN` and `RELEASE` clauses. `CHAIN` and `RELEASE` can be used for additional control over transaction completion. The value of the `completion_type` [489] system variable determines the default completion behavior. See [Section 5.1.4, “Server System Variables”](#).

The `AND CHAIN` clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The `RELEASE` clause causes the server to disconnect the current client session after terminating the current transaction. Including the `NO` keyword suppresses `CHAIN` or `RELEASE` completion, which can be useful if the `completion_type` [489] system variable is set to cause chaining or release completion by default.

Beginning a transaction causes any pending transaction to be committed. See [Section 13.3.3, “Statements That Cause an Implicit Commit”](#), for more information.

Beginning a transaction also causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

For best results, transactions should be performed using only tables managed by a single transaction-safe storage engine. Otherwise, the following problems can occur:

- If you use tables from more than one transaction-safe storage engine (such as `InnoDB`), and the transaction isolation level is not `SERIALIZABLE` [1402], it is possible that when one transaction commits, another ongoing transaction that uses the same tables will see only some of the changes

made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use `SET TRANSACTION ISOLATION LEVEL` to set the isolation level to `SERIALIZABLE` [1402] on a per-transaction basis as necessary.)

- If you use tables that are not transaction-safe within a transaction, changes to those tables are stored at once, regardless of the status of autocommit mode.
- If you issue a `ROLLBACK` statement after updating a nontransactional table within a transaction, an `ER_WARNING_NOT_COMPLETE_ROLLBACK` [3992] warning occurs. Changes to transaction-safe tables are rolled back, but not changes to nontransaction-safe tables.

Each transaction is stored in the binary log in one chunk, upon `COMMIT`. Transactions that are rolled back are not logged. (**Exception:** Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that modifications to the nontransactional tables are replicated.) See [Section 5.2.4, “The Binary Log”](#).

You can change the isolation level for transactions with the `SET TRANSACTION` statement. See [Section 13.3.6, “SET TRANSACTION Syntax”](#).

Rolling back can be a slow operation that may occur implicitly without the user having explicitly asked for it (for example, when an error occurs). Because of this, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the session, not only for explicit rollbacks performed with the `ROLLBACK` statement but also for implicit rollbacks.



Note

In MySQL 5.5, `BEGIN`, `COMMIT`, and `ROLLBACK` are not affected by `--replicate-do-db` [1875] or `--replicate-ignore-db` [1877] rules.

13.3.2. Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a `ROLLBACK` statement.

13.3.3. Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a `COMMIT` before executing the statement. As of MySQL 5.5.3, most of these statements also cause an implicit commit after executing; for additional details, see the end of this section.

- **Data definition language (DDL) statements that define or modify database objects.** `ALTER DATABASE . . . UPGRADE DATA DIRECTORY NAME`, `ALTER EVENT`, `ALTER PROCEDURE`, `ALTER TABLE`, `ALTER VIEW`, `CREATE DATABASE`, `CREATE EVENT`, `CREATE INDEX`, `CREATE PROCEDURE`, `CREATE TABLE`, `CREATE TRIGGER`, `CREATE VIEW`, `DROP DATABASE`, `DROP EVENT`, `DROP INDEX`, `DROP PROCEDURE`, `DROP TABLE`, `DROP TRIGGER`, `DROP VIEW`, `RENAME TABLE`, `TRUNCATE TABLE`.

`ALTER FUNCTION`, `CREATE FUNCTION` and `DROP FUNCTION` also cause an implicit commit when used with stored functions, but not with UDFs. (`ALTER FUNCTION` can only be used with stored functions.)

`ALTER TABLE`, `CREATE TABLE`, and `DROP TABLE` do not commit a transaction if the `TEMPORARY` keyword is used. (This does not apply to other operations on temporary tables such as `CREATE INDEX`, which do cause a commit.) However, although no implicit commit occurs, neither can the statement be rolled back. Therefore, use of such statements will violate transaction atomicity: For example, if you use `CREATE TEMPORARY TABLE` and then roll back the transaction, the table remains in existence.

The `CREATE TABLE` statement in `InnoDB` is processed as a single transaction. This means that a `ROLLBACK` from the user does not undo `CREATE TABLE` statements the user made during that transaction.

`CREATE TABLE ... SELECT` causes an implicit commit before and after the statement is executed when you are creating nontemporary tables. (No commit occurs for `CREATE TEMPORARY TABLE ... SELECT`.) This is to prevent an issue during replication where the table could be created on the master after a rollback, but fail to be recorded in the binary log, and therefore not replicated to the slave. For more information, see Bug #22865.

- **Statements that implicitly use or modify tables in the `mysql` database.** `CREATE USER`, `DROP USER`, `GRANT`, `RENAME USER`, `REVOKE`, `SET PASSWORD`.
- **Transaction-control and locking statements.** `BEGIN`, `LOCK TABLES`, `SET autocommit = 1` (if the value is not already 1), `START TRANSACTION`, `UNLOCK TABLES`.

`UNLOCK TABLES` commits a transaction only if any tables currently have been locked with `LOCK TABLES` to acquire nontransactional table locks. A commit does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table-level locks.

Transactions cannot be nested. This is a consequence of the implicit commit performed for any current transaction when you issue a `START TRANSACTION` statement or one of its synonyms.

Statements that cause an implicit commit cannot be used in an XA transaction while the transaction is in an `ACTIVE` state.

The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN ... END` compound statement. The latter does not cause an implicit commit. See [Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”](#).

- **Data loading statements.** `LOAD DATA INFILE`. `LOAD DATA INFILE` causes an implicit commit only for tables using the `NDB` storage engine. For more information, see Bug #11151.
- **Administrative statements.** `ANALYZE TABLE`, `CACHE INDEX`, `CHECK TABLE`, `LOAD INDEX INTO CACHE`, `OPTIMIZE TABLE`, `REPAIR TABLE`.

As of MySQL 5.5.3, most statements that previously caused an implicit commit before executing also do so after executing. The intent is to handle each such statement in its own special transaction because it cannot be rolled back anyway. The following list provides additional details pertaining to this change:

- The `CREATE TABLE` variants (`CREATE TABLE` for `InnoDB` tables and `CREATE TABLE ... SELECT`) that previously were special cases no longer are so because `CREATE TABLE` uniformly causes an implicit commit before and after executing.
- The `FLUSH` and `RESET` statements cause an implicit commit.
- Transaction-control and locking statements behave as before.

13.3.4. SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

InnoDB supports the SQL statements `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, `RELEASE SAVEPOINT` and the optional `WORK` keyword for `ROLLBACK`.

The `SAVEPOINT` statement sets a named transaction savepoint with a name of *identifier*. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The `ROLLBACK TO SAVEPOINT` statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but InnoDB does *not* release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the `ROLLBACK TO SAVEPOINT` statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1305 (42000): SAVEPOINT identifier does not exist
```

The `RELEASE SAVEPOINT` statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.

All savepoints of the current transaction are deleted if you execute a `COMMIT`, or a `ROLLBACK` that does not name a savepoint.

A new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

13.3.5. LOCK TABLES and UNLOCK TABLES Syntax

```
LOCK TABLES
  tbl_name [[AS] alias] lock_type
  [, tbl_name [[AS] alias] lock_type] ...

lock_type:
  READ [LOCAL]
  | [LOW_PRIORITY] WRITE

UNLOCK TABLES
```

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables. This is explained in more detail later in this section.

`LOCK TABLES` explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or views. You must have the `LOCK TABLES [696]` privilege, and the `SELECT [697]` privilege for each object to be locked.

For view locking, `LOCK TABLES` adds all base tables used in the view to the set of tables to be locked and locks them automatically. If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly, as described in [Section 13.3.5.2, “LOCK TABLES and Triggers”](#).

`UNLOCK TABLES` explicitly releases any table locks held by the current session. `LOCK TABLES` implicitly releases any table locks held by the current session before acquiring new locks.

Another use for `UNLOCK TABLES` is to release the global read lock acquired with the `FLUSH TABLES WITH READ LOCK` statement, which enables you to lock all tables in all databases. See [Section 13.7.6.3, “FLUSH Syntax”](#). (This is a very convenient way to get backups if you have a file system such as Veritas that can take snapshots in time.)

A table lock protects only against inappropriate reads or writes by other sessions. The session holding the lock, even a read lock, can perform table-level operations such as `DROP TABLE`. Truncate operations are not transaction-safe, so an error occurs if the session attempts one during an active transaction or while holding a table lock.

The following discussion applies only to non-`TEMPORARY` tables. `LOCK TABLES` is permitted (but ignored) for a `TEMPORARY` table. The table can be accessed freely by the session within which it was created, regardless of what other locking may be in effect. No lock is necessary because no other session can see the table.

For information about other conditions on the use of `LOCK TABLES` and statements that cannot be used while `LOCK TABLES` is in effect, see [Section 13.3.5.3, “Table-Locking Restrictions and Conditions”](#)

Rules for Lock Acquisition

To acquire table locks within the current session, use the `LOCK TABLES` statement. The following lock types are available:

`READ [LOCAL]` lock:

- The session that holds the lock can read the table (but not write it).
- Multiple sessions can acquire a `READ` lock for the table at the same time.
- Other sessions can read the table without explicitly acquiring a `READ` lock.
- The `LOCAL` modifier enables nonconflicting `INSERT` statements (concurrent inserts) by other sessions to execute while the lock is held. (See [Section 8.10.3, “Concurrent Inserts”](#).) However, `READ LOCAL` cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock. For `InnoDB` tables, `READ LOCAL` is the same as `READ`.

`[LOW_PRIORITY] WRITE` lock:

- The session that holds the lock can read and write the table.
- Only the session that holds the lock can access the table. No other session can access it until the lock is released.
- Lock requests for the table by other sessions block while the `WRITE` lock is held.
- The `LOW_PRIORITY` modifier has no effect as of MySQL 5.5.3. Before 5.5.3, it affects lock scheduling if the `WRITE` lock request must wait, as described later.

If the `LOCK TABLES` statement must wait due to locks held by other sessions on any of the tables, it blocks until all locks can be acquired.

A session that requires locks must acquire all the locks that it needs in a single `LOCK TABLES` statement. While the locks thus obtained are held, the session can access only the locked tables. For example, in the following sequence of statements, an error occurs for the attempt to access `t2` because it was not locked in the `LOCK TABLES` statement:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+-----+
| COUNT(*) |
+-----+
|          3 |
+-----+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

Tables in the `INFORMATION_SCHEMA` database are an exception. They can be accessed without being locked explicitly even while a session holds table locks obtained with `LOCK TABLES`.

You cannot refer to a locked table multiple times in a single query using the same name. Use aliases instead, and obtain a separate lock for the table and each alias:

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

The error occurs for the first `INSERT` because there are two references to the same name for a locked table. The second `INSERT` succeeds because the references to the table use different names.

If your statements refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

`WRITE` locks normally have higher priority than `READ` locks to ensure that updates are processed as soon as possible. This means that if one session obtains a `READ` lock and then another session requests a `WRITE` lock, subsequent `READ` lock requests wait until the session that requested the `WRITE` lock has obtained the lock and released it. Before MySQL 5.5.3, the `LOW_PRIORITY` modifier can be given to affect locking behavior as follows (as of 5.5.3, it has no effect): A request for a `LOW_PRIORITY WRITE` lock permits subsequent `READ` lock requests by other sessions to be satisfied first if they occur while the `LOW_PRIORITY WRITE` request is waiting. You should use `LOW_PRIORITY WRITE` locks only if you are sure that eventually there will be a time when no sessions have a `READ` lock. For `InnoDB` tables in transactional mode (`autocommit = 0`), a waiting `LOW_PRIORITY WRITE` lock acts like a regular `WRITE` lock and causes subsequent `READ` lock requests to wait.

`LOCK TABLES` acquires locks as follows:

1. Sort all tables to be locked in an internally defined order. From the user standpoint, this order is undefined.
2. If a table is to be locked with a read and a write lock, put the write lock request before the read lock request.
3. Lock one table at a time until the session gets all locks.

This policy ensures that table locking is deadlock free. There are, however, other things you need to be aware of about this policy: If you are using a `LOW_PRIORITY WRITE` lock for a table, it means only that

MySQL waits for this particular lock until there are no other sessions that want a [READ](#) lock. When the session has gotten the [WRITE](#) lock and is waiting to get the lock for the next table in the lock table list, all other sessions wait for the [WRITE](#) lock to be released. If this becomes a serious problem with your application, you should consider converting some of your tables to transaction-safe tables.



Note

[LOCK TABLES](#) or [UNLOCK TABLES](#), when applied to a partitioned table, always locks or unlocks the entire table. See [Section 18.5.4, “Partitioning and Table-Level Locking”](#).

Rules for Lock Release

When the table locks held by a session are released, they are all released at the same time. A session can release its locks explicitly, or locks may be released implicitly under certain conditions.

- A session can release its locks explicitly with [UNLOCK TABLES](#).
- If a session issues a [LOCK TABLES](#) statement to acquire a lock while already holding locks, its existing locks are released implicitly before the new locks are granted.
- If a session begins a transaction (for example, with [START TRANSACTION](#)), an implicit [UNLOCK TABLES](#) is performed, which causes existing locks to be released. (For additional information about the interaction between table locking and transactions, see [Section 13.3.5.1, “Interaction of Table Locking and Transactions”](#).)

If the connection for a client session terminates, whether normally or abnormally, the server implicitly releases all table locks held by the session (transactional and nontransactional). If the client reconnects, the locks will no longer be in effect. In addition, if the client had an active transaction, the server rolls back the transaction upon disconnect, and if reconnect occurs, the new session begins with autocommit enabled. For this reason, clients may wish to disable auto-reconnect. With auto-reconnect in effect, the client is not notified if reconnect occurs but any table locks or current transaction will have been lost. With auto-reconnect disabled, if the connection drops, an error occurs for the next statement issued. The client can detect the error and take appropriate action such as reacquiring the locks or redoing the transaction. See [Section 22.8.12, “Controlling Automatic Reconnection Behavior”](#).



Note

If you use [ALTER TABLE](#) on a locked table, it may become unlocked. For example, if you attempt a second [ALTER TABLE](#) operation, the result may be an error `Table 'tbl_name' was not locked with LOCK TABLES`. To handle this, lock the table again prior to the second alteration. See also [Section C.5.7.1, “Problems with ALTER TABLE”](#).

13.3.5.1. Interaction of Table Locking and Transactions

[LOCK TABLES](#) and [UNLOCK TABLES](#) interact with the use of transactions as follows:

- [LOCK TABLES](#) is not transaction-safe and implicitly commits any active transaction before attempting to lock the tables.
- [UNLOCK TABLES](#) implicitly commits any active transaction, but only if [LOCK TABLES](#) has been used to acquire table locks. For example, in the following set of statements, [UNLOCK TABLES](#) releases the global read lock but does not commit the transaction because no table locks are in effect:

```
FLUSH TABLES WITH READ LOCK;
START TRANSACTION;
```

```
SELECT ... ;
UNLOCK TABLES;
```

- Beginning a transaction (for example, with `START TRANSACTION`) implicitly commits any current transaction and releases existing table locks.
- `FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits. For example, `START TRANSACTION` does not release the global read lock. See [Section 13.7.6.3, “FLUSH Syntax”](#).
- Other statements that implicitly cause transactions to be committed do not release existing table locks. For a list of such statements, see [Section 13.3.3, “Statements That Cause an Implicit Commit”](#).
- The correct way to use `LOCK TABLES` and `UNLOCK TABLES` with transactional tables, such as `InnoDB` tables, is to begin a transaction with `SET autocommit = 0` (not `START TRANSACTION`) followed by `LOCK TABLES`, and to not call `UNLOCK TABLES` until you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

When you call `LOCK TABLES`, `InnoDB` internally takes its own table lock, and MySQL takes its own table lock. `InnoDB` releases its internal table lock at the next commit, but for MySQL to release its table lock, you have to call `UNLOCK TABLES`. You should not have `autocommit = 1` [484], because then `InnoDB` releases its internal table lock immediately after the call of `LOCK TABLES`, and deadlocks can very easily happen. `InnoDB` does not acquire the internal table lock at all if `autocommit = 1` [484], to help old applications avoid unnecessary deadlocks.

- `ROLLBACK` does not release table locks.

13.3.5.2. LOCK TABLES and Triggers

If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly:

- The locks are taken at the same time as those acquired explicitly with the `LOCK TABLES` statement.
- The lock on a table used in a trigger depends on whether the table is used only for reading. If so, a read lock suffices. Otherwise, a write lock is used.
- If a table is locked explicitly for reading with `LOCK TABLES`, but needs to be locked for writing because it might be modified within a trigger, a write lock is taken rather than a read lock. (That is, an implicit write lock needed due to the table's appearance within a trigger causes an explicit read lock request for the table to be converted to a write lock request.)

Suppose that you lock two tables, `t1` and `t2`, using this statement:

```
LOCK TABLES t1 WRITE, t2 READ;
```

If `t1` or `t2` have any triggers, tables used within the triggers will also be locked. Suppose that `t1` has a trigger defined like this:

```
CREATE TRIGGER t1_a_ins AFTER INSERT ON t1 FOR EACH ROW
BEGIN
  UPDATE t4 SET count = count+1
    WHERE id = NEW.id AND EXISTS (SELECT a FROM t3);
  INSERT INTO t2 VALUES(1, 2);
```

```
END;
```

The result of the `LOCK TABLES` statement is that `t1` and `t2` are locked because they appear in the statement, and `t3` and `t4` are locked because they are used within the trigger:

- `t1` is locked for writing per the `WRITE` lock request.
- `t2` is locked for writing, even though the request is for a `READ` lock. This occurs because `t2` is inserted into within the trigger, so the `READ` request is converted to a `WRITE` request.
- `t3` is locked for reading because it is only read from within the trigger.
- `t4` is locked for writing because it might be updated within the trigger.

13.3.5.3. Table-Locking Restrictions and Conditions

You can safely use `KILL` to terminate a session that is waiting for a table lock. See [Section 13.7.6.4, “KILL Syntax”](#).

You should *not* lock any tables that you are using with `INSERT DELAYED`. An `INSERT DELAYED` in this case results in an error because the insert must be handled by a separate thread, not by the session which holds the lock.

`LOCK TABLES` and `UNLOCK TABLES` cannot be used within stored programs.

Tables in the `performance_schema` database cannot be locked with `LOCK TABLES`, except the `setup_xxx` tables.

The following statements are prohibited while a `LOCK TABLES` statement is in effect:

- As of MySQL 5.5.3, `CREATE TABLE`, `CREATE TABLE . . . LIKE`, `CREATE VIEW`, `DROP VIEW`, and DDL statements on stored procedures and functions.
- As of MySQL 5.5.8, DDL statements on events

For some operations, system tables in the `mysql` database must be accessed. For example, the `HELP` statement requires the contents of the server-side help tables, and `CONVERT_TZ()` [1134] might need to read the time zone tables. The server implicitly locks the system tables for reading as necessary so that you need not lock them explicitly. These tables are treated as just described:

```
mysql.help_category
mysql.help_keyword
mysql.help_relation
mysql.help_topic
mysql.proc
mysql.time_zone
mysql.time_zone_leap_second
mysql.time_zone_name
mysql.time_zone_transition
mysql.time_zone_transition_type
```

If you want to explicitly place a `WRITE` lock on any of those tables with a `LOCK TABLES` statement, the table must be the only one locked; no other table can be locked with the same statement.

Normally, you do not need to lock tables, because all single `UPDATE` statements are atomic; no other session can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

- If you are going to run many operations on a set of `MyISAM` tables, it is much faster to lock the tables you are going to use. Locking `MyISAM` tables speeds up inserting, updating, or deleting on them because

MySQL does not flush the key cache for the locked tables until `UNLOCK TABLES` is called. Normally, the key cache is flushed after each SQL statement.

The downside to locking the tables is that no session can update a `READ`-locked table (including the one holding the lock) and no session can access a `WRITE`-locked table other than the one holding the lock.

- If you are using tables for a nontransactional storage engine, you must use `LOCK TABLES` if you want to ensure that no other session modifies the tables between a `SELECT` and an `UPDATE`. The example shown here requires `LOCK TABLES` to execute safely:

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

Without `LOCK TABLES`, it is possible that another session might insert a new row in the `trans` table between execution of the `SELECT` and `UPDATE` statements.

You can avoid using `LOCK TABLES` in many cases by using relative updates (`UPDATE customer SET value=value+new_value`) or the `LAST_INSERT_ID()` [1194] function. See [Section 1.8.5.3, “Transaction and Atomic Operation Differences”](#).

You can also avoid locking tables in some cases by using the user-level advisory lock functions `GET_LOCK()` [1199] and `RELEASE_LOCK()` [1201]. These locks are saved in a hash table in the server and implemented with `pthread_mutex_lock()` and `pthread_mutex_unlock()` for high speed. See [Section 12.15, “Miscellaneous Functions”](#).

See [Section 8.10.1, “Internal Locking Methods”](#), for more information on locking policy.

13.3.6. SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{
  REPEATABLE READ
  | READ COMMITTED
  | READ UNCOMMITTED
  | SERIALIZABLE
}
```

This statement sets the transaction isolation level, used for operations on `InnoDB` tables.

Scope of the Isolation Level

You can set the isolation level globally, for the current session, or for the next transaction:

- With the `GLOBAL` keyword, the statement sets the default transaction level globally for all subsequent sessions. Existing sessions are unaffected.
- With the `SESSION` keyword, the statement sets the default transaction level for all subsequent transactions performed within the current session.
- Without any `SESSION` or `GLOBAL` keyword, the statement sets the isolation level for the next (not started) transaction performed within the current session.

A change to the global default isolation level requires the `SUPER` [697] privilege. Any session is free to change its session isolation level (even in the middle of a transaction), or the isolation level for its next transaction.

`SET TRANSACTION ISOLATION LEVEL` without `GLOBAL` or `SESSION` is not permitted while there is an active transaction:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.02 sec)

mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ERROR 1568 (25001): Transaction isolation level can't be changed
while a transaction is in progress
```

To set the global default isolation level at server startup, use the `--transaction-isolation=level` [465] option to `mysqld` on the command line or in an option file. Values of `level` for this option use dashes rather than spaces, so the permissible values are `READ-UNCOMMITTED` [1402], `READ-COMMITTED` [1402], `REPEATABLE-READ` [1401], or `SERIALIZABLE` [1402]. For example, to set the default isolation level to `REPEATABLE READ` [1401], use these lines in the `[mysqld]` section of an option file:

```
[mysqld]
transaction-isolation = REPEATABLE-READ
```

It is possible to check or set the global and session transaction isolation levels at runtime by using the `tx_isolation` [570] system variable:

```
SELECT @@GLOBAL.tx_isolation, @@tx_isolation;
SET GLOBAL tx_isolation='REPEATABLE-READ';
SET SESSION tx_isolation='SERIALIZABLE';
```

Details and Usage of Isolation Levels

`InnoDB` supports each of the transaction isolation levels described here using different locking strategies. You can enforce a high degree of consistency with the default `REPEATABLE READ` [1401] level, for operations on crucial data where ACID compliance is important. Or you can relax the consistency rules with `READ COMMITTED` [1402] or even `READ UNCOMMITTED` [1402], in situations such as bulk reporting where precise consistency and repeatable results are less important than minimizing the amount of overhead for locking. `SERIALIZABLE` [1402] enforces even stricter rules than `REPEATABLE READ` [1401], and is used mainly in specialized situations, such as with XA transactions and for troubleshooting issues with concurrency and deadlocks.

For full information about how these isolation levels work with `InnoDB` transactions, see [Section 14.3.9, “The `InnoDB` Transaction Model and Locking”](#). In particular, for additional information about `InnoDB` record-level locks and how it uses them to execute various types of statements, see [Section 14.3.9.4, “`InnoDB` Record, Gap, and Next-Key Locks”](#) and [Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”](#).

The following list describes how MySQL supports the different transaction levels. The list goes from the most commonly used level to the least used.

- `REPEATABLE READ`

This is the default isolation level for `InnoDB`. For consistent reads, there is an important difference from the `READ COMMITTED` [1402] isolation level: All consistent reads within the same transaction read the snapshot established by the first read. This convention means that if you issue several plain (nonlocking) `SELECT` statements within the same transaction, these `SELECT` statements are consistent also with respect to each other. See [Section 14.3.9.2, “Consistent Nonlocking Reads”](#).

For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE`, and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition. For a unique index with a unique search condition, `InnoDB`

locks only the index record found, not the gap before it. For other search conditions, `InnoDB` locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range.

- `READ COMMITTED`

A somewhat Oracle-like isolation level with respect to consistent (nonlocking) reads: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See [Section 14.3.9.2, “Consistent Nonlocking Reads”](#).

For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `InnoDB` locks only index records, not the gaps before them, and thus permits the free insertion of new records next to locked records. For `UPDATE` and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition (such as `WHERE id = 100`), or a range-type search condition (such as `WHERE id > 100`). For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it. For range-type searches, `InnoDB` locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range. This is necessary because “phantom rows” must be blocked for MySQL replication and recovery to work.



Note

In MySQL 5.5, if the `READ COMMITTED` isolation level is used or the `innodb_locks_unsafe_for_binlog [1561]` system variable is enabled, there is no `InnoDB` gap locking except for foreign-key constraint checking and duplicate-key checking. Also, record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition.

If you use `READ COMMITTED` or enable `innodb_locks_unsafe_for_binlog [1561]`, you *must* use row-based binary logging.

- `READ UNCOMMITTED`

`SELECT` statements are performed in a nonlocking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a “dirty read.” Otherwise, this isolation level works like `READ COMMITTED [1402]`.

- `SERIALIZABLE`

This level is like `REPEATABLE READ [1401]`, but `InnoDB` implicitly converts all plain `SELECT` statements to `SELECT ... LOCK IN SHARE MODE` if autocommit is disabled. If autocommit is enabled, the `SELECT` is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (nonlocking) read and need not block for other transactions. (To force a plain `SELECT` to block if other transactions have modified the selected rows, disable autocommit.)

13.3.7. XA Transactions

Support for XA transactions is available for the `InnoDB` storage engine. The MySQL XA implementation is based on the X/Open CAE document *Distributed Transaction Processing: The XA Specification*. This document is published by The Open Group and available at <http://www.opengroup.org/public/pubs/catalog/c193.htm>. Limitations of the current XA implementation are described in [Section E.6, “Restrictions on XA Transactions”](#).

On the client side, there are no special requirements. The XA interface to a MySQL server consists of SQL statements that begin with the `XA` keyword. MySQL client programs must be able to send SQL statements

and to understand the semantics of the XA statement interface. They do not need be linked against a recent client library. Older client libraries also will work.

Currently, among the MySQL Connectors, MySQL Connector/J 5.0.0 supports XA directly (by means of a class interface that handles the Xan SQL statement interface for you).

XA supports distributed transactions; that is, the ability to permit multiple separate transactional resources to participate in a global transaction. Transactional resources often are RDBMSs but may be other kinds of resources.

A global transaction involves several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. (However, for a distributed transaction, you must use the [SERIALIZABLE \[1402\]](#) isolation level to achieve ACID properties. It is enough to use [REPEATABLE READ \[1401\]](#) for a nondistributed transaction, but not for a distributed transaction.)

Some examples of distributed transactions:

- An application may act as an integration tool that combines a messaging service with an RDBMS. The application makes sure that transactions dealing with message sending, retrieval, and processing that also involve a transactional database all happen in a global transaction. You can think of this as “transactional email.”
- An application performs actions that involve different database servers, such as a MySQL server and an Oracle server (or multiple MySQL servers), where actions that involve multiple servers must happen as part of a global transaction, rather than as separate transactions local to each server.
- A bank keeps account information in an RDBMS and distributes and receives money through automated teller machines (ATMs). It is necessary to ensure that ATM actions are correctly reflected in the accounts, but this cannot be done with the RDBMS alone. A global transaction manager integrates the ATM and database resources to ensure overall consistency of financial transactions.

Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

- A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.
- A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are “branches” of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

The MySQL implementation of XA MySQL enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.

To carry out a global transaction, it is necessary to know which components are involved, and bring each component to a point when it can be committed or rolled back. Depending on what each component reports about its ability to succeed, they must all commit or roll back as an atomic group. That is, either all components must commit, or all components must roll back. To manage a global transaction, it is necessary to take into account that any component or the connecting network might fail.

The process for executing a global transaction uses two-phase commit (2PC). This takes place after the actions performed by the branches of the global transaction have been executed.

1. In the first phase, all branches are prepared. That is, they are told by the TM to get ready to commit. Typically, this means each RM that manages a branch records the actions for the branch in stable storage. The branches indicate whether they are able to do this, and these results are used for the second phase.
2. In the second phase, the TM tells the RMs whether to commit or roll back. If all branches indicated when they were prepared that they will be able to commit, all branches are told to commit. If any branch indicated when it was prepared that it will not be able to commit, all branches are told to roll back.

In some cases, a global transaction might use one-phase commit (1PC). For example, when a Transaction Manager finds that a global transaction consists of only one transactional resource (that is, a single branch), that resource can be told to prepare and commit at the same time.

13.3.7.1. XA Transaction SQL Syntax

To perform XA transactions in MySQL, use the following statements:

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER
```

For `XA START`, the `JOIN` and `RESUME` clauses are not supported.

For `XA END` the `SUSPEND [FOR MIGRATE]` clause is not supported.

Each XA statement begins with the `XA` keyword, and most of them require an `xid` value. An `xid` is an XA transaction identifier. It indicates which transaction the statement applies to. `xid` values are supplied by the client, or generated by the MySQL server. An `xid` value has from one to three parts:

```
xid: gtrid [, bqual [, formatID ]]
```

`gtrid` is a global transaction identifier, `bqual` is a branch qualifier, and `formatID` is a number that identifies the format used by the `gtrid` and `bqual` values. As indicated by the syntax, `bqual` and `formatID` are optional. The default `bqual` value is `'` if not given. The default `formatID` value is 1 if not given.

`gtrid` and `bqual` must be string literals, each up to 64 bytes (not characters) long. `gtrid` and `bqual` can be specified in several ways. You can use a quoted string (`'ab'`), hex string (`0x6162`, `X'ab'`), or bit value (`b'nnnn'`).

`formatID` is an unsigned integer.

The `gtrid` and `bqual` values are interpreted in bytes by the MySQL server's underlying XA support routines. However, while an SQL statement containing an XA statement is being parsed, the server works with some specific character set. To be safe, write `gtrid` and `bqual` as hex strings.

`xid` values typically are generated by the Transaction Manager. Values generated by one TM must be different from values generated by other TMs. A given TM must be able to recognize its own `xid` values in a list of values returned by the `XA RECOVER` statement.

`XA START xid` starts an XA transaction with the given `xid` value. Each XA transaction must have a unique `xid` value, so the value must not currently be used by another XA transaction. Uniqueness is

assessed using the *gtrid* and *bqual* values. All following XA statements for the XA transaction must be specified using the same *xid* value as that given in the `XA START` statement. If you use any of those statements but specify an *xid* value that does not correspond to some existing XA transaction, an error occurs.

One or more XA transactions can be part of the same global transaction. All XA transactions within a given global transaction must use the same *gtrid* value in the *xid* value. For this reason, *gtrid* values must be globally unique so that there is no ambiguity about which global transaction a given XA transaction is part of. The *bqual* part of the *xid* value must be different for each XA transaction within a global transaction. (The requirement that *bqual* values be different is a limitation of the current MySQL XA implementation. It is not part of the XA specification.)

The `XA RECOVER` statement returns information for those XA transactions on the MySQL server that are in the `PREPARED` state. (See [Section 13.3.7.2, “XA Transaction States”](#).) The output includes a row for each such XA transaction on the server, regardless of which client started it.

`XA RECOVER` output rows look like this (for an example *xid* value consisting of the parts 'abc', 'def', and 7):

```
mysql> XA RECOVER;
+-----+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data |
+-----+-----+-----+-----+
|          7 |              3 |              3 | abcdef |
+-----+-----+-----+-----+
```

The output columns have the following meanings:

- *formatID* is the *formatID* part of the transaction *xid*
- *gtrid_length* is the length in bytes of the *gtrid* part of the *xid*
- *bqual_length* is the length in bytes of the *bqual* part of the *xid*
- *data* is the concatenation of the *gtrid* and *bqual* parts of the *xid*

13.3.7.2. XA Transaction States

An XA transaction progresses through the following states:

1. Use `XA START` to start an XA transaction and put it in the `ACTIVE` state.
2. For an `ACTIVE` XA transaction, issue the SQL statements that make up the transaction, and then issue an `XA END` statement. `XA END` puts the transaction in the `IDLE` state.
3. For an `IDLE` XA transaction, you can issue either an `XA PREPARE` statement or an `XA COMMIT ... ONE PHASE` statement:
 - `XA PREPARE` puts the transaction in the `PREPARED` state. An `XA RECOVER` statement at this point will include the transaction's *xid* value in its output, because `XA RECOVER` lists all XA transactions that are in the `PREPARED` state.
 - `XA COMMIT ... ONE PHASE` prepares and commits the transaction. The *xid* value will not be listed by `XA RECOVER` because the transaction terminates.
4. For a `PREPARED` XA transaction, you can issue an `XA COMMIT` statement to commit and terminate the transaction, or `XA ROLLBACK` to roll back and terminate the transaction.

Here is a simple XA transaction that inserts a row into a table as part of a global transaction:

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

Within the context of a given client connection, XA transactions and local (non-XA) transactions are mutually exclusive. For example, if `XA START` has been issued to begin an XA transaction, a local transaction cannot be started until the XA transaction has been committed or rolled back. Conversely, if a local transaction has been started with `START TRANSACTION`, no XA statements can be used until the transaction has been committed or rolled back.

Note that if an XA transaction is in the `ACTIVE` state, you cannot issue any statements that cause an implicit commit. That would violate the XA contract because you could not roll back the XA transaction. You will receive the following error if you try to execute such a statement:

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
when global transaction is in the ACTIVE state
```

Statements to which the preceding remark applies are listed at [Section 13.3.3, “Statements That Cause an Implicit Commit”](#).

13.4. Replication Statements

Replication can be controlled through the SQL interface using the statements described in this section. One group of statements controls master servers, the other controls slave servers.

13.4.1. SQL Statements for Controlling Master Servers

This section discusses statements for managing master replication servers. [Section 13.4.2, “SQL Statements for Controlling Slave Servers”](#), discusses statements for managing slave servers.

In addition to the statements described here, the following `SHOW` statements are used with master servers in replication. For information about these statements, see [Section 13.7.5, “SHOW Syntax”](#).

- `SHOW BINARY LOGS`
- `SHOW BINLOG EVENTS`
- `SHOW MASTER STATUS`
- `SHOW SLAVE HOSTS`

13.4.1.1. `PURGE BINARY LOGS` Syntax

```
PURGE { BINARY | MASTER } LOGS
      { TO 'log_name' | BEFORE datetime_expr }
```

The binary log is a set of files that contain information about data modifications made by the MySQL server. The log consists of a set of binary log files, plus an index file (see [Section 5.2.4, “The Binary Log”](#)).

The `PURGE BINARY LOGS` statement deletes all the binary log files listed in the log index file prior to the specified log file name or date. `BINARY` and `MASTER` are synonyms. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

This statement has no effect if the server was not started with the `--log-bin` [1892] option to enable binary logging.

Examples:

```
PURGE BINARY LOGS TO 'mysql-bin.010';
PURGE BINARY LOGS BEFORE '2008-04-02 22:46:26';
```

The `BEFORE` variant's `datetime_expr` argument should evaluate to a `DATETIME` value (a value in `'YYYY-MM-DD hh:mm:ss'` format).

This statement is safe to run while slaves are replicating. You need not stop them. If you have an active slave that currently is reading one of the log files you are trying to delete, this statement does nothing and fails with an error. However, if a slave is not connected and you happen to purge one of the log files it has yet to read, the slave will be unable to replicate after it reconnects.

To safely purge binary log files, follow this procedure:

1. On each slave server, use `SHOW SLAVE STATUS` to check which log file it is reading.
2. Obtain a listing of the binary log files on the master server with `SHOW BINARY LOGS`.
3. Determine the earliest log file among all the slaves. This is the target file. If all the slaves are up to date, this is the last log file on the list.
4. Make a backup of all the log files you are about to delete. (This step is optional, but always advisable.)
5. Purge all log files up to but not including the target file.

You can also set the `expire_logs_days` [498] system variable to expire binary log files automatically after a given number of days (see [Section 5.1.4, "Server System Variables"](#)). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master.

`PURGE BINARY LOGS TO` and `PURGE BINARY LOGS BEFORE` both fail with an error when binary log files listed in the `.index` file had been removed from the system by some other means (such as using `rm` on Linux). (Bug #18199, Bug #18453) To handle such errors, edit the `.index` file (which is a simple text file) manually to ensure that it lists only the binary log files that are actually present, then run again the `PURGE BINARY LOGS` statement that failed.

13.4.1.2. RESET MASTER Syntax

```
RESET MASTER
```

Deletes all binary log files listed in the index file, resets the binary log index file to be empty, and creates a new binary log file. This statement is intended to be used only when the master is started for the first time.



Important

The effects of `RESET MASTER` differ from those of `PURGE BINARY LOGS` in 2 key ways:

1. `RESET MASTER` removes *all* binary log files that are listed in the index file, leaving only a single, empty binary log file with a numeric suffix of `.000001`, whereas the numbering is not reset by `PURGE BINARY LOGS`.

2. `RESET MASTER` is *not* intended to be used while any replication slaves are running. The behavior of `RESET MASTER` when used while slaves are running is undefined (and thus unsupported), whereas `PURGE BINARY LOGS` may be safely used while replication slaves are running.

See also [Section 13.4.1.1](#), “`PURGE BINARY LOGS Syntax`”.

`RESET MASTER` can prove useful when you first set up the master and the slave, so that you can verify the setup as follows:

1. Start the master and slave, and start replication (see [Section 16.1.1](#), “`How to Set Up Replication`”).
2. Execute a few test queries on the master.
3. Check that the queries were replicated to the slave.
4. When replication is running correctly, issue `STOP SLAVE` followed by `RESET SLAVE` on the slave, then verify that any unwanted data no longer exists on the slave.
5. Issue `RESET MASTER` on the master to clean up the test queries.

After verifying the setup and getting rid of any unwanted and log files generated by testing, you can start the slave and begin replicating.

13.4.1.3. `SET sql_log_bin Syntax`

```
SET sql_log_bin = {0|1}
```

The `sql_log_bin` [555] variable controls whether logging to the binary log is done. The default value is 1 (do logging). To change logging for the current session, change the session value of this variable. The session user must have the `SUPER` [697] privilege to set this variable.

Beginning with MySQL 5.5.5, it is no longer possible to set `@@session.sql_log_bin` within a transaction or subquery. (Bug #53437)

13.4.2. SQL Statements for Controlling Slave Servers

This section discusses statements for managing slave replication servers. [Section 13.4.1](#), “`SQL Statements for Controlling Master Servers`”, discusses statements for managing master servers.

In addition to the statements described here, `SHOW SLAVE STATUS` and `SHOW RELAYLOG EVENTS` are also used with replication slaves. For information about these statements, see [Section 13.7.5.35](#), “`SHOW SLAVE STATUS Syntax`”, and [Section 13.7.5.33](#), “`SHOW RELAYLOG EVENTS Syntax`”.

13.4.2.1. `CHANGE MASTER TO Syntax`

```
CHANGE MASTER TO option [, option] ...
```

option:

```
MASTER_BIND = 'interface_name'
| MASTER_HOST = 'host_name'
| MASTER_USER = 'user_name'
| MASTER_PASSWORD = 'password'
| MASTER_PORT = port_num
| MASTER_CONNECT_RETRY = interval
| MASTER_HEARTBEAT_PERIOD = interval
| MASTER_LOG_FILE = 'master_log_name'
| MASTER_LOG_POS = master_log_pos
| RELAY_LOG_FILE = 'relay_log_name'
```

```

| RELAY_LOG_POS = relay_log_pos
| MASTER_SSL = {0|1}
| MASTER_SSL_CA = 'ca_file_name'
| MASTER_SSL_CAPATH = 'ca_directory_name'
| MASTER_SSL_CERT = 'cert_file_name'
| MASTER_SSL_KEY = 'key_file_name'
| MASTER_SSL_CIPHER = 'cipher_list'
| MASTER_SSL_VERIFY_SERVER_CERT = {0|1}
| IGNORE_SERVER_IDS = (server_id_list)

server_id_list:
[server_id [, server_id] ... ]

```

CHANGE MASTER TO changes the parameters that the slave server uses for connecting to the master server, for reading the master binary log, and reading the slave relay log. It also updates the contents of the `master.info` and `relay-log.info` files. To use **CHANGE MASTER TO**, the slave replication threads must be stopped (use **STOP SLAVE** if necessary).

Options not specified retain their value, except as indicated in the following discussion. Thus, in most cases, there is no need to specify options that do not change. For example, if the password to connect to your MySQL master has changed, you just need to issue these statements to tell the slave about the new password:

```

STOP SLAVE; -- if replication was running
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- if you want to restart replication

```

MASTER_HOST, **MASTER_USER**, **MASTER_PASSWORD**, and **MASTER_PORT** provide information to the slave about how to connect to its master:

- **MASTER_HOST** and **MASTER_PORT** are the host name (or IP address) of the master host and its TCP/IP port.



Note

Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

If you specify the **MASTER_HOST** or **MASTER_PORT** option, the slave assumes that the master server is different from before (even if the option value is the same as its current value.) In this case, the old values for the master binary log file name and position are considered no longer applicable, so if you do not specify **MASTER_LOG_FILE** and **MASTER_LOG_POS** in the statement, **MASTER_LOG_FILE=''** and **MASTER_LOG_POS=4** are silently appended to it.

Setting **MASTER_HOST=''** (that is, setting its value explicitly to an empty string) is *not* the same as not setting **MASTER_HOST** at all. Beginning with MySQL 5.5, trying to set **MASTER_HOST** to an empty string fails with an error. Previously, setting **MASTER_HOST** to an empty string caused **START SLAVE** subsequently to fail. (Bug #28796)

- **MASTER_USER** and **MASTER_PASSWORD** are the user name and password of the account to use for connecting to the master.

In MySQL 5.5.20 and later, **MASTER_USER** cannot be made empty; setting **MASTER_USER = ''** or leaving it unset when setting a value for **MASTER_PASSWORD** causes an error (Bug #13427949).

Currently, a password used for a replication slave account is effectively limited to 32 characters in length; the password can be longer, but any excess characters are truncated. This is not due to any limit imposed by the MySQL Server generally, but rather is an issue specific to MySQL Replication. (For more information, see Bug #43439.)

The text of a running `CHANGE MASTER TO` statement, including values for `MASTER_USER` and `MASTER_PASSWORD`, can be seen in the output of a concurrent `SHOW PROCESSLIST` statement.

The `MASTER_SSL_XXX` options provide information about using SSL for the connection. They correspond to the `--ssl-xxx` options described in [Section 6.3.8.4, “SSL Command Options”](#), and [Section 16.3.7, “Setting Up Replication Using SSL”](#). These options can be changed even on slaves that are compiled without SSL support. They are saved to the `master.info` file, but are ignored if the slave does not have SSL support enabled.

`MASTER_CONNECT_RETRY` specifies how many seconds to wait between connect retries. The default is 60. The *number* of reconnection attempts is limited by the `--master-retry-count` [\[1872\]](#) server option; for more information, see [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).

The `MASTER_BIND` option is available in MySQL Cluster NDB 7.2 and later, but is not supported in mainline MySQL 5.5.

`MASTER_BIND` is for use on replication slaves having multiple network interfaces, and determines which of the slave's network interfaces is chosen for connecting to the master.

`MASTER_HEARTBEAT_PERIOD` sets the interval in seconds between replication heartbeats. Whenever the master's binary log is updated with an event, the waiting period for the next heartbeat is reset. *interval* is a decimal value having the range 0 to 4294967 seconds and a resolution in milliseconds; the smallest nonzero value is 0.001. Heartbeats are sent by the master only if there are no unsent events in the binary log file for a period longer than *interval*.

Setting *interval* to 0 disables heartbeats altogether. The default value for *interval* is equal to the value of `slave_net_timeout` [\[1888\]](#) divided by 2.

Setting `@@global.slave_net_timeout` to a value less than that of the current heartbeat interval results in a warning being issued. The effect of issuing `RESET SLAVE` on the heartbeat interval is to reset it to the default value.

`MASTER_LOG_FILE` and `MASTER_LOG_POS` are the coordinates at which the slave I/O thread should begin reading from the master the next time the thread starts. `RELAY_LOG_FILE` and `RELAY_LOG_POS` are the coordinates at which the slave SQL thread should begin reading from the relay log the next time the thread starts. If you specify either of `MASTER_LOG_FILE` or `MASTER_LOG_POS`, you cannot specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. If neither of `MASTER_LOG_FILE` or `MASTER_LOG_POS` is specified, the slave uses the last coordinates of the *slave SQL thread* before `CHANGE MASTER TO` was issued. This ensures that there is no discontinuity in replication, even if the slave SQL thread was late compared to the slave I/O thread, when you merely want to change, say, the password to use.

`CHANGE MASTER TO` *deletes all relay log files* and starts a new one, unless you specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. In that case, relay log files are kept; the `relay_log_purge` [\[546\]](#) global variable is set silently to 0.

Prior to MySQL 5.5, `RELAY_LOG_FILE` required an absolute path. In MySQL 5.5, the path can be relative, in which case the path is assumed to be relative to the slave's data directory. (Bug #12190)

`IGNORE_SERVER_IDS` was added in MySQL 5.5. This option takes a comma-separated list of 0 or more server IDs. Events originating from the corresponding servers are ignored, with the exception of log rotation and deletion events, which are still recorded in the relay log.

In circular replication, the originating server normally acts as the terminator of its own events, so that they are not applied more than once. Thus, this option is useful in circular replication when one of the servers in the circle is removed. Suppose that you have a circular replication setup with 4 servers, having server

IDs 1, 2, 3, and 4, and server 3 fails. When bridging the gap by starting replication from server 2 to server 4, you can include `IGNORE_SERVER_IDS = (3)` in the `CHANGE MASTER TO` statement that you issue on server 4 to tell it to use server 2 as its master instead of server 3. Doing so causes it to ignore and not to propagate any statements that originated with the server that is no longer in use.

If a `CHANGE MASTER TO` statement is issued without any `IGNORE_SERVER_IDS` option, any existing list is preserved; `RESET SLAVE` also has no effect on the server ID list. To clear the list of ignored servers, it is necessary to use the option with an empty list:

```
CHANGE MASTER TO IGNORE_SERVER_IDS = ( );
```

If `IGNORE_SERVER_IDS` contains the server's own ID and the server was started with the `--replicate-same-server-id` [1879] option enabled, an error results.

Also beginning with MySQL 5.5, the `master.info` file and the output of `SHOW SLAVE STATUS` are extended to provide the list of servers that are currently ignored. For more information, see [Section 16.2.2.2, “Slave Status Logs”](#), and [Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”](#).

Beginning with MySQL 5.5.5, invoking `CHANGE MASTER TO` causes the previous values for `MASTER_HOST`, `MASTER_PORT`, `MASTER_LOG_FILE`, and `MASTER_LOG_POS` to be written to the error log, along with other information about the slave's state prior to execution.

`CHANGE MASTER TO` is useful for setting up a slave when you have the snapshot of the master and have recorded the master binary log coordinates corresponding to the time of the snapshot. After loading the snapshot into the slave to synchronize it to the slave, you can run `CHANGE MASTER TO MASTER_LOG_FILE='log_name', MASTER_LOG_POS=log_pos` on the slave to specify the coordinates at which the slave should begin reading the master binary log.

The following example changes the master server the slave uses and establishes the master binary log coordinates from which the slave begins reading. This is used when you want to set up the slave to replicate the master:

```
CHANGE MASTER TO
  MASTER_HOST='master2.mycompany.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='big3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

The next example shows an operation that is less frequently employed. It is used when the slave has relay log files that you want it to execute again for some reason. To do this, the master need not be reachable. You need only use `CHANGE MASTER TO` and start the SQL thread (`START SLAVE SQL_THREAD`):

```
CHANGE MASTER TO
  RELAY_LOG_FILE='slave-relay-bin.006',
  RELAY_LOG_POS=4025;
```

You can even use the second operation in a nonreplication setup with a standalone, nonslave server for recovery following a crash. Suppose that your server has crashed and you have restored it from a backup. You want to replay the server's own binary log files (not relay log files, but regular binary log files), named (for example) `myhost-bin.*`. First, make a backup copy of these binary log files in some safe place, in case you don't exactly follow the procedure below and accidentally have the server purge the binary log. Use `SET GLOBAL relay_log_purge=0` for additional safety. Then start the server without the `--log-bin` [1892] option. Instead, use the `--replicate-same-server-id` [1879], `--relay-log=myhost-bin` [1873] (to make the server believe that these regular binary log files are relay log files) and `--skip-slave-start` [1882] options. After the server starts, issue these statements:

```
CHANGE MASTER TO
  RELAY_LOG_FILE='myhost-bin.153',
  RELAY_LOG_POS=410,
  MASTER_HOST='some_dummy_string';
START SLAVE SQL_THREAD;
```

The server reads and executes its own binary log files, thus achieving crash recovery. Once the recovery is finished, run `STOP SLAVE`, shut down the server, delete the `master.info` and `relay-log.info` files, and restart the server with its original options.

Specifying the `MASTER_HOST` option (even with a dummy value) is required to make the server think it is a slave.

The following table shows the maximum permissible length for the string-valued options.

Option	Maximum Length
<code>MASTER_HOST</code>	60
<code>MASTER_USER</code>	16
<code>MASTER_PASSWORD</code>	32
<code>MASTER_LOG_FILE</code>	255
<code>RELAY_LOG_FILE</code>	255
<code>MASTER_SSL_CA</code>	255
<code>MASTER_SSL_CAPATH</code>	255
<code>MASTER_SSL_CERT</code>	255
<code>MASTER_SSL_KEY</code>	255
<code>MASTER_SSL_CIPHER</code>	511

13.4.2.2. `MASTER_POS_WAIT()` Syntax

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos [, timeout])
```

This is actually a function, not a statement. It is used to ensure that the slave has read and executed events up to a given position in the master's binary log. See [Section 12.15, "Miscellaneous Functions"](#), for a full description.

13.4.2.3. `RESET SLAVE` Syntax

```
RESET SLAVE [ALL]
```

`RESET SLAVE` makes the slave forget its replication position in the master's binary log. This statement is meant to be used for a clean start: It deletes the `master.info` and `relay-log.info` files, all the relay log files, and starts a new relay log file. To use `RESET SLAVE`, the slave replication threads must be stopped (use `STOP SLAVE` if necessary).



Note

All relay log files are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a `STOP SLAVE` statement or if the slave is highly loaded.)

In MySQL 5.5 (unlike the case in MySQL 5.1 and earlier), `RESET SLAVE` does not change any replication connection parameters such as master host, master port, master user, or master password, which are

retained in memory. This means that `START SLAVE` can be issued without requiring a `CHANGE MASTER TO` statement following `RESET SLAVE`.

Connection parameters are reset if the slave `mysqld` is shut down following `RESET SLAVE`. In MySQL 5.5.16 and later, you can instead use `RESET SLAVE ALL` to reset these connection parameters (Bug #11809016).

If the slave SQL thread was in the middle of replicating temporary tables when it was stopped, and `RESET SLAVE` is issued, these replicated temporary tables are deleted on the slave.

13.4.2.4. `SET GLOBAL sql_slave_skip_counter` Syntax

```
SET GLOBAL sql_slave_skip_counter = N
```

This statement skips the next *N* events from the master. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the slave threads are not running. Otherwise, it produces an error.

When using this statement, it is important to understand that the binary log is actually organized as a sequence of groups known as *event groups*. Each event group consists of a sequence of events.

- For transactional tables, an event group corresponds to a transaction.
- For nontransactional tables, an event group corresponds to a single SQL statement.



Note

A single transaction can contain changes to both transactional and nontransactional tables.

When you use `SET GLOBAL sql_slave_skip_counter` to skip events and the result is in the middle of a group, the slave continues to skip events until it reaches the end of the group. Execution then starts with the next event group.

Beginning with MySQL 5.5.5, issuing this statement causes the previous values of `RELAY_LOG_FILE`, `RELAY_LOG_POS`, and `sql_slave_skip_counter` [1890] to be written to the error log.

13.4.2.5. `START SLAVE` Syntax

```
START SLAVE [thread_types]

START SLAVE [SQL_THREAD] UNTIL
  MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos

START SLAVE [SQL_THREAD] UNTIL
  RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos

thread_types:
  [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

`START SLAVE` with no *thread_type* options starts both of the slave threads. The I/O thread reads events from the master server and stores them in the relay log. The SQL thread reads events from the relay log and executes them. `START SLAVE` requires the `SUPER` [697] privilege.

If `START SLAVE` succeeds in starting the slave threads, it returns without any error. However, even in that case, it might be that the slave threads start and then later stop (for example, because they do not manage

to connect to the master or read its binary log, or some other problem). `START SLAVE` does not warn you about this. You must check the slave's error log for error messages generated by the slave threads, or check that they are running satisfactorily with `SHOW SLAVE STATUS`.

`START SLAVE` sends an acknowledgment to the user after both the I/O thread and the SQL thread have started. However, the I/O thread may not yet have connected. For this reason, a successful `START SLAVE` causes `SHOW SLAVE STATUS` to show `Slave_SQL_Running=Yes`, but this does not guarantee that `Slave_IO_Running=Yes` (because `Slave_IO_Running=Yes` only if the I/O thread is running *and connected*). For more information, see [Section 13.7.5.35](#), “`SHOW SLAVE STATUS Syntax`”, and [Section 16.1.4.1](#), “`Checking Replication Status`”.

You can add `IO_THREAD` and `SQL_THREAD` options to the statement to name which of the threads to start.

An `UNTIL` clause may be added to specify that the slave should start and run until the SQL thread reaches a given point in the master binary log or in the slave relay log. When the SQL thread reaches that point, it stops. If the `SQL_THREAD` option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both slave threads. If the SQL thread is running, the `UNTIL` clause is ignored and a warning is issued.

For an `UNTIL` clause, you must specify both a log file name and position. Do not mix master and relay log options.

Any `UNTIL` condition is reset by a subsequent `STOP SLAVE` statement, a `START SLAVE` statement that includes no `UNTIL` clause, or a server restart.

The `UNTIL` clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the slave replicate an event. For example, if an unwise `DROP TABLE` statement was executed on the master, you can use `UNTIL` to tell the slave to execute up to that point but no farther. To find what the event is, use `mysqlbinlog` with the master binary log or slave relay log, or by using a `SHOW BINLOG EVENTS` statement.

If you are using `UNTIL` to have the slave process replicated queries in sections, it is recommended that you start the slave with the `--skip-slave-start` [1882] option to prevent the SQL thread from running when the slave server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The `SHOW SLAVE STATUS` statement includes output fields that display the current values of the `UNTIL` condition.

In old versions of MySQL (before 4.0.5), this statement was called `SLAVE START`. This usage is still accepted in MySQL 5.5 for backward compatibility, but is deprecated and is removed in MySQL 5.6.

13.4.2.6. `STOP SLAVE Syntax`

```
STOP SLAVE [thread_types]

thread_types:
  [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

Stops the slave threads. `STOP SLAVE` requires the `SUPER` [697] privilege. Recommended best practice is to execute `STOP SLAVE` on the slave before stopping the slave server (see [Section 5.1.12](#), “`The Shutdown Process`”, for more information).

When using the row-based logging format. You should execute `STOP SLAVE` on the slave prior to shutting down the slave server if you are replicating any tables that use a nontransactional storage engine (see the

Note later in this section). In MySQL 5.5.9 and later, you can also use `STOP SLAVE SQL_THREAD` for this purpose.

Like `START SLAVE`, this statement may be used with the `IO_THREAD` and `SQL_THREAD` options to name the thread or threads to be stopped.

**Note**

In MySQL 5.5, `STOP SLAVE` waits until the current replication event group affecting one or more non-transactional tables has finished executing (if there is any such replication group), or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. (Bug #319, Bug #38205)

In old versions of MySQL (before 4.0.5), this statement was called `SLAVE STOP`. This usage is still accepted in MySQL 5.5 for backward compatibility, but is deprecated and is removed in MySQL 5.6.

13.5. SQL Syntax for Prepared Statements

MySQL 5.5 provides support for server-side prepared statements. This support takes advantage of the efficient client/server binary protocol implemented in MySQL 4.1, provided that you use an appropriate client programming interface. Candidate interfaces include the MySQL C API client library (for C programs), MySQL Connector/J (for Java programs), and MySQL Connector/Net. For example, the C API provides a set of function calls that make up its prepared statement API. See [Section 22.8.4, “C API Prepared Statements”](#). Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library, one example being the `mysql_i` extension, available in PHP 5.0 and later.

An alternative SQL interface to prepared statements is available. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.
- You can use it from any program that enables you to send SQL statements to the server to be executed, such as the `mysql` client program.
- You can use it even if the client is using an old version of the client library. The only requirement is that you be able to connect to a server that is recent enough to support SQL syntax for prepared statements.

SQL syntax for prepared statements is intended to be used for situations such as these:

- You want to test how prepared statements work in your application before coding it.
- An application has problems executing prepared statements and you want to determine interactively what the problem is.
- You want to create a test case that describes a problem you are having with prepared statements, so that you can file a bug report.
- You need to use prepared statements but do not have access to a programming API that supports them.

SQL syntax for prepared statements is based on three SQL statements:

- `PREPARE` prepares a statement for execution (see [Section 13.5.1, “PREPARE Syntax”](#)).
- `EXECUTE` executes a prepared statement (see [Section 13.5.2, “EXECUTE Syntax”](#)).

- `DEALLOCATE PREPARE` releases a prepared statement (see [Section 13.5.3](#), “`DEALLOCATE PREPARE Syntax`”).

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|          5 |
+-----+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement as a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|          10 |
+-----+
mysql> DEALLOCATE PREPARE stmt2;
```

Here is an additional example that demonstrates how to choose the table on which to perform a query at runtime, by storing the name of the table as a user variable:

```
mysql> USE test;
mysql> CREATE TABLE t1 (a INT NOT NULL);
mysql> INSERT INTO t1 VALUES (4), (8), (11), (32), (80);

mysql> SET @table = 't1';
mysql> SET @s = CONCAT('SELECT * FROM ', @table);

mysql> PREPARE stmt3 FROM @s;
mysql> EXECUTE stmt3;
+-----+
| a |
+-----+
|  4 |
|  8 |
| 11 |
| 32 |
| 80 |
+-----+

mysql> DEALLOCATE PREPARE stmt3;
```

A prepared statement is specific to the session in which it was created. If you terminate a session without deallocating a previously prepared statement, the server deallocates it automatically.

A prepared statement is also global to the session. If you create a prepared statement within a stored routine, it is not deallocated when the stored routine ends.

To guard against too many prepared statements being created simultaneously, set the `max_prepared_stmt_count` [523] system variable. To prevent the use of prepared statements, set the value to 0.

The following SQL statements can be used as prepared statements:

```
ALTER TABLE
ANALYZE TABLE
CACHE INDEX
CALL
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
COMMIT
{CREATE | DROP} DATABASE
{CREATE | RENAME | DROP} USER
CREATE INDEX
CREATE TABLE
DELETE
DO
DROP INDEX
DROP TABLE
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
      | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
INSERT
INSTALL PLUGIN
KILL
LOAD INDEX INTO CACHE
OPTIMIZE TABLE
RENAME TABLE
REPAIR TABLE
REPLACE
RESET {MASTER | SLAVE | QUERY CACHE}
REVOKE
SELECT
SET
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
TRUNCATE TABLE
UNINSTALL PLUGIN
UPDATE
```

Other statements are not yet supported.

Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. Exceptions are noted in [Section E.1, “Restrictions on Stored Programs”](#).

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic reparation of the statement when it is next executed. For more information, see [Section 13.5.4, “Automatic Prepared Statement Reparation”](#).

Placeholders can be used for the arguments of the `LIMIT` clause when using prepared statements. See [Section 13.2.9, “SELECT Syntax”](#).

In prepared `CALL` statements used with `PREPARE` and `EXECUTE`, placeholder support for `OUT` and `INOUT` parameters is available beginning with MySQL 5.5. See [Section 13.2.1, “CALL Syntax”](#), for an example and a workaround for earlier versions. Placeholders can be used for `IN` parameters regardless of version.

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to `PREPARE` cannot itself be a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the `mysql_stmt_prepare()` C API function to prepare a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements can be used within stored procedures, but not in stored functions or triggers. However, a cursor cannot be used for a dynamic statement that is prepared and executed with `PREPARE` and `EXECUTE`. The statement for a cursor is checked at cursor creation time, so the statement cannot be dynamic.

SQL syntax for prepared statements does not support multi-statements (that is, multiple statements within a single string separated by “;” characters).

Prepared statements use the query cache under the conditions described in [Section 8.9.3.1, “How the Query Cache Operates”](#).

To write C programs that use the `CALL` SQL statement to execute stored procedures that contain prepared statements, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). For additional information, see [Section 13.2.1, “CALL Syntax”](#).

13.5.1. PREPARE Syntax

```
PREPARE stmt_name FROM preparable_stmt
```

The `PREPARE` statement prepares a statement and assigns it a name, `stmt_name`, by which to refer to the statement later. Statement names are not case sensitive. `preparable_stmt` is either a string literal or a user variable that contains the text of the statement. The text must represent a single SQL statement, not multiple statements. Within the statement, “?” characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The “?” characters should not be enclosed within quotation marks, even if you intend to bind them to string values. Parameter markers can be used only where data values should appear, not for SQL keywords, identifiers, and so forth.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

A prepared statement is executed with `EXECUTE` and released with `DEALLOCATE PREPARE`.

The scope of a prepared statement is the session within which it is created. Other sessions cannot see it.

For examples, see [Section 13.5, “SQL Syntax for Prepared Statements”](#).

13.5.2. EXECUTE Syntax

```
EXECUTE stmt_name
[USING @var_name [, @var_name] ...]
```

After preparing a statement with `PREPARE`, you execute it with an `EXECUTE` statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a `USING` clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the `USING` clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

For examples, see [Section 13.5, “SQL Syntax for Prepared Statements”](#).

13.5.3. DEALLOCATE PREPARE Syntax

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

To deallocate a prepared statement produced with `PREPARE`, use a `DEALLOCATE PREPARE` statement that refers to the prepared statement name. Attempting to execute a prepared statement after deallocating it results in an error.

For examples, see [Section 13.5, “SQL Syntax for Prepared Statements”](#).

13.5.4. Automatic Prepared Statement Repreparation

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic reparation of the statement when it is next executed. This applies to prepared statements processed at the SQL level (using the `PREPARE` statement) and those processed using the binary client/server protocol (using the `mysql_stmt_prepare()` C API function).

The server attempts reparation up to three times. An error occurs if all attempts fail.

Metadata changes occur for DDL statements such as those that create, drop, alter, rename, or truncate tables, or that analyze, optimize, or repair tables. Reparation also occurs after referenced tables or views are flushed from the table definition cache, either implicitly to make room for new entries in the cache, or explicitly due to `FLUSH TABLES`.

Table content changes (for example, with `INSERT` or `UPDATE`) do not cause reparation, nor do `SELECT` statements.

Reparation is automatic, but to the extent that it occurs, diminishes prepared statement performance.

Reparation uses the default database and SQL mode that were in effect for the original preparation.

The `Com_stmt_reprepare` [597] status variable tracks the number of reparations.

13.6. MySQL Compound-Statement Syntax

This section describes the syntax for the `BEGIN ... END` compound statement and other statements that can be used in the body of stored programs: Stored procedures and functions, triggers, and events. These objects are defined in terms of SQL code that is stored on the server for later invocation (see [Chapter 19, *Stored Programs and Views*](#)).

A compound statement is a block that can contain other blocks; declarations for variables, condition handlers, and cursors; and flow control constructs such as loops and conditional tests.

13.6.1. BEGIN ... END Compound-Statement Syntax

```
[begin_label:] BEGIN
  [statement_list]
END [end_label]
```

`BEGIN ... END` syntax is used for writing compound statements, which can appear within stored programs (stored procedures and functions, triggers, and events). A compound statement can contain

multiple statements, enclosed by the `BEGIN` and `END` keywords. `statement_list` represents a list of one or more statements, each terminated by a semicolon (`;`) statement delimiter. The `statement_list` itself is optional, so the empty compound statement (`BEGIN END`) is legal.

`BEGIN ... END` blocks can be nested.

Use of multiple statements requires that a client is able to send statement strings containing the `;` statement delimiter. In the `mysql` command-line client, this is handled with the `delimiter` command. Changing the `;` end-of-statement delimiter (for example, to `//`) permit `;` to be used in a program body. For an example, see [Section 19.1, “Defining Stored Programs”](#).

A `BEGIN ... END` block can be labeled. See [Section 13.6.2, “Statement Label Syntax”](#).

The optional `[NOT] ATOMIC` clause is not supported. This means that no transactional savepoint is set at the start of the instruction block and the `BEGIN` clause used in this context has no effect on the current transaction.



Note

Within all stored programs, the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. To begin a transaction in this context, use `START TRANSACTION` instead.

13.6.2. Statement Label Syntax

```
[begin_label:] BEGIN
  [statement_list]
END [end_label]

[begin_label:] LOOP
  statement_list
END LOOP [end_label]

[begin_label:] REPEAT
  statement_list
UNTIL search_condition
END REPEAT [end_label]

[begin_label:] WHILE search_condition DO
  statement_list
END WHILE [end_label]
```

Labels are permitted for `BEGIN ... END` blocks and for the `LOOP`, `REPEAT`, and `WHILE` statements. Label use for those statements follows these rules:

- `begin_label` must be followed by a colon.
- `begin_label` can be given without `end_label`. If `end_label` is present, it must be the same as `begin_label`.
- `end_label` cannot be given without `begin_label`.
- Labels at the same nesting level must be distinct.
- Labels can be up to 16 characters long.

To refer to a label within the labeled construct, use an `ITERATE` or `LEAVE` statement. The following example uses those statements to continue iterating or terminate the loop:

```
CREATE PROCEDURE doiterate(p1 INT)
```

```

BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
END;

```

The scope of a block label does not include the code for handlers declared within the block. For details, see [Section 13.6.7.2, “DECLARE . . . HANDLER Syntax”](#).

13.6.3. DECLARE Syntax

The `DECLARE` statement is used to define various items local to a program:

- Local variables. See [Section 13.6.4, “Variables in Stored Programs”](#).
- Conditions and handlers. See [Section 13.6.7, “Condition Handling”](#).
- Cursors. See [Section 13.6.6, “Cursors”](#).

`DECLARE` is permitted only inside a `BEGIN . . . END` compound statement and must be at its start, before any other statements.

Declarations must follow a certain order. Cursor declarations must appear before handler declarations. Variable and condition declarations must appear before cursor or handler declarations.

13.6.4. Variables in Stored Programs

System variables and user-defined variables can be used in stored programs, just as they can be used outside stored-program context. In addition, stored programs can use `DECLARE` to define local variables, and stored routines (procedures and functions) can be declared to take parameters that communicate values between the routine and its caller.

- To declare local variables, use the `DECLARE` statement, as described in [Section 13.6.4.1, “Local Variable DECLARE Syntax”](#).
- Variables can be set directly with the `SET` statement. See [Section 13.7.4, “SET Syntax”](#).
- Results from queries can be retrieved into local variables using `SELECT . . . INTO var_list` or by opening a cursor and using `FETCH . . . INTO var_list`. See [Section 13.2.9.1, “SELECT . . . INTO Syntax”](#), and [Section 13.6.6, “Cursors”](#).

For information about the scope of local variables and how MySQL resolves ambiguous names, see [Section 13.6.4.2, “Local Variable Scope and Resolution”](#).

13.6.4.1. Local Variable DECLARE Syntax

```

DECLARE var_name [, var_name] . . . type [DEFAULT value]

```

This statement declares local variables within stored programs. To provide a default value for a variable, include a `DEFAULT` clause. The value can be specified as an expression; it need not be a constant. If the `DEFAULT` clause is missing, the initial value is `NULL`.

Local variables are treated like stored routine parameters with respect to data type and overflow checking. See [Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).

Variable declarations must appear before cursor or handler declarations.

Local variable names are not case sensitive. Permissible characters and quoting rules are the same as for other identifiers, as described in [Section 9.2, “Schema Object Names”](#).

The scope of a local variable is the `BEGIN ... END` block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

13.6.4.2. Local Variable Scope and Resolution

The scope of a local variable is the `BEGIN ... END` block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

Local variables are in scope only during stored program execution. References to them are not permitted within prepared statements because those are global to the current session and the variables might have gone out of scope when the statement is executed. For example, `SELECT ... INTO local_var` cannot be used as a prepared statement.

A local variable should not have the same name as a table column. If an SQL statement, such as a `SELECT ... INTO` statement, contains a reference to a column and a declared local variable with the same name, MySQL currently interprets the reference as the name of a variable. Consider the following procedure definition:

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname, id INTO newname, xid
    FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

MySQL interprets `xname` in the `SELECT` statement as a reference to the `xname` variable rather than the `xname` column. Consequently, when the procedure `sp1()` is called, the `newname` variable returns the value `'bob'` regardless of the value of the `table1.xname` column.

Similarly, the cursor definition in the following procedure contains a `SELECT` statement that refers to `xname`. MySQL interprets this as a reference to the variable of that name rather than a column reference.

```
CREATE PROCEDURE sp2 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;
  DECLARE done TINYINT DEFAULT 0;
  DECLARE cur1 CURSOR FOR SELECT xname, id FROM table1;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur1;
  read_loop: LOOP
    FETCH FROM cur1 INTO newname, xid;
    IF done THEN LEAVE read_loop; END IF;
    SELECT newname;
  END LOOP;
  CLOSE cur1;
END;
```

See also [Section E.1, “Restrictions on Stored Programs”](#).

13.6.5. Flow Control Statements

MySQL supports the `IF`, `CASE`, `ITERATE`, `LEAVE LOOP`, `WHILE`, and `REPEAT` constructs for flow control within stored programs. It also supports `RETURN` within stored functions.

Many of these constructs contain other statements, as indicated by the grammar specifications in the following sections. Such constructs may be nested. For example, an `IF` statement might contain a `WHILE` loop, which itself contains a `CASE` statement.

MySQL does not support `FOR` loops.

13.6.5.1. CASE Syntax

```
CASE case_value
  WHEN when_value THEN statement_list
  [WHEN when_value THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

Or:

```
CASE
  WHEN search_condition THEN statement_list
  [WHEN search_condition THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

The `CASE` statement for stored programs implements a complex conditional construct.



Note

There is also a `CASE [1097] expression`, which differs from the `CASE statement` described here. See [Section 12.4, “Control Flow Functions”](#). The `CASE statement` cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

For the first syntax, *case_value* is an expression. This value is compared to the *when_value* expression in each `WHEN` clause until one of them is equal. When an equal *when_value* is found, the corresponding `THEN` clause *statement_list* executes. If no *when_value* is equal, the `ELSE` clause *statement_list* executes, if there is one.

This syntax cannot be used to test for equality with `NULL` because `NULL = NULL` is false. See [Section 3.3.4.6, “Working with NULL Values”](#).

For the second syntax, each `WHEN` clause *search_condition* expression is evaluated until one is true, at which point its corresponding `THEN` clause *statement_list* executes. If no *search_condition* is equal, the `ELSE` clause *statement_list* executes, if there is one.

If no *when_value* or *search_condition* matches the value tested and the `CASE` statement contains no `ELSE` clause, a `Case not found for CASE statement` error results.

Each *statement_list* consists of one or more SQL statements; an empty *statement_list* is not permitted.

To handle situations where no value is matched by any `WHEN` clause, use an `ELSE` containing an empty `BEGIN ... END` block, as shown in this example. (The indentation used here in the `ELSE` clause is for purposes of clarity only, and is not otherwise significant.)

```

DELIMITER |

CREATE PROCEDURE p()
BEGIN
  DECLARE v INT DEFAULT 1;

  CASE v
    WHEN 2 THEN SELECT v;
    WHEN 3 THEN SELECT 0;
    ELSE
      BEGIN
        END;
      END CASE;
  END;
|

```

13.6.5.2. IF Syntax

```

IF search_condition THEN statement_list
  [ELSEIF search_condition THEN statement_list] ...
  [ELSE statement_list]
END IF

```

The **IF** statement for stored programs implements a basic conditional construct.



Note

There is also an **IF()** [1097] *function*, which differs from the **IF statement** described here. See [Section 12.4, “Control Flow Functions”](#). The **IF** statement can have **THEN**, **ELSE**, and **ELSEIF** clauses, and it is terminated with **END IF**.

If the *search_condition* evaluates to true, the corresponding **THEN** or **ELSEIF** clause *statement_list* executes. If no *search_condition* matches, the **ELSE** clause *statement_list* executes.

Each *statement_list* consists of one or more SQL statements; an empty *statement_list* is not permitted.

An **IF ... END IF** block, like all other flow-control blocks used within stored programs, must be terminated with a semicolon, as shown in this example:

```

DELIMITER //

CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)

BEGIN
  DECLARE s VARCHAR(20);

  IF n > m THEN SET s = '>';
  ELSEIF n = m THEN SET s = '=';
  ELSE SET s = '<';
  END IF;

  SET s = CONCAT(n, ' ', s, ' ', m);

  RETURN s;
END //

DELIMITER ;

```

As with other flow-control constructs, **IF ... END IF** blocks may be nested within other flow-control constructs, including other **IF** statements. Each **IF** must be terminated by its own **END IF** followed by a

semicolon. You can use indentation to make nested flow-control blocks more easily readable by humans (although this is not required by MySQL), as shown here:

```
DELIMITER //

CREATE FUNCTION VerboseCompare (n INT, m INT)
  RETURNS VARCHAR(50)

  BEGIN
    DECLARE s VARCHAR(50);

    IF n = m THEN SET s = 'equals';
    ELSE
      IF n > m THEN SET s = 'greater';
      ELSE SET s = 'less';
      END IF;

    SET s = CONCAT('is ', s, ' than');
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m, '.');

    RETURN s;
  END //

DELIMITER ;
```

In this example, the inner `IF` is evaluated only if `n` is not equal to `m`.

13.6.5.3. ITERATE Syntax

```
ITERATE label
```

`ITERATE` can appear only within `LOOP`, `REPEAT`, and `WHILE` statements. `ITERATE` means “start the loop again.”

For an example, see [Section 13.6.5.5, “LOOP Syntax”](#).

13.6.5.4. LEAVE Syntax

```
LEAVE label
```

This statement is used to exit the flow control construct that has the given label. If the label is for the outermost stored program block, `LEAVE` exits the program.

`LEAVE` can be used within `BEGIN ... END` or loop constructs (`LOOP`, `REPEAT`, `WHILE`).

For an example, see [Section 13.6.5.5, “LOOP Syntax”](#).

13.6.5.5. LOOP Syntax

```
[begin_label:] LOOP
  statement_list
END LOOP [end_label]
```

`LOOP` implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon (`;`) statement delimiter. The statements within the loop are repeated until the loop is terminated. Usually, this is accomplished with a `LEAVE` statement. Within a stored function, `RETURN` can also be used, which exits the function entirely.

Neglecting to include a loop-termination statement results in an infinite loop.

A `LOOP` statement can be labeled. For the rules regarding label use, see [Section 13.6.2, “Statement Label Syntax”](#).

Example:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END;
```

13.6.5.6. REPEAT Syntax

```
[begin_label:] REPEAT
  statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a `REPEAT` statement is repeated until the `search_condition` expression is true. Thus, a `REPEAT` always enters the loop at least once. `statement_list` consists of one or more statements, each terminated by a semicolon (`;`) statement delimiter.

A `REPEAT` statement can be labeled. For the rules regarding label use, see [Section 13.6.2, “Statement Label Syntax”](#).

Example:

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
->   SET @x = 0;
->   REPEAT
->     SET @x = @x + 1;
->   UNTIL @x > p1 END REPEAT;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+-----+
| @x   |
+-----+
| 1001 |
+-----+
1 row in set (0.00 sec)
```

13.6.5.7. RETURN Syntax

```
RETURN expr
```

The `RETURN` statement terminates execution of a stored function and returns the value `expr` to the function caller. There must be at least one `RETURN` statement in a stored function. There may be more than one if the function has multiple exit points.

This statement is not used in stored procedures, triggers, or events. The `LEAVE` statement can be used to exit a stored program of those types.

13.6.5.8. `WHILE` Syntax

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

The statement list within a `WHILE` statement is repeated as long as the `search_condition` expression is true. `statement_list` consists of one or more SQL statements, each terminated by a semicolon (`;`) statement delimiter.

A `WHILE` statement can be labeled. For the rules regarding label use, see [Section 13.6.2, “Statement Label Syntax”](#).

Example:

```
CREATE PROCEDURE dowhile()
BEGIN
    DECLARE v1 INT DEFAULT 5;

    WHILE v1 > 0 DO
        ...
        SET v1 = v1 - 1;
    END WHILE;
END;
```

13.6.6. Cursors

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL. Cursors have these properties:

- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable
- Nonscrollable: Can be traversed only in one direction and cannot skip rows

Cursor declarations must appear before handler declarations and after variable and condition declarations.

Example:

```
CREATE PROCEDURE curdemo()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE a CHAR(16);
    DECLARE b, c INT;
    DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
    DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur1;
    OPEN cur2;

    read_loop: LOOP
        FETCH cur1 INTO a, b;
        FETCH cur2 INTO c;
        IF done THEN
            LEAVE read_loop;
        END IF;
        IF b < c THEN
            INSERT INTO test.t3 VALUES (a,b);
        
```

```
ELSE
  INSERT INTO test.t3 VALUES (a,c);
END IF;
END LOOP;

CLOSE cur1;
CLOSE cur2;
END;
```

13.6.6.1. Cursor **CLOSE** Syntax

```
CLOSE cursor_name
```

This statement closes a previously opened cursor. For an example, see [Section 13.6.6, “Cursors”](#).

An error occurs if the cursor is not open.

If not closed explicitly, a cursor is closed at the end of the `BEGIN ... END` block in which it was declared.

13.6.6.2. Cursor **DECLARE** Syntax

```
DECLARE cursor_name CURSOR FOR select_statement
```

This statement declares a cursor and associates it with a `SELECT` statement that retrieves the rows to be traversed by the cursor. To fetch the rows later, use a `FETCH` statement. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

The `SELECT` statement cannot have an `INTO` clause.

Cursor declarations must appear before handler declarations and after variable and condition declarations.

A stored program may contain multiple cursor declarations, but each cursor declared in a given block must have a unique name. For an example, see [Section 13.6.6, “Cursors”](#).

For information available through `SHOW` statements, it is possible in many cases to obtain equivalent information by using a cursor with an `INFORMATION_SCHEMA` table.

13.6.6.3. Cursor **FETCH** Syntax

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

This statement fetches the next row for the `SELECT` statement associated with the specified cursor (which must be open), and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

If no more rows are available, a No Data condition occurs with SQLSTATE value `'02000'`. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition). For an example, see [Section 13.6.6, “Cursors”](#).

13.6.6.4. Cursor **OPEN** Syntax

```
OPEN cursor_name
```

This statement opens a previously declared cursor. For an example, see [Section 13.6.6, “Cursors”](#).

13.6.7. Condition Handling

Conditions may arise during stored program execution that require special handling, such as exiting the current program block or continuing execution. Handlers can be defined for general conditions such as

warnings or exceptions, or for specific conditions such as a particular error code. Specific conditions can be assigned names and referred to that way in handlers.

To name a condition, use the `DECLARE ... CONDITION` statement. To declare a handler, use the `DECLARE ... HANDLER` statement. See [Section 13.6.7.1, “DECLARE ... CONDITION Syntax”](#), and [Section 13.6.7.2, “DECLARE ... HANDLER Syntax”](#).

To raise a condition, use the `SIGNAL` statement. To modify condition information within a condition handler, use `RESIGNAL`. See [Section 13.6.7.1, “DECLARE ... CONDITION Syntax”](#), and [Section 13.6.7.2, “DECLARE ... HANDLER Syntax”](#).

Another statement related to conditions is `GET DIAGNOSTICS`. The `GET DIAGNOSTICS` statement is not supported until MySQL 5.6.

Before MySQL 5.6.3, if a statement that generates a warning or error causes a condition handler to be invoked, the handler may not clear the diagnostic area. This might lead to the appearance that the handler was not invoked. The following discussion demonstrates the issue and provides a workaround.

Suppose that a table `t1` is empty. The following procedure selects from it, raising a No Data condition:

```
CREATE PROCEDURE p1()
BEGIN
  DECLARE a INT;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  BEGIN
    SET @handler_invoked = 1;
  END;
  SELECT c1 INTO a FROM t1;
END;
```

As can be seen from the following sequence of statements, the condition is not cleared by handler invocation (otherwise, the `SHOW WARNINGS` output would be empty). But as can be seen by the value of `@handler_invoked`, the handler was indeed invoked (otherwise its value would be 1).

```
mysql> SET @handler_invoked = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL p1();
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Warning | 1329 | No data - zero rows fetched, selected, or processed |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT @handler_invoked;
+-----+
| @handler_invoked |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

There are two ways to work around this issue:

- Add an extra dummy statement that clears warnings at the end of the condition handler:

```
CREATE PROCEDURE p1()
BEGIN
  DECLARE a INT;
```

```

DECLARE CONTINUE HANDLER FOR NOT FOUND
BEGIN
    SET @handler_invoked = 1;
    SELECT 1 FROM (SELECT 1) AS t;
END;
SELECT c1 INTO a FROM t1;
END;

```

This works for [CONTINUE](#) and [EXIT](#) handlers.

- Put an extra dummy statement that clears warnings following the statement that causes handler activation:

```

CREATE PROCEDURE p1()
BEGIN
    DECLARE a INT;
    DECLARE CONTINUE HANDLER FOR NOT FOUND
    BEGIN
        SET @handler_invoked = 1;
    END;
    SELECT c1 INTO a FROM t1;
    SELECT 1 FROM (SELECT 1) AS t;
END;

```

This works only for [CONTINUE](#) handlers.

This issue is resolved as of MySQL 5.6.3 and no workaround is needed.

13.6.7.1. [DECLARE ... CONDITION Syntax](#)

```

DECLARE condition_name CONDITION FOR condition_value

condition_value:
    mysql_error_code
    | SQLSTATE [VALUE] sqlstate_value

```

The [DECLARE ... CONDITION](#) statement declares a named error condition, associating a name with a condition that needs specific handling. The name can be referred to in a subsequent [DECLARE ... HANDLER](#) statement (see [Section 13.6.7.2, “DECLARE ... HANDLER Syntax”](#)).

Condition declarations must appear before cursor or handler declarations.

The *condition_value* for [DECLARE ... CONDITION](#) can be a MySQL error code (a number) or an SQLSTATE value (a 5-character string literal). You should not use MySQL error code 0 or SQLSTATE values that begin with '00', because those indicate success rather than an error condition. For a list of MySQL error codes and SQLSTATE values, see [Section C.3, “Server Error Codes and Messages”](#).

Using names for conditions can help make stored program code clearer. For example, this handler applies to attempts to drop a nonexistent table, but that is apparent only if you know the meaning of MySQL error code 1051:

```

DECLARE CONTINUE HANDLER FOR 1051
BEGIN
    -- body of handler
END;

```

By declaring a name for the condition, the purpose of the handler is more readily seen:

```

DECLARE no_such_table CONDITION FOR 1051;
DECLARE CONTINUE HANDLER FOR no_such_table
BEGIN
    -- body of handler
END;

```

Here is a named condition for the same condition, but based on the corresponding SQLSTATE value rather than the MySQL error code:

```
DECLARE no_such_table CONDITION FOR SQLSTATE '42S02';
DECLARE CONTINUE HANDLER FOR no_such_table
BEGIN
  -- body of handler
END;
```

Condition names referred to in [SIGNAL](#) or use [RESIGNAL](#) statements must be associated with SQLSTATE values, not MySQL error codes.

13.6.7.2. DECLARE ... HANDLER Syntax

```
DECLARE handler_action HANDLER
  FOR condition_value [, condition_value] ...
  statement

handler_action:
  CONTINUE
  | EXIT
  | UNDO

condition_value:
  mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
```

The `DECLARE ... HANDLER` statement specifies a handler that deals with one or more conditions. If one of these conditions occurs, the specified *statement* executes. *statement* can be a simple statement such as `SET var_name = value`, or a compound statement written using `BEGIN` and `END` (see [Section 13.6.1](#), “`BEGIN ... END Compound-Statement Syntax`”).

Handler declarations must appear after variable or condition declarations.

The *handler_action* value indicates what action the handler takes after execution of the handler statement:

- `CONTINUE`: Execution of the current program continues.
- `EXIT`: Execution terminates for the `BEGIN ... END` compound statement in which the handler is declared. This is true even if the condition occurs in an inner block.
- `UNDO`: Not supported.

The *condition_value* for `DECLARE ... HANDLER` indicates the specific condition or class of conditions that activates the handler:

- A MySQL error code (a number) or an SQLSTATE value (a 5-character string literal). You should not use MySQL error code 0 or SQLSTATE values that begin with '00', because those indicate success rather than an error condition. For a list of MySQL error codes and SQLSTATE values, see [Section C.3](#), “[Server Error Codes and Messages](#)”.
- A condition name previously specified with `DECLARE ... CONDITION`. A condition name can be associated with a MySQL error code or SQLSTATE value. See [Section 13.6.7.1](#), “`DECLARE ... CONDITION Syntax`”.
- `SQLWARNING` is shorthand for the class of SQLSTATE values that begin with '01'.

- `NOT FOUND` is shorthand for the class of SQLSTATE values that begin with `'02'`. This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. If no more rows are available, a No Data condition occurs with SQLSTATE value `'02000'`. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition). For an example, see [Section 13.6.6, “Cursors”](#). This condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.
- `SQLException` is shorthand for the class of SQLSTATE values that do not begin with `'00'`, `'01'`, or `'02'`.

If a condition occurs for which no handler has been declared, the action taken depends on the condition class:

- For `SQLException` conditions, the stored program terminates at the statement that raised the condition, as if there were an `EXIT` handler. If the program was called by another stored program, the calling program handles the condition using the handler selection rules applied to its own handlers.
- For `SQLWARNING` conditions, the program continues executing, as if there were a `CONTINUE` handler.
- For `NOT FOUND` conditions, if the condition was raised normally, the action is `CONTINUE`. If it was raised by `SIGNAL` or `RESIGNAL`, the action is `EXIT`.

The following example uses a handler for SQLSTATE `'23000'`, which occurs for a duplicate-key error:

```
mysql> CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
-> BEGIN
->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
->   SET @x = 1;
->   INSERT INTO test.t VALUES (1);
->   SET @x = 2;
->   INSERT INTO test.t VALUES (1);
->   SET @x = 3;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo();//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+-----+
| @x   |
+-----+
| 3    |
+-----+
1 row in set (0.00 sec)
```

Notice that `@x` is 3 after the procedure executes, which shows that execution continued to the end of the procedure after the error occurred. If the `DECLARE ... HANDLER` statement had not been present, MySQL would have taken the default action (`EXIT`) after the second `INSERT` failed due to the `PRIMARY KEY` constraint, and `SELECT @x` would have returned 2.

To ignore a condition, declare a `CONTINUE` handler for it and associate it with an empty block. For example:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

The scope of a block label does not include the code for handlers declared within the block. Therefore, the statement associated with a handler cannot use `ITERATE` or `LEAVE` to refer to labels for blocks that enclose the handler declaration. Consider the following example, where the `REPEAT` block has a label of `retry`:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  retry:
  REPEAT
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLWARNING
    BEGIN
      ITERATE retry;    # illegal
    END;
    IF i < 0 THEN
      LEAVE retry;      # legal
    END IF;
    SET i = i - 1;
  END;
  UNTIL FALSE END REPEAT;
END;
```

The `retry` label is in scope for the `IF` statement within the block. It is not in scope for the `CONTINUE` handler, so the reference there is invalid and results in an error:

```
ERROR 1308 (42000): LEAVE with no matching label: retry
```

To avoid references to outer labels in handlers, use one of these strategies:

- To leave the block, use an `EXIT` handler. If no block cleanup is required, the `BEGIN ... END` handler body can be empty:

```
DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;
```

Otherwise, put the cleanup statements in the handler body:

```
DECLARE EXIT HANDLER FOR SQLWARNING
BEGIN
  block cleanup statements
END;
```

- To continue execution, set a status variable in a `CONTINUE` handler that can be checked in the enclosing block to determine whether the handler was invoked. The following example uses the variable `done` for this purpose:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  DECLARE done INT DEFAULT FALSE;
  retry:
  REPEAT
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLWARNING
    BEGIN
      SET done = TRUE;
    END;
    IF done OR i < 0 THEN
      LEAVE retry;
    END IF;
    SET i = i - 1;
  END;
  UNTIL FALSE END REPEAT;
END;
```

13.6.7.3. RESIGNAL Syntax

```

RESIGNAL [condition_value]
  [SET signal_information_item
    [, signal_information_item] ...]

condition_value:
  SQLSTATE [VALUE] sqlstate_value
  | condition_name

signal_information_item:
  condition_information_item_name = simple_value_specification

condition_information_item_name:
  CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME

condition_name, simple_value_specification:
  (see following discussion)

```

RESIGNAL passes on the error condition information that is available during execution of a condition handler within a compound statement inside a stored procedure or function, trigger, or event. **RESIGNAL** may change some or all information before passing it on. **RESIGNAL** is related to **SIGNAL**, but instead of originating a condition as **SIGNAL** does, **RESIGNAL** relays existing condition information, possibly after modifying it.

RESIGNAL makes it possible to both handle an error and return the error information. Otherwise, by executing an SQL statement within the handler, information that caused the handler's activation is destroyed. **RESIGNAL** also can make some procedures shorter if a given handler can handle part of a situation, then pass the condition “up the line” to another handler.

No special privileges are required to execute the **RESIGNAL** statement.

For *condition_value* and *signal_information_item*, the definitions and rules are the same for **RESIGNAL** as for **SIGNAL** (see [Section 13.6.7.4, “SIGNAL Syntax”](#)).

The **RESIGNAL** statement takes *condition_value* and **SET** clauses, both of which are optional. This leads to several possible uses:

- **RESIGNAL** alone:

```
RESIGNAL;
```

- **RESIGNAL** with new signal information:

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

- **RESIGNAL** with a condition value and possibly new signal information:

```
RESIGNAL condition_value
  [SET signal_information_item [, signal_information_item] ...];
```

These use cases all cause changes to the diagnostics and condition areas:

- A diagnostics area contains one or more condition areas.
- A condition area contains condition information items, such as the `SQLSTATE` value, `MYSQL_ERRNO`, or `MESSAGE_TEXT`.

The maximum number of condition areas in a diagnostics area is determined by the value of the `max_error_count` [520] system variable.

13.6.7.3.1. `RESIGNAL` Alone

A simple `RESIGNAL` alone means “pass on the error with no change.” It restores the last diagnostics area and makes it the current diagnostics area. That is, it “pops” the diagnostics area stack.

Within a condition handler that catches a condition, one use for `RESIGNAL` alone is to perform some other actions, and then pass on without change the original condition information (the information that existed before entry into the handler).

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

The `DROP TABLE xx` statement fails. The diagnostics area stack looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

Then execution enters the `EXIT` handler. It starts by pushing the top of the diagnostics area stack, which now looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

Usually a procedure statement clears the first diagnostics area (also called the “current” diagnostics area). `BEGIN` is an exception, it does not clear, it does nothing. `SET` is not an exception, it clears, performs the operation, and then produces a result of “success.” The diagnostics area stack now looks like this:

```
DA 1. ERROR 0000 (00000): Successful operation
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

At this point, if `@a = 0`, `RESIGNAL` pops the diagnostics area stack, which now looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

And that is what the caller sees.

If `@a` is not 0, the handler simply ends, which means that there is no more use for the last diagnostics area (it has been “handled”), so it can be thrown away. The diagnostics area stack looks like this:

```
DA 1. ERROR 0000 (00000): Successful operation
```

The details make it look complex, but the end result is quite useful: Handlers can execute without destroying information about the condition that caused activation of the handler.

13.6.7.3.2. **RESIGNAL** with New Signal Information

RESIGNAL with a **SET** clause provides new signal information, so the statement means “pass on the error with changes”:

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

As with **RESIGNAL** alone, the idea is to pop the diagnostics area stack so that the original information will go out. Unlike **RESIGNAL** alone, anything specified in the **SET** clause changes.

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL SET MYSQL_ERRNO = 5; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

Remember from the previous discussion that **RESIGNAL** alone results in a diagnostics area stack like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

The **RESIGNAL SET MYSQL_ERRNO = 5** statement results in this stack instead:

```
DA 1. ERROR 5 (42S02): Unknown table 'xx'
```

In other words, it changes the error number, and nothing else.

The **RESIGNAL** statement can change any or all of the signal information items, making the first condition area of the diagnostics area look quite different.

13.6.7.3.3. **RESIGNAL** with a Condition Value and Optional New Signal Information

RESIGNAL with a condition value means “push a condition into the current diagnostics stack area.” If the **SET** clause is present, it also changes the error information.

```
RESIGNAL condition_value
[SET signal_information_item [, signal_information_item] ...];
```

This form of **RESIGNAL** restores the last diagnostics area and makes it the current diagnostics area. That is, it “pops” the diagnostics area stack, which is the same as what a simple **RESIGNAL** alone would do. However, it also changes the diagnostics area depending on the condition value or signal information.

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
```

```

BEGIN
  SET @error_count = @error_count + 1;
  IF @a = 0 THEN RESIGNAL SQLSTATE '45000' SET MYSQL_ERRNO=5; END IF;
END;
DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
SET @@max_error_count = 2;
CALL p();
SHOW ERRORS;

```

This is similar to the previous example, and the effects are the same, except that if `RESIGNAL` happens the current condition area looks different at the end. (The reason the condition is added rather than replaced is the use of a condition value.)

The `RESIGNAL` statement includes a condition value (`SQLSTATE '45000'`), so it “pushes” a new condition area, resulting in a diagnostics area stack that looks like this:

```

DA 1. (condition 1) ERROR 5 (45000) Unknown table 'xx'
      (condition 2) ERROR 1051 (42S02): Unknown table 'xx'

```

The result of `CALL p()` and `SHOW ERRORS` for this example is:

```

mysql> CALL p();
ERROR 5 (45000): Unknown table 'xx'
mysql> SHOW ERRORS;
+-----+-----+-----+
| Level | Code | Message                               |
+-----+-----+-----+
| Error | 5    | Unknown table 'xx'                   |
| Error | 1051 | Unknown table 'xx'                   |
+-----+-----+-----+

```

13.6.7.3.4. `RESIGNAL` Requires an Active Handler

All forms of `RESIGNAL` require that a handler be active when it executes. If no handler is active, `RESIGNAL` is illegal and a `resignal when handler not active` error occurs. For example:

```

mysql> CREATE PROCEDURE p () RESIGNAL;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL p();
ERROR 1739 (0K000): RESIGNAL when handler not active

```

Here is a more difficult example:

```

delimiter //
CREATE FUNCTION f () RETURNS INT
BEGIN
  RESIGNAL;
  RETURN 5;
END//
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION SET @a=f();
  SIGNAL SQLSTATE '55555';
END//
delimiter ;
CALL p();

```

At the time the `RESIGNAL` executes, there is a handler, even though the `RESIGNAL` is not defined inside the handler.

A statement such as the one following may appear bizarre because `RESIGNAL` apparently is not in a handler:

```
CREATE TRIGGER t_bi BEFORE INSERT ON t FOR EACH ROW RESIGNAL;
```

But it does not matter. `RESIGNAL` does not have to be technically “in” (that is, contained in), a handler declaration. The requirement is that a handler must be active.

13.6.7.4. SIGNAL Syntax

```
SIGNAL condition_value
  [SET signal_information_item
  [, signal_information_item] ...]

condition_value:
  SQLSTATE [VALUE] sqlstate_value
  | condition_name

signal_information_item:
  condition_information_item_name = simple_value_specification

condition_information_item_name:
  CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME

condition_name, simple_value_specification:
  (see following discussion)
```

`SIGNAL` is the way to “return” an error. `SIGNAL` provides error information to a handler, to an outer portion of the application, or to the client. Also, it provides control over the error's characteristics (error number, `SQLSTATE` value, message). Without `SIGNAL`, it is necessary to resort to workarounds such as deliberately referring to a nonexistent table to cause a routine to return an error.

No special privileges are required to execute the `SIGNAL` statement.

The *condition_value* in a `SIGNAL` statement indicates the error value to be returned. It can be an `SQLSTATE` value (a 5-character string literal) or a *condition_name* that refers to a named condition previously defined with `DECLARE ... CONDITION` (see Section 13.6.7.1, “`DECLARE ... CONDITION` Syntax”).

An `SQLSTATE` value can indicate errors, warnings, or “not found.” The first two characters of the value indicate its error class, as discussed in Section 13.6.7.4.1, “Signal Condition Information Items”. Some signal values cause statement termination; see Section 13.6.7.4.2, “Effect of Signals on Handlers, Cursors, and Statements”.

The `SQLSTATE` value for a `SIGNAL` statement should not start with '00' because such values indicate success and are not valid for signaling an error. This is true whether the `SQLSTATE` value is specified directly in the `SIGNAL` statement or in a named condition referred to in the statement. If the value is invalid, a `Bad SQLSTATE` error occurs.

To signal a generic `SQLSTATE` value, use '45000', which means “unhandled user-defined exception.”

The `SIGNAL` statement optionally includes a `SET` clause that contains multiple signal items, in a comma-separated list of `condition_information_item_name = simple_value_specification` assignments.

Each `condition_information_item_name` may be specified only once in the `SET` clause. Otherwise, a `Duplicate condition information item` error occurs.

Valid `simple_value_specification` designators can be specified using stored procedure or function parameters, stored program local variables declared with `DECLARE`, user-defined variables, system variables, or literals. A character literal may include a `_charset` introducer.

For information about permissible `condition_information_item_name` values, see [Section 13.6.7.4.1, “Signal Condition Information Items”](#).

The following procedure signals an error or warning depending on the value of `pval`, its input parameter:

```
CREATE PROCEDURE p (pval INT)
BEGIN
  DECLARE specialty CONDITION FOR SQLSTATE '45000';
  IF pval = 0 THEN
    SIGNAL SQLSTATE '01000';
  ELSEIF pval = 1 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An error occurred';
  ELSEIF pval = 2 THEN
    SIGNAL specialty
    SET MESSAGE_TEXT = 'An error occurred';
  ELSE
    SIGNAL SQLSTATE '01000'
    SET MESSAGE_TEXT = 'A warning occurred', MYSQL_ERRNO = 1000;
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An error occurred', MYSQL_ERRNO = 1001;
  END IF;
END;
```

If `pval` is 0, `p()` signals a warning because `SQLSTATE` values that begin with `'01'` are signals in the warning class. The warning does not terminate the procedure, and can be seen with `SHOW WARNINGS` after the procedure returns.

If `pval` is 1, `p()` signals an error and sets the `MESSAGE_TEXT` condition information item. The error terminates the procedure, and the text is returned with the error information.

If `pval` is 2, the same error is signaled, although the `SQLSTATE` value is specified using a named condition in this case.

If `pval` is anything else, `p()` first signals a warning and sets the message text and error number condition information items. This warning does not terminate the procedure, so execution continues and `p()` then signals an error. The error does terminate the procedure. The message text and error number set by the warning are replaced by the values set by the error, which are returned with the error information.

`SIGNAL` is typically used within stored programs, but it is a MySQL extension that it is permitted outside that context. For example, if you invoke the `mysql` client program, you can enter any of these statements at the prompt:

```
mysql> SIGNAL SQLSTATE '77777';
mysql> CREATE TRIGGER t_bi BEFORE INSERT ON t
-> FOR EACH ROW SIGNAL SQLSTATE '77777';
mysql> CREATE EVENT e ON SCHEDULE EVERY 1 SECOND
-> DO SIGNAL SQLSTATE '77777';
```

`SIGNAL` executes according to the following rules:

If the `SIGNAL` statement indicates a particular `SQLSTATE` value, that value is used to signal the condition specified. Example:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  IF divisor = 0 THEN
    SIGNAL SQLSTATE '22012';
  END IF;
END;
```

If the `SIGNAL` statement uses a named condition, the condition must be declared in some scope that applies to the `SIGNAL` statement, and must be defined using an `SQLSTATE` value, not a MySQL error number. Example:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE divide_by_zero CONDITION FOR SQLSTATE '22012';
  IF divisor = 0 THEN
    SIGNAL divide_by_zero;
  END IF;
END;
```

If the named condition does not exist in the scope of the `SIGNAL` statement, an `Undefined CONDITION` error occurs.

If `SIGNAL` refers to a named condition that is defined with a MySQL error number rather than an `SQLSTATE` value, a `SIGNAL/RESIGNAL` can only use a `CONDITION` defined with `SQLSTATE` error occurs. The following statements cause that error because the named condition is associated with a MySQL error number:

```
DECLARE no_such_table CONDITION FOR 1051;
SIGNAL no_such_table;
```

If a condition with a given name is declared multiple times in different scopes, the declaration with the most local scope applies. Consider the following procedure:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE my_error CONDITION FOR SQLSTATE '45000';
  IF divisor = 0 THEN
    BEGIN
      DECLARE my_error CONDITION FOR SQLSTATE '22012';
      SIGNAL my_error;
    END;
  END IF;
  SIGNAL my_error;
END;
```

If `divisor` is 0, the first `SIGNAL` statement executes. The innermost `my_error` condition declaration applies, raising `SQLSTATE '22012'`.

If `divisor` is not 0, the second `SIGNAL` statement executes. The outermost `my_error` condition declaration applies, raising `SQLSTATE '45000'`.

Signals can be raised within exception handlers:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SIGNAL SQLSTATE VALUE '99999'
    SET MESSAGE_TEXT = 'An error occurred';
  END;
  DROP TABLE no_such_table;
```

```
END;
```

`CALL p()` reaches the `DROP TABLE` statement. There is no table named `no_such_table`, so the error handler is activated. The error handler destroys the original error (“no such table”) and makes a new error with `SQLSTATE '99999'` and message `An error occurred`.

13.6.7.4.1. Signal Condition Information Items

The following table lists the names of diagnostics area condition information items that can be set in a `SIGNAL` (or `RESIGNAL`) statement. All items are standard SQL except `MYSQL_ERRNO`, which is a MySQL extension.

Item Name	Definition
CLASS_ORIGIN	VARCHAR(64)
SUBCLASS_ORIGIN	VARCHAR(64)
CONSTRAINT_CATALOG	VARCHAR(64)
CONSTRAINT_SCHEMA	VARCHAR(64)
CONSTRAINT_NAME	VARCHAR(64)
CATALOG_NAME	VARCHAR(64)
SCHEMA_NAME	VARCHAR(64)
TABLE_NAME	VARCHAR(64)
COLUMN_NAME	VARCHAR(64)
CURSOR_NAME	VARCHAR(64)
MESSAGE_TEXT	VARCHAR(128)
MYSQL_ERRNO	SMALLINT UNSIGNED

The character set for character items is UTF-8.

It is illegal to assign `NULL` to a condition information item in a `SIGNAL` statement.

A `SIGNAL` statement always specifies an `SQLSTATE` value, either directly, or indirectly by referring to a named condition defined with an `SQLSTATE` value. The first two characters of an `SQLSTATE` value are its class, and the class determines the default value for the condition information items:

- Class = '00' (success)

Illegal. `SQLSTATE` values that begin with '00' indicate success and are not valid for `SIGNAL`.

- Class = '01' (warning)

```
MESSAGE_TEXT = 'Unhandled user-defined warning';
MYSQL_ERRNO = ER_SIGNAL_WARN
```

- Class = '02' (not found)

```
MESSAGE_TEXT = 'Unhandled user-defined not found';
MYSQL_ERRNO = ER_SIGNAL_NOT_FOUND
```

- Class > '02' (exception)

```
MESSAGE_TEXT = 'Unhandled user-defined exception';
MYSQL_ERRNO = ER_SIGNAL_EXCEPTION
```

For legal classes, the other condition information items are set as follows:

```
CLASS_ORIGIN = SUBCLASS_ORIGIN = '';
CONSTRAINT_CATALOG = CONSTRAINT_SCHEMA = CONSTRAINT_NAME = '';
CATALOG_NAME = SCHEMA_NAME = TABLE_NAME = COLUMN_NAME = '';
CURSOR_NAME = '';
```

The error values that are accessible after `SIGNAL` executes are the `SQLSTATE` value raised by the `SIGNAL` statement and the `MESSAGE_TEXT` and `MYSQL_ERRNO` items. These values are available from the C API:

- `SQLSTATE` value: Call `mysql_sqlstate()`
- `MYSQL_ERRNO` value: Call `mysql_errno()`
- `MESSAGE_TEXT` value: Call `mysql_error()`

From SQL, the output from `SHOW WARNINGS` and `SHOW ERRORS` indicates the `MYSQL_ERRNO` and `MESSAGE_TEXT` values in the `Code` and `Message` columns.

Other condition information items can be set, but currently have no effect, in the sense that they are not accessible from error returns. For example, you can set `CLASS_ORIGIN` in a `SIGNAL` statement, but cannot see it after `SIGNAL` executes. In MySQL 5.6, condition information can be inspected with the `GET DIAGNOSTICS` statement.

13.6.7.4.2. Effect of Signals on Handlers, Cursors, and Statements

Signals have different effects on statement execution depending on the signal class. The class determines how severe an error is. MySQL ignores the value of the `sql_mode` [556] system variable; in particular, strict SQL mode does not matter. MySQL also ignores `IGNORE`: The intent of `SIGNAL` is to raise a user-generated error explicitly, so a signal is never ignored.

In the following descriptions, “unhandled” means that no handler for the signaled `SQLSTATE` value has been defined with `DECLARE ... HANDLER`.

- Class = '00' (success)

Illegal `SQLSTATE` values that begin with '00' indicate success and are not valid for `SIGNAL`.

- Class = '01' (warning)

The value of the `warning_count` [573] system variable goes up. `SHOW WARNINGS` shows the signal. `SQLWARNING` handlers catch the signal. If the signal is unhandled in a function, statements do not end.

- Class = '02' (not found)

`NOT FOUND` handlers catch the signal. There is no effect on cursors. If the signal is unhandled in a function, statements end.

- Class > '02' (exception)

`SQLEXCEPTION` handlers catch the signal. If the signal is unhandled in a function, statements end.

- Class = '40'

Treated as an ordinary exception.

Example:

```
mysql> delimiter //
mysql> CREATE FUNCTION f () RETURNS INT
-> BEGIN
-> SIGNAL SQLSTATE '01234'; -- signal a warning
-> RETURN 5;
-> END//
mysql> delimiter ;
mysql> CREATE TABLE t (s1 INT);
mysql> INSERT INTO t VALUES (f());
```

The result is that a row containing 5 is inserted into table `t`. The warning that is signaled can be viewed with `SHOW WARNINGS`.

13.7. Database Administration Statements

13.7.1. Account Management Statements

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in [Chapter 5, MySQL Server Administration](#), which you should consult for additional details.



Important

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

13.7.1.1. CREATE USER Syntax

```
CREATE USER user_specification
  [, user_specification] ...

user_specification:
  user
  [
    IDENTIFIED BY [PASSWORD] 'password'
  | IDENTIFIED WITH auth_plugin [AS 'auth_string']
  ]
```

The `CREATE USER` statement creates new MySQL accounts. To use it, you must have the global `CREATE USER [696]` privilege or the `INSERT [696]` privilege for the `mysql` database. For each account, `CREATE USER` creates a new row in the `mysql.user` table and assigns the account no privileges. An error occurs if the account already exists.

Each account name uses the format described in [Section 6.2.3, “Specifying Account Names”](#). For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.

The user specification may indicate how the user should authenticate when connecting to the server:

- To enable the user to connect with no password (which is *insecure*), include no `IDENTIFIED BY` clause:

```
CREATE USER 'jeffrey'@'localhost';
```

In this case, the account uses built-in authentication and clients must provide no password.

- To assign a password, use `IDENTIFIED BY` with the literal plaintext password value:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

The account uses built-in authentication and clients must match the given password.

- To avoid specifying the plaintext password if you know its hash value (the value that `PASSWORD() [1188]` would return for the password), specify the hash value preceded by the keyword `PASSWORD`:

```
CREATE USER 'jeffrey'@'localhost'
```

```
IDENTIFIED BY PASSWORD '*90E462C37378CED12064BB3388827D2BA3A9B689';
```

The account uses built-in authentication and clients must match the given password.

- To authenticate the account using a specific authentication plugin, use `IDENTIFIED WITH`, where `auth_plugin` is the plugin name. It can be an unquoted name or a quoted string literal. `'auth_string'` is an optional quoted string literal to pass to the plugin. The plugin interprets the meaning of the string, so its format is plugin specific. Consult the documentation for a given plugin for information about the authentication string values it accepts.

```
CREATE USER 'jeffrey'@'localhost'
IDENTIFIED WITH my_auth_plugin;
```

For connections that use this account, the server invokes the named plugin and clients must provide credentials as required for the authentication method that the plugin implements. If the server cannot find the plugin, either at account-creation time or connect time, an error occurs. `IDENTIFIED WITH` can be used as of MySQL 5.5.7.

The `IDENTIFIED BY` and `IDENTIFIED WITH` clauses are mutually exclusive, so at most one of them can be specified for a given user.

For additional information about setting passwords, see [Section 6.3.5, “Assigning Account Passwords”](#).



Important

`CREATE USER` may be recorded in server logs or in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. See [Section 6.1.2, “Keeping Passwords Secure”](#).



Important

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

13.7.1.2. DROP USER Syntax

```
DROP USER user [, user] ...
```

The `DROP USER` statement removes one or more MySQL accounts and their privileges. It removes privilege rows for the account from all grant tables. To use this statement, you must have the global `CREATE USER [696]` privilege or the `DELETE [696]` privilege for the `mysql` database. Each account name uses the format described in [Section 6.2.3, “Specifying Account Names”](#). For example:

```
DROP USER 'jeffrey'@'localhost';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.



Important

`DROP USER` does not automatically close any open user sessions. Rather, in the event that a user with an open session is dropped, the statement does not take effect until that user's session is closed. Once the session is closed, the user is dropped, and that user's next attempt to log in will fail. *This is by design.*

`DROP USER` does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the `DEFINER` attribute names the dropped user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see [Section 19.6, “Access Control for Stored Programs and Views”](#).)

13.7.1.3. GRANT Syntax

```

GRANT
  priv_type [(column_list)]
    [, priv_type [(column_list)] ...]
ON [object_type] priv_level
TO user_specification [, user_specification] ...
[REQUIRE {NONE | ssl_option [[AND] ssl_option] ...}]
[WITH with_option ...]

GRANT PROXY ON user_specification
  TO user_specification [, user_specification] ...
  [WITH GRANT OPTION]

object_type:
TABLE
| FUNCTION
| PROCEDURE

priv_level:
*
| *.*
| db_name.*
| db_name.tbl_name
| tbl_name
| db_name.routine_name

user_specification:
user
[
  IDENTIFIED BY [PASSWORD] 'password'
  | IDENTIFIED WITH auth_plugin [AS 'auth_string']
]

ssl_option:
SSL
| X509
| CIPHER 'cipher'
| ISSUER 'issuer'
| SUBJECT 'subject'

with_option:
GRANT OPTION
| MAX_QUERIES_PER_HOUR count
| MAX_UPDATES_PER_HOUR count
| MAX_CONNECTIONS_PER_HOUR count
| MAX_USER_CONNECTIONS count

```

The `GRANT` statement grants privileges to MySQL user accounts. `GRANT` also serves to specify other account characteristics such as use of secure connections and limits on access to server resources. To use `GRANT`, you must have the `GRANT OPTION [696]` privilege, and you must have the privileges that you are granting.

Normally, a database administrator first uses `CREATE USER` to create an account, then `GRANT` to define its privileges and characteristics. For example:

```

CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT SELECT ON db2.invoice TO 'jeffrey'@'localhost';

```

```
GRANT USAGE ON *.* TO 'jeffrey'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
```

However, if an account named in a `GRANT` statement does not already exist, `GRANT` may create it under the conditions described later in the discussion of the `NO_AUTO_CREATE_USER` [613] SQL mode.

The `REVOKE` statement is related to `GRANT` and enables administrators to remove account privileges. See [Section 13.7.1.5, “REVOKE Syntax”](#).

When successfully executed from the `mysql` program, `GRANT` responds with `Query OK, 0 rows affected`. To determine what privileges result from the operation, use `SHOW GRANTS`. See [Section 13.7.5.22, “SHOW GRANTS Syntax”](#).

There are several aspects to the `GRANT` statement, described under the following topics in this section:

- [Privileges Supported by MySQL](#)
- [Global Privileges](#)
- [Database Privileges](#)
- [Table Privileges](#)
- [Column Privileges](#)
- [Stored Routine Privileges](#)
- [Proxy User Privileges](#)
- [Account Names and Passwords](#)
- [Other Account Characteristics](#)
- [MySQL and Standard SQL Versions of GRANT](#)



Important

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

Privileges Supported by MySQL

The following table summarizes the permissible *priv_type* privilege types that can be specified for the `GRANT` and `REVOKE` statements. For additional information about these privileges, see [Section 6.2.1, “Privileges Provided by MySQL”](#).

Table 13.1. Permissible Privileges for `GRANT` and `REVOKE`

Privilege	Meaning
<code>ALL [PRIVILEGES]</code> [695]	Grant all privileges at specified access level except <code>GRANT OPTION</code> [696]
<code>ALTER</code> [695]	Enable use of <code>ALTER TABLE</code>
<code>ALTER ROUTINE</code> [695]	Enable stored routines to be altered or dropped
<code>CREATE</code> [695]	Enable database and table creation

Privilege	Meaning
CREATE ROUTINE [695]	Enable stored routine creation
CREATE TABLESPACE [695]	Enable tablespaces and log file groups to be created, altered, or dropped
CREATE TEMPORARY TABLES [695]	Enable use of CREATE TEMPORARY TABLE
CREATE USER [696]	Enable use of CREATE USER , DROP USER , RENAME USER , and REVOKE ALL PRIVILEGES
CREATE VIEW [696]	Enable views to be created or altered
DELETE [696]	Enable use of DELETE
DROP [696]	Enable databases, tables, and views to be dropped
EVENT [696]	Enable use of events for the Event Scheduler
EXECUTE [696]	Enable the user to execute stored routines
FILE [696]	Enable the user to cause the server to read or write files
GRANT OPTION [696]	Enable privileges to be granted to or removed from other accounts
INDEX [696]	Enable indexes to be created or dropped
INSERT [696]	Enable use of INSERT
LOCK TABLES [696]	Enable use of LOCK TABLES on tables for which you have the SELECT privilege
PROCESS [696]	Enable the user to see all processes with SHOW PROCESSLIST
PROXY [696]	Enable user proxying
REFERENCES [696]	Not implemented
RELOAD [696]	Enable use of FLUSH operations
REPLICATION CLIENT [697]	Enable the user to ask where master or slave servers are
REPLICATION SLAVE [697]	Enable replication slaves to read binary log events from the master
SELECT [697]	Enable use of SELECT
SHOW DATABASES [697]	Enable SHOW DATABASES to show all databases
SHOW VIEW [697]	Enable use of SHOW CREATE VIEW
SHUTDOWN [697]	Enable use of <code>mysqladmin shutdown</code>
SUPER [697]	Enable use of other administrative operations such as CHANGE MASTER TO , KILL , PURGE BINARY LOGS , SET GLOBAL , and <code>mysqladmin debug</code> command
TRIGGER [697]	Enable trigger operations
UPDATE [697]	Enable use of UPDATE
USAGE [697]	Synonym for “no privileges”

The [PROXY \[696\]](#) privilege was added in MySQL 5.5.7.

A trigger is associated with a table, so to create or drop a trigger, you must have the [TRIGGER \[697\]](#) privilege for the table, not the trigger.

In [GRANT](#) statements, the [ALL \[PRIVILEGES\] \[695\]](#) or [PROXY \[696\]](#) privilege must be named by itself and cannot be specified along with other privileges. [ALL \[PRIVILEGES\] \[695\]](#) stands for all privileges available for the level at which privileges are to be granted except for the [GRANT OPTION \[696\]](#) and [PROXY \[696\]](#) privileges.

`USAGE` [697] can be specified to create a user that has no privileges, or to specify the `REQUIRE` or `WITH` clauses for an account without changing its existing privileges.

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in [Section 6.2, “The MySQL Access Privilege System”](#), which you should consult for additional details.

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` [518] system variable is set to a nonzero value, `REVOKE` cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (`GRANT` will not create such rows when `lower_case_table_names` [518] is set, but such rows might have been created prior to setting that variable.)

Privileges can be granted at several levels, depending on the syntax used for the `ON` clause. For `REVOKE`, the same `ON` syntax specifies which privileges to take away. The examples shown here include no `IDENTIFIED BY 'password'` clause for brevity, but you should include one if the account does not already exist, to avoid creating an insecure account that has no password.

Global Privileges

Global privileges are administrative or apply to all databases on a given server. To assign global privileges, use `ON *.*` syntax:

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

The `CREATE TABLESPACE` [695], `CREATE USER` [696], `FILE` [696], `PROCESS` [696], `RELOAD` [696], `REPLICATION CLIENT` [697], `REPLICATION SLAVE` [697], `SHOW DATABASES` [697], `SHUTDOWN` [697], and `SUPER` [697] privileges are administrative and can only be granted globally.

Other privileges can be granted globally or at more specific levels.

MySQL stores global privileges in the `mysql.user` table.

Database Privileges

Database privileges apply to all objects in a given database. To assign database-level privileges, use `ON db_name.*` syntax:

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

If you use `ON *` syntax (rather than `ON *.*`) and you have selected a default database, privileges are assigned at the database level for the default database. An error occurs if there is no default database.

The `CREATE` [695], `DROP` [696], `EVENT` [696], `GRANT OPTION` [696], and `LOCK TABLES` [696] privileges can be specified at the database level. Table or routine privileges also can be specified at the database level, in which case they apply to all tables or routines in the database.

MySQL stores database privileges in the `mysql.db` table.

Table Privileges

Table privileges apply to all columns in a given table. To assign table-level privileges, use `ON db_name.tbl_name` syntax:

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
```

```
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

If you specify *tbl_name* rather than *db_name.tbl_name*, the statement applies to *tbl_name* in the default database. An error occurs if there is no default database.

The permissible *priv_type* values at the table level are ALTER [695], CREATE VIEW [696], CREATE [695], DELETE [696], DROP [696], GRANT OPTION [696], INDEX [696], INSERT [696], SELECT [697], SHOW VIEW [697], TRIGGER [697], and UPDATE [697].

MySQL stores table privileges in the `mysql.tables_priv` table.

Column Privileges

Column privileges apply to single columns in a given table. Each privilege to be granted at the column level must be followed by the column or columns, enclosed within parentheses.

```
GRANT SELECT (col1), INSERT (col1,col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

The permissible *priv_type* values for a column (that is, when you use a *column_list* clause) are INSERT [696], SELECT [697], and UPDATE [697].

MySQL stores column privileges in the `mysql.columns_priv` table.

Stored Routine Privileges

The ALTER ROUTINE [695], CREATE ROUTINE [695], EXECUTE [696], and GRANT OPTION [696] privileges apply to stored routines (procedures and functions). They can be granted at the global and database levels. Except for CREATE ROUTINE [695], these privileges can be granted at the routine level for individual routines.

```
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

The permissible *priv_type* values at the routine level are ALTER ROUTINE [695], EXECUTE [696], and GRANT OPTION [696]. CREATE ROUTINE [695] is not a routine-level privilege because you must have this privilege to create a routine in the first place.

MySQL stores routine-level privileges in the `mysql.procs_priv` table.

Proxy User Privileges

The PROXY [696] privilege enables one user to be a proxy for another. The proxy user impersonates or takes the identity of the proxied user.

```
GRANT PROXY ON 'localuser'@'localhost' TO 'externaluser'@'somehost';
```

When PROXY [696] is granted, it must be the only privilege named in the GRANT statement, the REQUIRE clause cannot be given, and the only permitted WITH option is WITH GRANT OPTION.

Proxying requires that the proxy user authenticate through a plugin that returns the name of the proxied user to the server when the proxy user connects, and that the proxy user have the PROXY privilege for the proxied user. For details and examples, see Section 6.3.7, “Proxy Users”.

MySQL stores proxy privileges in the `mysql.proxies_priv` table.

For the global, database, table, and routine levels, GRANT ALL assigns only the privileges that exist at the level you are granting. For example, GRANT ALL ON *db_name.** is a database-level statement, so

it does not grant any global-only privileges such as [FILE \[696\]](#). Granting [ALL \[695\]](#) does not assign the [PROXY \[696\]](#) privilege.

The *object_type* clause, if present, should be specified as [TABLE](#), [FUNCTION](#), or [PROCEDURE](#) when the following object is a table, a stored function, or a stored procedure.

The privileges for a database, table, column, or routine are formed additively as the logical [OR \[1095\]](#) of the privileges at each of the privilege levels. For example, if a user has a global [SELECT \[697\]](#) privilege, the privilege cannot be denied by an absence of the privilege at the database, table, or column level. Details of the privilege-checking procedure are presented in [Section 6.2.5, “Access Control, Stage 2: Request Verification”](#).

MySQL enables you to grant privileges on databases or tables that do not exist. For tables, the privileges to be granted must include the [CREATE \[695\]](#) privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for databases or tables that are to be created at a later time.



Important

MySQL does not automatically revoke any privileges when you drop a database or table. However, if you drop a routine, any routine-level privileges granted for that routine are revoked.

Account Names and Passwords

The *user* value indicates the MySQL account to which the [GRANT](#) statement applies. To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *user* value in the form *user_name@host_name*. If a *user_name* or *host_name* value is legal as an unquoted identifier, you need not quote it. However, quotation marks are necessary to specify a *user_name* string containing special characters (such as “-”), or a *host_name* string containing special characters or wildcard characters (such as “%”); for example, 'test-user'@'%.com'. Quote the user name and host name separately.

You can specify wildcards in the host name. For example, *user_name@'%.example.com'* applies to *user_name* for any host in the *example.com* domain, and *user_name@'192.168.1.%'* applies to *user_name* for any host in the *192.168.1* class C subnet.

The simple form *user_name* is a synonym for *user_name@'%'*.

MySQL does not support wildcards in user names. To refer to an anonymous user, specify an account with an empty user name with the [GRANT](#) statement:

```
GRANT ALL ON test.* TO ''@'localhost' ...
```

In this case, any user who connects from the local host with the correct password for the anonymous user will be permitted access, with the privileges associated with the anonymous-user account.

For additional information about user name and host name values in account names, see [Section 6.2.3, “Specifying Account Names”](#).

To specify quoted values, quote database, table, column, and routine names as identifiers. Quote user names and host names as identifiers or as strings. Quote passwords as strings. For string-quoting and identifier-quoting guidelines, see [Section 9.1.1, “String Literals”](#), and [Section 9.2, “Schema Object Names”](#).

The “_” and “%” wildcards are permitted when specifying database names in [GRANT](#) statements that grant privileges at the global or database levels. This means, for example, that if you want to use a “_” character

as part of a database name, you should specify it as “_” in the `GRANT` statement, to prevent the user from being able to access additional databases matching the wildcard pattern; for example, `GRANT ... ON `foo_bar`.* TO ...`



Warning

If you permit anonymous users to connect to the MySQL server, you should also grant privileges to all local users as `user_name@localhost`. Otherwise, the anonymous user account for `localhost` in the `mysql.user` table (created during MySQL installation) is used when named users try to log in to the MySQL server from the local machine. For details, see [Section 6.2.4, “Access Control, Stage 1: Connection Verification”](#).

To determine whether the preceding warning applies to you, execute the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

To avoid the problem just described, delete the local anonymous user account using this statement:

```
DROP USER ''@'localhost';
```

`GRANT` supports host names up to 60 characters long. Database, table, column, and routine names can be up to 64 characters. User names can be up to 16 characters.



Warning

The permissible length for user names cannot be changed by altering the `mysql.user` table. Attempting to do so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server. You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure described in [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

The user specification may indicate how the user should authenticate when connecting to the server, through inclusion of an `IDENTIFIED BY` or `IDENTIFIED WITH` clause. The syntax is the same as for the `CREATE USER` statement. See [Section 13.7.1.1, “CREATE USER Syntax”](#).

When the `IDENTIFIED BY` clause is present and you have global grant privileges, the password becomes the new password for the account, even if the account exists and already has a password. With no `IDENTIFIED BY` clause, the account password remains unchanged.

If the account named in a `GRANT` statement does not exist in the `mysql.user` table, `GRANT` creates it if the `NO_AUTO_CREATE_USER` [613] SQL mode is not enabled. *This is very insecure* unless you specify a nonempty password using `IDENTIFIED BY` or an authentication plugin using `IDENTIFIED WITH`.

If the account does not exist and `NO_AUTO_CREATE_USER` [613] is enabled, `GRANT` fails and does not create the account unless you specify a nonempty password with `IDENTIFIED BY` or specify an `IDENTIFIED WITH` clause to name an authentication plugin.



Important

`GRANT` may be recorded in server logs or in a history file such as `~/mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. See [Section 6.1.2, “Keeping Passwords Secure”](#).

Other Account Characteristics

The `WITH` clause is used for several purposes:

- To enable a user to grant privileges to other users
- To specify resource limits for a user
- To specify whether and how a user must use secure connections to the server

The `WITH GRANT OPTION` clause gives the user the ability to give to other users any privileges the user has at the specified privilege level. You should be careful to whom you give the `GRANT OPTION` [696] privilege because two users with different privileges may be able to combine privileges!

You cannot grant another user a privilege which you yourself do not have; the `GRANT OPTION` [696] privilege enables you to assign only those privileges which you yourself possess.

Be aware that when you grant a user the `GRANT OPTION` [696] privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the `INSERT` [696] privilege on a database. If you then grant the `SELECT` [697] privilege on the database and specify `WITH GRANT OPTION`, that user can give to other users not only the `SELECT` [697] privilege, but also `INSERT` [696]. If you then grant the `UPDATE` [697] privilege to the user on the database, the user can grant `INSERT` [696], `SELECT` [697], and `UPDATE` [697].

For a nonadministrative user, you should not grant the `ALTER` [695] privilege globally or for the `mysql` database. If you do that, the user can try to subvert the privilege system by renaming tables!

For additional information about security risks associated with particular privileges, see [Section 6.2.1, “Privileges Provided by MySQL”](#).

Several `WITH` clause options specify limits on use of server resources by an account:

- The `MAX_QUERIES_PER_HOUR count`, `MAX_UPDATES_PER_HOUR count`, and `MAX_CONNECTIONS_PER_HOUR count` limits restrict the number of queries, updates, and connections to the server permitted to this account during any given one-hour period. (Queries for which results are served from the query cache do not count against the `MAX_QUERIES_PER_HOUR` limit.) If `count` is 0 (the default), this means that there is no limitation for the account.
- The `MAX_USER_CONNECTIONS count` limit restricts the maximum number of simultaneous connections to the server by the account. A nonzero `count` specifies the limit for the account explicitly. If `count` is 0 (the default), the server determines the number of simultaneous connections for the account from the global value of the `max_user_connections` [525] system variable. If `max_user_connections` [525] is also zero, there is no limit for the account.

To specify resource limits for an existing user without affecting existing privileges, use `GRANT USAGE` at the global level (`ON *.*`) and name the limits to be changed. For example:

```
GRANT USAGE ON *.* TO ...
WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

Limits not specified retain their current values.

For more information on restricting access to server resources, see [Section 6.3.4, “Setting Account Resource Limits”](#).

MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the user name and password. To specify SSL-related options for a MySQL account, use the `REQUIRE` clause

of the `GRANT` statement. (For background information on the use of SSL with MySQL, see [Section 6.3.8, “Using SSL for Secure Connections”](#).)

There are a number of different possibilities for limiting connection types for a given account:

- `REQUIRE NONE` indicates that the account has no SSL or X509 requirements. This is the default if no SSL-related `REQUIRE` options are specified. Unencrypted connections are permitted if the user name and password are valid. However, encrypted connections can also be used, at the client's option, if the client has the proper certificate and key files. That is, the client need not specify any SSL command options, in which case the connection will be unencrypted. To use an encrypted connection, the client must specify either the `--ssl-ca` [750] option, or all three of the `--ssl-ca` [750], `--ssl-key` [751], and `--ssl-cert` [750] options.
- The `REQUIRE SSL` option tells the server to permit only SSL-encrypted connections for the account.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

To connect, the client must specify the `--ssl-ca` [750] option to authenticate the server certificate, and may additionally specify the `--ssl-key` [751] and `--ssl-cert` [750] options. If neither `--ssl-ca` [750] option nor `--ssl-capath` [750] option is specified, the client does not authenticate the server certificate.

- `REQUIRE X509` means that the client must have a valid certificate but that the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

To connect, the client must specify the `--ssl-ca` [750], `--ssl-key` [751], and `--ssl-cert` [750] options. This is also true for `ISSUER` and `SUBJECT` because those `REQUIRE` options imply `X509`.

- `REQUIRE ISSUER 'issuer'` places the restriction on connection attempts that the client must present a valid X509 certificate issued by CA `'issuer'`. If the client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
  O=MySQL Finland AB/CN=Tonu Samuel/emailAddress=tonu@example.com';
```

The `'issuer'` value should be entered as a single string.



Note

If MySQL is linked against a version of OpenSSL older than 0.9.6h, use `Email` rather than `emailAddress` in the `'issuer'` value.

- `REQUIRE SUBJECT 'subject'` places the restriction on connection attempts that the client must present a valid X509 certificate containing the subject `subject`. If the client presents a certificate that is valid but has a different subject, the server rejects the connection.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
  O=MySQL demo client certificate/
  CN=Tonu Samuel/emailAddress=tonu@example.com';
```

The `'subject'` value should be entered as a single string.



Note

Regarding `emailAddress`, see the note in the description of [REQUIRE ISSUER](#).

- [REQUIRE CIPHER](#) `'cipher'` is needed to ensure that ciphers and key lengths of sufficient strength are used. SSL itself can be weak if old algorithms using short encryption keys are used. Using this option, you can ask that a specific cipher method is used for a connection.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The [SUBJECT](#), [ISSUER](#), and [CIPHER](#) options can be combined in the [REQUIRE](#) clause like this:

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
    O=MySQL demo client certificate/
    CN=Tonu Samuel/emailAddress=tonu@example.com'
  AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
    O=MySQL Finland AB/CN=Tonu Samuel/emailAddress=tonu@example.com'
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The order of the options does not matter, but no option can be specified twice. The [AND](#) keyword is optional between [REQUIRE](#) options.

If you are using table, column, or routine privileges for even one user, the server examines table, column, and routine privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

MySQL and Standard SQL Versions of [GRANT](#)

The biggest differences between the MySQL and standard SQL versions of [GRANT](#) are:

- MySQL associates privileges with the combination of a host name and user name and not with only a user name.
- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.
- MySQL does not support the standard SQL [UNDER](#) privilege.
- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges the user has been granted are revoked. This is also true in MySQL if you use [DROP USER](#). See [Section 13.7.1.2, “DROP USER Syntax”](#).
- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped only with explicit [DROP USER](#) or [REVOKE](#) statements or by manipulating the MySQL grant tables directly.
- In MySQL, it is possible to have the [INSERT \[696\]](#) privilege for only some of the columns in a table. In this case, you can still execute [INSERT](#) statements on the table, provided that you insert values only for those columns for which you have the [INSERT \[696\]](#) privilege. The omitted columns are set to their implicit default values if strict SQL mode is not enabled. In strict mode, the statement is rejected if any of the omitted columns have no default value. (Standard SQL requires you to have the [INSERT \[696\]](#)

privilege on all columns.) [Section 5.1.7, “Server SQL Modes”](#), discusses strict mode. [Section 11.5, “Data Type Default Values”](#), discusses implicit default values.

13.7.1.4. RENAME USER Syntax

```
RENAME USER old_user TO new_user
[, old_user TO new_user] ...
```

The `RENAME USER` statement renames existing MySQL accounts. To use it, you must have the global `CREATE USER` [696] privilege or the `UPDATE` [697] privilege for the `mysql` database. An error occurs if any old account does not exist or any new account exists. Each account name uses the format described in [Section 6.2.3, “Specifying Account Names”](#). For example:

```
RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.

`RENAME USER` causes the privileges held by the old user to be those held by the new user. However, `RENAME USER` does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the `DEFINER` attribute names the old user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see [Section 19.6, “Access Control for Stored Programs and Views”](#).)

The privilege changes take effect as indicated in [Section 6.2.6, “When Privilege Changes Take Effect”](#).

13.7.1.5. REVOKE Syntax

```
REVOKE
  priv_type [(column_list)]
  [, priv_type [(column_list))] ...
ON [object_type] priv_level
FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
FROM user [, user] ...

REVOKE PROXY ON user
FROM user [, user] ...
```

The `REVOKE` statement enables system administrators to revoke privileges from MySQL accounts. Each account name uses the format described in [Section 6.2.3, “Specifying Account Names”](#). For example:

```
REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.

For details on the levels at which privileges exist, the permissible `priv_type` and `priv_level` values, and the syntax for specifying users and passwords, see [Section 13.7.1.3, “GRANT Syntax”](#)

To use the first `REVOKE` syntax, you must have the `GRANT OPTION` [696] privilege, and you must have the privileges that you are revoking.

To revoke all privileges, use the second syntax, which drops all global, database, table, column, and routine privileges for the named user or users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this `REVOKE` syntax, you must have the global `CREATE USER` [696] privilege or the `UPDATE` [697] privilege for the `mysql` database.

`REVOKE` removes privileges, but does not drop `mysql.user` table entries. To remove a user account entirely, use `DROP USER` (see [Section 13.7.1.2](#), “`DROP USER Syntax`”) or `DELETE`.

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` [518] system variable is set to a nonzero value, `REVOKE` cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (`GRANT` will not create such rows when `lower_case_table_names` [518] is set, but such rows might have been created prior to setting the variable.)

When successfully executed from the `mysql` program, `REVOKE` responds with `Query OK, 0 rows affected`. To determine what privileges result from the operation, use `SHOW GRANTS`. See [Section 13.7.5.22](#), “`SHOW GRANTS Syntax`”.

13.7.1.6. `SET PASSWORD Syntax`

```
SET PASSWORD [FOR user] =
{
  PASSWORD('cleartext password')
  | OLD_PASSWORD('cleartext password')
  | 'encrypted password'
}
```

The `SET PASSWORD` statement assigns a password to an existing MySQL user account. When the `read_only` [544] system variable is enabled, the `SUPER` [697] privilege is required to use `SET PASSWORD`, in addition to whatever other privileges might be required.

If the password is specified using the `PASSWORD()` [1188] or `OLD_PASSWORD()` [1187] function, the cleartext (unencrypted) password should be given as the argument to the function, which hashes the password and returns the encrypted password string. If the password is specified without using either function, it should be the already encrypted password value as a literal string. In all cases, the encrypted password string must be in the format required by the authentication method used for the account.

With no `FOR user` clause, this statement sets the password for the current user. (To see which account the server authenticated you as, invoke the `CURRENT_USER()` [1192] function.) Any client who successfully connects to the server using a nonanonymous account can change the password for that account.

With a `FOR user` clause, this statement sets the password for the named user. You must have the `UPDATE` privilege for the `mysql` database to do this. The user account name uses the format described in [Section 6.2.3](#), “`Specifying Account Names`”. The `user` value should be given as `'user_name'@'host_name'`, where `'user_name'` and `'host_name'` are exactly as listed in the `User` and `Host` columns of the `mysql.user` table row. (If you specify only a user name, a host name of `'%'` is used.) For example, to set the password for an account with `User` and `Host` column values of `'bob'` and `'%.example.org'`, write the statement like this:

```
SET PASSWORD FOR 'bob'@'%.example.org' = PASSWORD('cleartext password');
```

That is equivalent to the following statements:

```
UPDATE mysql.user SET Password=PASSWORD('cleartext password')
  WHERE User='bob' AND Host='%.example.org';
FLUSH PRIVILEGES;
```

Another way to set the password is to use `GRANT`:

```
GRANT USAGE ON *.* TO 'bob'@'%.example.org' IDENTIFIED BY 'cleartext password';
```

The `old_passwords` [534] system variable value determines the hashing method used by `PASSWORD()` [1188]. If you specify the password using that function and `SET PASSWORD` rejects the

password as not being in the correct format, it may be necessary to set `old_passwords` [534] to change the hashing method. For descriptions of the permitted values, see [Section 5.1.4, “Server System Variables”](#).

For more information about setting passwords, see [Section 6.3.5, “Assigning Account Passwords”](#)



Important

`SET PASSWORD` may be recorded in server logs or in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. See [Section 6.1.2, “Keeping Passwords Secure”](#).



Caution

If you are connecting to a MySQL 4.1 or later server using a pre-4.1 client program, do not change your password without first reading [Section 6.1.2.4, “Password Hashing in MySQL”](#). The default password hashing format changed in MySQL 4.1, and if you change your password, it might be stored using a hashing format that pre-4.1 clients cannot generate, thus preventing you from connecting to the server afterward.

If you are using MySQL Replication, be aware that, currently, a password used by a replication slave as part of a `CHANGE MASTER TO` statement is effectively limited to 32 characters in length; the password can be longer, but any excess characters are truncated. This is not due to any limit imposed by the MySQL Server generally, but rather is an issue specific to MySQL Replication. (For more information, see Bug #43439.)

13.7.2. Table Maintenance Statements

13.7.2.1. ANALYZE TABLE Syntax

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
      tbl_name [, tbl_name] ...
```

`ANALYZE TABLE` analyzes and stores the key distribution for a table. During the analysis, the table is locked with a read lock for `InnoDB` and `MyISAM`. This statement works with `InnoDB`, `NDB`, and `MyISAM` tables. For `MyISAM` tables, this statement is equivalent to using `myisamchk --analyze`.

For more information on how the analysis works within `InnoDB`, see [Section 14.3.15, “Limits on InnoDB Tables”](#).

MySQL uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant. In addition, key distributions can be used when deciding which indexes to use for a specific table within a query.

This statement requires `SELECT` [697] and `INSERT` [696] privileges for the table.

`ANALYZE TABLE` is supported for partitioned tables, and you can use `ALTER TABLE ... ANALYZE PARTITION` to analyze one or more partitions; for more information, see [Section 13.1.7, “ALTER TABLE Syntax”](#), and [Section 18.3.3, “Maintenance of Partitions”](#).

`ANALYZE TABLE` returns a result set with the following columns.

Column	Value
Table	The table name

Column	Value
Op	Always <i>analyze</i>
Msg_type	<i>status, error, info, note, or warning</i>
Msg_text	An informational message

You can check the stored key distribution with the `SHOW INDEX` statement. See [Section 13.7.5.23, “SHOW INDEX Syntax”](#).

If the table has not changed since the last `ANALYZE TABLE` statement, the table is not analyzed again.

By default, the server writes `ANALYZE TABLE` statements to the binary log so that they replicate to replication slaves. To suppress logging, use the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

13.7.2.2. CHECK TABLE Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...
option = {FOR UPGRADE | QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

`CHECK TABLE` checks a table or tables for errors. `CHECK TABLE` works for InnoDB, MyISAM, ARCHIVE, and CSV tables. For MyISAM tables, the key statistics are updated as well.

To check a table, you must have some privilege for it.

`CHECK TABLE` can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

`CHECK TABLE` is supported for partitioned tables, and you can use `ALTER TABLE ... CHECK PARTITION` to check one or more partitions; for more information, see [Section 13.1.7, “ALTER TABLE Syntax”](#), and [Section 18.3.3, “Maintenance of Partitions”](#).

Output

`CHECK TABLE` returns a result set with the following columns.

Column	Value
Table	The table name
Op	Always <i>check</i>
Msg_type	<i>status, error, info, note, or warning</i>
Msg_text	An informational message

Note that the statement might produce many rows of information for each checked table. The last row has a `Msg_type` value of *status* and the `Msg_text` normally should be *OK*. If you don't get *OK*, or *Table is already up to date* you should normally run a repair of the table. See [Section 7.6, “MyISAM Table Maintenance and Crash Recovery”](#). *Table is already up to date* means that the storage engine for the table indicated that there was no need to check the table.

Checking Version Compatibility

The `FOR UPGRADE` option checks whether the named tables are compatible with the current version of MySQL. With `FOR UPGRADE`, the server checks each table to determine whether there have been any incompatible changes in any of the table's data types or indexes since the table was created. If not, the check succeeds. Otherwise, if there is a possible incompatibility, the server runs a full check on the table

(which might take some time). If the full check succeeds, the server marks the table's `.frm` file with the current MySQL version number. Marking the `.frm` file ensures that further checks for the table with the same version of the server will be fast.

Incompatibilities might occur because the storage format for a data type has changed or because its sort order has changed. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.

Currently, `FOR UPGRADE` discovers these incompatibilities:

- The indexing order for end-space in `TEXT` columns for `InnoDB` and `MyISAM` tables changed between MySQL 4.1 and 5.0.
- The storage method of the new `DECIMAL` data type changed between MySQL 5.0.3 and 5.0.5.
- If your table was created by a different version of the MySQL server than the one you are currently running, `FOR UPGRADE` indicates that the table has an `.frm` file with an incompatible version. In this case, the result set returned by `CHECK TABLE` contains a line with a `Msg_type` value of `error` and a `Msg_text` value of `Table upgrade required. Please do "REPAIR TABLE `tbl_name`" to fix it!`
- Changes are sometimes made to character sets or collations that require table indexes to be rebuilt. For details about these changes and when `FOR UPGRADE` detects them, see [Section 2.12.3, "Checking Whether Tables or Indexes Must Be Rebuilt"](#).

Checking Data Consistency

The other check options that can be given are shown in the following table. These options are passed to the storage engine, which may use them or not.

Type	Meaning
<code>QUICK</code>	Do not scan the rows to check for incorrect links. Applies to <code>InnoDB</code> and <code>MyISAM</code> tables and views.
<code>FAST</code>	Check only tables that have not been closed properly. Applies only to <code>MyISAM</code> tables and views; ignored for <code>InnoDB</code> .
<code>CHANGED</code>	Check only tables that have been changed since the last check or that have not been closed properly. Applies only to <code>MyISAM</code> tables and views; ignored for <code>InnoDB</code> .
<code>MEDIUM</code>	Scan rows to verify that deleted links are valid. This also calculates a key checksum for the rows and verifies this with a calculated checksum for the keys. Applies only to <code>MyISAM</code> tables and views; ignored for <code>InnoDB</code> .
<code>EXTENDED</code>	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time. Applies only to <code>MyISAM</code> tables and views; ignored for <code>InnoDB</code> .

If none of the options `QUICK`, `MEDIUM`, or `EXTENDED` are specified, the default check type for dynamic-format `MyISAM` tables is `MEDIUM`. This has the same result as running `myisamchk --medium-check tbl_name` on the table. The default check type also is `MEDIUM` for static-format `MyISAM` tables, unless `CHANGED` or `FAST` is specified. In that case, the default is `QUICK`. The row scan is skipped for `CHANGED` and `FAST` because the rows are very seldom corrupted.

You can combine check options, as in the following example that does a quick check on the table to determine whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

**Note**

In some cases, `CHECK TABLE` changes the table. This happens if the table is marked as “corrupted” or “not closed properly” but `CHECK TABLE` does not find any problems in the table. In this case, `CHECK TABLE` marks the table as okay.

If a table is corrupted, it is most likely that the problem is in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

If you just want to check a table that you assume is okay, you should use no check options or the `QUICK` option. The latter should be used when you are in a hurry and can take the very small risk that `QUICK` does not find an error in the data file. (In most cases, under normal usage, MySQL should find any error in the data file. If this happens, the table is marked as “corrupted” and cannot be used until it is repaired.)

`FAST` and `CHANGED` are mostly intended to be used from a script (for example, to be executed from `cron`) if you want to check tables from time to time. In most cases, `FAST` is to be preferred over `CHANGED`. (The only case when it is not preferred is when you suspect that you have found a bug in the `MyISAM` code.)

`EXTENDED` is to be used only after you have run a normal check but still get strange errors from a table when MySQL tries to update a row or find a row by key. This is very unlikely if a normal check has succeeded.

Use of `CHECK TABLE ... EXTENDED` might influence the execution plan generated by the query optimizer.

Some problems reported by `CHECK TABLE` cannot be corrected automatically:

- Found row where the `auto_increment` column has the value 0.

This means that you have a row in the table where the `AUTO_INCREMENT` index column contains the value 0. (It is possible to create a row where the `AUTO_INCREMENT` column is 0 by explicitly setting the column to 0 with an `UPDATE` statement.)

This is not an error in itself, but could cause trouble if you decide to dump the table and restore it or do an `ALTER TABLE` on the table. In this case, the `AUTO_INCREMENT` column changes value according to the rules of `AUTO_INCREMENT` columns, which could cause problems such as a duplicate-key error.

To get rid of the warning, simply execute an `UPDATE` statement to set the column to some value other than 0.

InnoDB Tables

The following notes apply to `InnoDB` tables:

- If `CHECK TABLE` finds a problem for an `InnoDB` table, the server shuts down to prevent error propagation. Details of the error will be written to the error log.
- When an `InnoDB` table is stored in its own `.ibd` file in `file-per-table` mode, the first 3 pages of the `.ibd` contain header information rather than table or index data. The `CHECK TABLE` statement does not detect inconsistencies that only affect the header data. To verify the entire contents of an `InnoDB .ibd` file, use the `innochecksum` command.

13.7.2.3. CHECKSUM TABLE Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

`CHECKSUM TABLE` reports a table checksum. This statement requires the `SELECT [697]` privilege for the table.

With `QUICK`, the live table checksum is reported if it is available, or `NULL` otherwise. This is very fast. A live checksum is enabled by specifying the `CHECKSUM=1` table option when you create the table; currently, this is supported only for `MyISAM` tables. See [Section 13.1.17, “CREATE TABLE Syntax”](#).

With `EXTENDED`, the entire table is read row by row and the checksum is calculated. This can be very slow for large tables.

If neither `QUICK` nor `EXTENDED` is specified, MySQL returns a live checksum if the table storage engine supports it and scans the table otherwise.

For a nonexistent table, `CHECKSUM TABLE` returns `NULL` and generates a warning.

In MySQL 5.5, `CHECKSUM TABLE` returns 0 for partitioned tables unless you include the `EXTENDED` option. This issue is resolved in MySQL 5.6. (Bug #11933226, Bug #60681)

The checksum value depends on the table row format. If the row format changes, the checksum also changes. For example, the storage format for `VARCHAR` changed between MySQL 4.1 and 5.0, so if a 4.1 table is upgraded to MySQL 5.0, the checksum value may change.



Important

If the checksums for two tables are different, then it is almost certain that the tables are different in some way. However, because the hashing function used by `CHECKSUM TABLE` is not guaranteed to be collision-free, there is a slight chance that two tables which are not identical can produce the same checksum.

13.7.2.4. OPTIMIZE TABLE Syntax

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
tbl_name [, tbl_name] ...
```

Reorganizes the physical storage of table data and associated index data, to reduce storage space and improve I/O efficiency when accessing the table. The exact changes made to each table depend on the [storage engine](#) used by that table.

Use `OPTIMIZE TABLE` in these cases, depending on the type of table:

- After doing substantial insert, update, or delete operations on an `InnoDB` table that has its own `.ibd` file because it was created with the `innodb_file_per_table` [1555] option enabled. The table and indexes are reorganized, and disk space can be reclaimed for use by the operating system.
- After deleting a large part of a `MyISAM` or `ARCHIVE` table, or making many changes to a `MyISAM` or `ARCHIVE` table with variable-length rows (tables that have `VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT` columns). Deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions. You can use `OPTIMIZE TABLE` to reclaim the unused space and to defragment the data file. After extensive changes to a table, this statement may also improve performance of statements that use the table, sometimes significantly.

This statement requires `SELECT` [697] and `INSERT` [696] privileges for the table.

`OPTIMIZE TABLE` is also supported for partitioned tables. For information about using this statement with partitioned tables and table partitions, see [Section 18.3.3, “Maintenance of Partitions”](#).

`OPTIMIZE TABLE` works for `InnoDB`, `MyISAM`, and `ARCHIVE` tables.

`OPTIMIZE TABLE` is also supported for dynamic columns of in-memory `NDBCLUSTER` tables. It does not work for Disk Data tables. The performance of `OPTIMIZE` on Cluster tables can be tuned by adjusting the

value of the `ndb_optimization_delay` system variable, which controls the number of milliseconds to wait between processing batches of rows by `OPTIMIZE TABLE`. See [Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”](#), for more information.

`OPTIMIZE TABLE` is not supported for tables using other storage engines except as noted previously.

For MySQL Cluster tables, `OPTIMIZE TABLE` can be interrupted by (for example) killing the SQL thread performing the `OPTIMIZE` operation.

InnoDB Details

For InnoDB tables, `OPTIMIZE TABLE` is mapped to `ALTER TABLE`, which rebuilds the table to update index statistics and free unused space in the clustered index. This is displayed in the output of `OPTIMIZE TABLE` when you run it on an InnoDB table, as shown here:

```
mysql> OPTIMIZE TABLE foo;
+-----+-----+-----+-----+
| Table      | Op       | Msg_type  | Msg_text                                     |
+-----+-----+-----+-----+
| test.foo   | optimize | note      | Table does not support optimize, doing recreate + analyze instead |
| test.foo   | optimize | status    | OK                                          |
+-----+-----+-----+-----+
```

InnoDB stores data using a page-allocation method and does not suffer from fragmentation in the same way that legacy storage engines (such as `MyISAM`) will. When considering whether or not to run `optimize`, consider the workload of transactions that your server will process:

- Some level of fragmentation is expected. InnoDB only fills [pages](#) 93% full, to leave room for updates without having to split pages.
- Delete operations might leave gaps that leave pages less filled than desired, which could make it worthwhile to optimize the table.
- Updates to rows usually rewrite the data within the same page, depending on the data type and row format, when sufficient space is available. See [Section 14.4.3.4, “Compressing BLOB, VARCHAR and TEXT Columns”](#) and [Section 14.4.5.1, “Overview of InnoDB Row Storage”](#).
- High-concurrency workloads might leave gaps in indexes over time, as InnoDB retains multiple versions of the same data due through its [MVCC](#) mechanism. See [Section 14.3.10, “InnoDB Multi-Versioning”](#).

MyISAM Details

For `MyISAM` tables, `OPTIMIZE TABLE` works as follows:

1. If the table has deleted or split rows, repair the table.
2. If the index pages are not sorted, sort them.
3. If the table's statistics are not up to date (and the repair could not be accomplished by sorting the index), update them.

You can make `OPTIMIZE TABLE` work on other storage engines by starting `mysqld --skip-new` option. In this case, `OPTIMIZE TABLE` is just mapped to `ALTER TABLE`.

Result Set

`OPTIMIZE TABLE` returns a result set with the following columns.

Column	Value
Table	The table name
Op	Always <code>optimize</code>
Msg_type	<code>status</code> , <code>error</code> , <code>info</code> , <code>note</code> , or <code>warning</code>
Msg_text	An informational message

Note that MySQL [locks the table](#) during the time `OPTIMIZE TABLE` is running.

By default, the server writes `OPTIMIZE TABLE` statements to the binary log so that they replicate to replication slaves. To suppress logging, use the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

`OPTIMIZE TABLE` does not sort R-tree indexes, such as spatial indexes on `POINT` columns. (Bug #23578)

As of MySQL 5.5.6, `OPTIMIZE TABLE` catches and throws any errors that occur while copying table statistics from the old file to the newly created file. For example, if the user ID of the owner of the `.frm`, `.MYD`, or `.MYI` file is different from the user ID of the `mysqld` process, `OPTIMIZE TABLE` generates a "cannot change ownership of the file" error unless `mysqld` is started by the `root` user.

13.7.2.5. REPAIR TABLE Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] TABLE
      tbl_name [, tbl_name] ...
      [QUICK] [EXTENDED] [USE_FRM]
```

`REPAIR TABLE` repairs a possibly corrupted table. By default, it has the same effect as `myisamchk --recover tbl_name`. `REPAIR TABLE` works for `MyISAM`, `ARCHIVE`, and `CSV` tables. See [Section 14.5, "The MyISAM Storage Engine"](#), and [Section 14.8, "The ARCHIVE Storage Engine"](#), and [Section 14.7, "The CSV Storage Engine"](#)

This statement requires [SELECT \[697\]](#) and [INSERT \[696\]](#) privileges for the table.

`REPAIR TABLE` is supported for partitioned tables. However, the `USE_FRM` option cannot be used with this statement on a partitioned table.

You can use `ALTER TABLE ... REPAIR PARTITION` to repair one or more partitions; for more information, see [Section 13.1.7, "ALTER TABLE Syntax"](#), and [Section 18.3.3, "Maintenance of Partitions"](#).

Normally, you should never have to run `REPAIR TABLE`. However, if disaster strikes, this statement is very likely to get back all your data from a `MyISAM` table. If your tables become corrupted often, you should try to find the reason for it, to eliminate the need to use `REPAIR TABLE`. See [Section C.5.4.2, "What to Do If MySQL Keeps Crashing"](#), and [Section 14.5.4, "MyISAM Table Problems"](#).



Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors. See [Chapter 7, Backup and Recovery](#).



Warning

If the server crashes during a `REPAIR TABLE` operation, it is essential after restarting it that you immediately execute another `REPAIR TABLE` statement for the table before performing any other operations on it. In the worst case, you might

have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely but possible scenario that underscores the value of making a backup first.

`REPAIR TABLE` returns a result set with the following columns.

Column	Value
Table	The table name
Op	Always <code>repair</code>
Msg_type	<code>status</code> , <code>error</code> , <code>info</code> , <code>note</code> , or <code>warning</code>
Msg_text	An informational message

The `REPAIR TABLE` statement might produce many rows of information for each repaired table. The last row has a `Msg_type` value of `status` and `Msg_text` normally should be `OK`. If you do not get `OK` for a `MyISAM` table, you should try repairing it with `myisamchk --safe-recover`. (`REPAIR TABLE` does not implement all the options of `myisamchk`.) With `myisamchk --safe-recover`, you can also use options that `REPAIR TABLE` does not support, such as `--max-record-length` [351].

If you use the `QUICK` option, `REPAIR TABLE` tries to repair only the index file, and not the data file. This type of repair is like that done by `myisamchk --recover --quick`.

If you use the `EXTENDED` option, MySQL creates the index row by row instead of creating one index at a time with sorting. This type of repair is like that done by `myisamchk --safe-recover`.

The `USE_FRM` option is available for use if the `.MYI` index file is missing or if its header is corrupted. This option tells MySQL not to trust the information in the `.MYI` file header and to re-create it using information from the `.frm` file. This kind of repair cannot be done with `myisamchk`.



Note

Use the `USE_FRM` option *only* if you cannot use regular `REPAIR` modes! Telling the server to ignore the `.MYI` file makes important table metadata stored in the `.MYI` unavailable to the repair process, which can have deleterious consequences:

- The current `AUTO_INCREMENT` value is lost.
- The link to deleted records in the table is lost, which means that free space for deleted records will remain unoccupied thereafter.
- The `.MYI` header indicates whether the table is compressed. If the server ignores this information, it cannot tell that a table is compressed and repair can cause change or loss of table contents. This means that `USE_FRM` should not be used with compressed tables. That should not be necessary, anyway: Compressed tables are read only, so they should not become corrupt.



Caution

If you use `USE_FRM` for a table that was created by a different version of the MySQL server than the one you are currently running, `REPAIR TABLE` will not attempt to repair the table. In this case, the result set returned by `REPAIR TABLE` contains a line with a `Msg_type` value of `error` and a `Msg_text` value of `Failed repairing incompatible .FRM file`.

If `USE_FRM` is *not* used, `REPAIR TABLE` checks the table to see whether an upgrade is required. If so, it performs the upgrade, following the same rules as `CHECK TABLE ... FOR UPGRADE`. See

Section 13.7.2.2, “[CHECK TABLE Syntax](#)”, for more information. `REPAIR TABLE` without `USE_FRM` upgrades the `.frm` file to the current version.

By default, the server writes `REPAIR TABLE` statements to the binary log so that they replicate to replication slaves. To suppress logging, use the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.



Important

In the event that a table on the master becomes corrupted and you run `REPAIR TABLE` on it, any resulting changes to the original table are *not* propagated to slaves.

You may be able to increase `REPAIR TABLE` performance by setting certain system variables. See Section 8.6.3, “[Speed of REPAIR TABLE Statements](#)”.

As of MySQL 5.5.6, `REPAIR TABLE` catches and throws any errors that occur while copying table statistics from the old corrupted file to the newly created file. For example, if the user ID of the owner of the `.frm`, `.MYD`, or `.MYI` file is different from the user ID of the `mysqld` process, `REPAIR TABLE` generates a “cannot change ownership of the file” error unless `mysqld` is started by the `root` user.

13.7.3. Plugin and User-Defined Function Statements

13.7.3.1. CREATE FUNCTION Syntax for User-Defined Functions

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL|DECIMAL}
SONAME shared_library_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` [1123] or `CONCAT()` [1102].

function_name is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value. `DECIMAL` is a legal value after `RETURNS`, but currently `DECIMAL` functions return string values and should be written like `STRING` functions.

shared_library_name is the basename of the shared object file that contains the code that implements the function. The file must be located in the plugin directory. This directory is given by the value of the `plugin_dir` [537] system variable. For more information, see Section 23.3.2.5, “[Compiling and Installing User-Defined Functions](#)”.

To create a function, you must have the `INSERT` [696] privilege for the `mysql` database. This is necessary because `CREATE FUNCTION` adds a row to the `mysql.func` system table that records the function's name, type, and shared library name. If you do not have this table, you should run the `mysql_upgrade` command to create it. See Section 4.4.7, “[mysql_upgrade — Check and Upgrade MySQL Tables](#)”.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` [461] option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see Section 23.3.2, “[Adding a New User-Defined Function](#)”. For the UDF mechanism to work, functions must be written in C or C++ (or another language that can use C calling conventions), your operating system must support dynamic loading and you must have compiled `mysqld` dynamically (not statically).

An `AGGREGATE` function works exactly like a native MySQL aggregate (summary) function such as `SUM` or `COUNT()` [1204]. For `AGGREGATE` to work, your `mysql.func` table must contain a `type` column. If your

`mysql.func` table does not have this column, you should run the `mysql_upgrade` program to create it (see [Section 4.4.7](#), “`mysql_upgrade` — Check and Upgrade MySQL Tables”).



Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may crash.

13.7.3.2. `DROP FUNCTION` Syntax

```
DROP FUNCTION function_name
```

This statement drops the user-defined function (UDF) named *function_name*.

To drop a function, you must have the `DELETE` [696] privilege for the `mysql` database. This is because `DROP FUNCTION` removes a row from the `mysql.func` system table that records the function's name, type, and shared library name.



Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may crash.

`DROP FUNCTION` is also used to drop stored functions (see [Section 13.1.26](#), “`DROP PROCEDURE and DROP FUNCTION Syntax`”).

13.7.3.3. `INSTALL PLUGIN` Syntax

```
INSTALL PLUGIN plugin_name SONAME 'shared_library_name'
```

This statement installs a server plugin. It requires the `INSERT` privilege for the `mysql.plugin` table.

plugin_name is the name of the plugin as defined in the plugin descriptor structure contained in the library file (see [Section 23.2.4.2](#), “`Plugin Data Structures`”). Plugin names are not case sensitive. For maximal compatibility, plugin names should be limited to ASCII letters, digits, and underscore because they are used in C source files, shell command lines, M4 and Bourne shell scripts, and SQL environments.

shared_library_name is the name of the shared library that contains the plugin code. The name includes the file name extension (for example, `libmyplugin.so`, `libmyplugin.dll`, or `libmyplugin.dylib`).

The shared library must be located in the plugin directory (the directory named by the `plugin_dir` [537] system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, `plugin_dir` [537] is the `plugin` directory under the directory named by the `pkglibdir` configuration variable, but it can be changed by setting the value of `plugin_dir` [537] at server startup. For example, set its value in a `my.cnf` file:

```
[mysqld]
plugin_dir=/path/to/plugin/directory
```

If the value of `plugin_dir` [537] is a relative path name, it is taken to be relative to the MySQL base directory (the value of the `basedir` [485] system variable).

`INSTALL PLUGIN` loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used. When the server shuts down, it executes the deinitialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

`INSTALL PLUGIN` also registers the plugin by adding a line that indicates the plugin name and library file name to the `mysql.plugin` table. At server startup, the server loads and initializes any plugin that is listed in the `mysql.plugin` table. This means that a plugin is installed with `INSTALL PLUGIN` only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the `--skip-grant-tables` [461] option.

A plugin library can contain multiple plugins. For each of them to be installed, use a separate `INSTALL PLUGIN` statement. Each statement names a different plugin, but all of them specify the same library name.

`INSTALL PLUGIN` causes the server to read option (`my.cnf`) files just as during server startup. This enables the plugin to pick up any relevant options from those files. It is possible to add plugin options to an option file even before loading a plugin (if the `loose` prefix is used). It is also possible to uninstall a plugin, edit `my.cnf`, and install the plugin again. Restarting the plugin this way enables it to the new option values without a server restart.

For options that control individual plugin loading at server startup, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#). If you need to load plugins for a single server startup when the `--skip-grant-tables` [461] option is given (which tells the server not to read system tables), use the `--plugin-load` [457] option. See [Section 5.1.3, “Server Command Options”](#).

To remove a plugin, use the `UNINSTALL PLUGIN` statement.

For additional information about plugin loading, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

To see what plugins are installed, use the `SHOW PLUGINS` statement or query the `INFORMATION_SCHEMA.PLUGINS` table.

If you recompile a plugin library and need to reinstall it, you can use either of the following methods:

- Use `UNINSTALL PLUGIN` to uninstall all plugins in the library, install the new plugin library file in the plugin directory, and then use `INSTALL PLUGIN` to install all plugins in the library. This procedure has the advantage that it can be used without stopping the server. However, if the plugin library contains many plugins, you must issue many `INSTALL PLUGIN` and `UNINSTALL PLUGIN` statements.
- Stop the server, install the new plugin library file in the plugin directory, and restart the server.

13.7.3.4. `UNINSTALL PLUGIN` Syntax

```
UNINSTALL PLUGIN plugin_name
```

This statement removes an installed server plugin. It requires the `DELETE` [696] privilege for the `mysql.plugin` table.

plugin_name must be the name of some plugin that is listed in the `mysql.plugin` table. The server executes the plugin's deinitialization function and removes the row for the plugin from the `mysql.plugin` table, so that subsequent server restarts will not load and initialize the plugin. `UNINSTALL PLUGIN` does not remove the plugin's shared library file.

You cannot uninstall a plugin if any table that uses it is open.

Plugin removal has implications for the use of associated tables. For example, if a full-text parser plugin is associated with a `FULLTEXT` index on the table, uninstalling the plugin makes the table unusable. Any

attempt to access the table results in an error. The table cannot even be opened, so you cannot drop an index for which the plugin is used. This means that uninstalling a plugin is something to do with care unless you do not care about the table contents. If you are uninstalling a plugin with no intention of reinstalling it later and you care about the table contents, you should dump the table with `mysqldump` and remove the `WITH PARSER` clause from the dumped `CREATE TABLE` statement so that you can reload the table later. If you do not care about the table, `DROP TABLE` can be used even if any plugins associated with the table are missing.

For additional information about plugin loading, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

13.7.4. SET Syntax

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:
    user_var_name = expr
  | [GLOBAL | SESSION] system_var_name = expr
  | [@@global. | @@session. | @@]system_var_name = expr
```

The `SET` statement assigns values to different types of variables that affect the operation of the server or your client. Older versions of MySQL employed `SET OPTION`, but this syntax is deprecated in favor of `SET` without `OPTION`.

This section describes use of `SET` for assigning values to variables. The `SET` statement can be used to assign values to these types of variables:

- System variables. See [Section 5.1.4, “Server System Variables”](#). System variables also can be set at server startup, as described in [Section 5.1.5, “Using System Variables”](#).

User-defined variables. See [Section 9.4, “User-Defined Variables”](#).

- Stored procedure and function parameters, and stored program local variables. See [Section 13.6.4, “Variables in Stored Programs”](#).

Some variants of `SET` syntax are used in other contexts:

- `SET CHARACTER SET` and `SET NAMES` assign values to character set and collation variables associated with the connection to the server. `SET ONE_SHOT` is used for replication. These variants are described later in this section.
- `SET PASSWORD` assigns account passwords. See [Section 13.7.1.6, “SET PASSWORD Syntax”](#).
- `SET TRANSACTION ISOLATION LEVEL` sets the isolation level for transaction processing. See [Section 13.3.6, “SET TRANSACTION Syntax”](#).

The following discussion shows the different `SET` syntaxes that you can use to set variables. The examples use the `= [1096]` assignment operator, but you can also use the `:= [1095]` assignment operator for this purpose.

A user variable is written as `@var_name` and can be set as follows:

```
SET @var_name = expr;
```

Many system variables are dynamic and can be changed while the server runs by using the `SET` statement. For a list, see [Section 5.1.5.2, “Dynamic System Variables”](#). To change a system variable with `SET`, refer to it as `var_name`, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by `GLOBAL` or `@@global..` The `SUPER [697]` privilege is required to set global variables.

- To indicate explicitly that a variable is a session variable, precede its name by `SESSION`, `@@session.`, or `@@`. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- `LOCAL` and `@@local.` are synonyms for `SESSION` and `@@session..`
- If no modifier is present, `SET` changes the session variable.

A `SET` statement can contain multiple variable assignments, separated by commas. For example, the statement can assign values to a user-defined variable and a system variable. If you set several system variables, the most recent `GLOBAL` or `SESSION` modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

The `@@var_name` syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the `SET GLOBAL` statement).

To prevent incorrect usage, MySQL produces an error if you use `SET GLOBAL` with a variable that can only be used with `SET SESSION` or if you do not specify `GLOBAL` (or `@@global.`) when setting a global variable.

To set a `SESSION` variable to the `GLOBAL` value or a `GLOBAL` value to the compiled-in MySQL default value, use the `DEFAULT` keyword. For example, the following two statements are identical in setting the session value of `max_join_size` [522] to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to `DEFAULT`. In such cases, use of `DEFAULT` results in an error.

It is not permitted to assign the value `DEFAULT` to user-defined variables, and not supported for stored procedure or function parameters or stored program local variables. This results in a syntax error for user-defined variables, and the results are undefined for parameters or local variables.

You can refer to the values of specific global or session system variables in expressions by using one of the `@@`-modifiers. For example, you can retrieve values in a `SELECT` statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as `@@var_name` (that is, when you do not specify `@@global.` or `@@session.`), MySQL returns the session value if it exists and the global value otherwise. (This differs from `SET @@var_name = value`, which always refers to the session value.)

**Note**

Some variables displayed by `SHOW VARIABLES` may not be available using `SELECT @@var_name` syntax; an `Unknown system variable` occurs. As a workaround in such cases, you can use `SHOW VARIABLES LIKE 'var_name'`.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

To display system variables names and values, use the `SHOW VARIABLES` statement. (See [Section 13.7.5.40](#), “`SHOW VARIABLES Syntax`”.)

The following list describes `SET` options that have nonstandard syntax (that is, options that are not set with `name = value` syntax).

- `CHARACTER SET {charset_name | DEFAULT}`

This maps all strings from and to the client with the given mapping. You can add new mappings by editing `sql/convert.cc` in the MySQL source distribution. `SET CHARACTER SET` sets three session system variables: `character_set_client` [486] and `character_set_results` [488] are set to the given character set, and `character_set_connection` [487] to the value of `character_set_database` [487]. See [Section 10.1.4](#), “`Connection Character Sets and Collations`”.

The default mapping can be restored by using the value `DEFAULT`. The default depends on the server configuration.

`ucs2`, `utf16`, and `utf32` cannot be used as a client character set, which means that they do not work for `SET CHARACTER SET`.

- `NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT}`

`SET NAMES` sets the three session system variables `character_set_client` [486], `character_set_connection` [487], and `character_set_results` [488] to the given character set. Setting `character_set_connection` [487] to `charset_name` also sets `collation_connection` [489] to the default collation for `charset_name`. The optional `COLLATE` clause may be used to specify a collation explicitly. See [Section 10.1.4](#), “`Connection Character Sets and Collations`”.

The default mapping can be restored by using a value of `DEFAULT`. The default depends on the server configuration.

`ucs2`, `utf16`, and `utf32` cannot be used as a client character set, which means that they do not work for `SET NAMES`.

- `ONE_SHOT`

This option is a modifier, not a variable. It is *only* for internal use for replication: `mysqlbinlog` uses `SET ONE_SHOT` to modify temporarily the values of character set, collation, and time zone variables to reflect

at rollforward what they were originally. `ONE_SHOT` is for internal use only and is deprecated for MySQL 5.0 and up.

`ONE_SHOT` is intended for use only with the permitted set of variables. It changes the variables as requested, but only for the next non-`SET` statement. After that, the server resets all character set, collation, and time zone-related system variables to their previous values. Example:

```
mysql> SET ONE_SHOT character_set_connection = latin5;

mysql> SET ONE_SHOT collation_connection = latin5_turkish_ci;

mysql> SHOW VARIABLES LIKE '%_connection';
+-----+-----+
| Variable_name          | Value          |
+-----+-----+
| character_set_connection | latin5         |
| collation_connection    | latin5_turkish_ci |
+-----+-----+

mysql> SHOW VARIABLES LIKE '%_connection';
+-----+-----+
| Variable_name          | Value          |
+-----+-----+
| character_set_connection | latin1         |
| collation_connection    | latin1_swedish_ci |
+-----+-----+
```

13.7.5. SHOW Syntax

`SHOW` has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW AUTHORS
SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CONTRIBUTORS
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW MASTER STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW SLAVE HOSTS
```

```
SHOW SLAVE STATUS
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW [FULL] TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]
```

```
like_or_where:
  LIKE 'pattern'
  | WHERE expr
```

If the syntax for a given `SHOW` statement includes a `LIKE 'pattern' [1111]` part, '*pattern*' is a string that can contain the SQL “%” and “_” wildcard characters. The pattern is useful for restricting statement output to matching values.

Several `SHOW` statements also accept a `WHERE` clause that provides more flexibility in specifying which rows to display. See [Section 20.31, “Extensions to SHOW Statements”](#).

Many MySQL APIs (such as PHP) enable you to treat the result returned from a `SHOW` statement as you would a result set from a `SELECT`; see [Chapter 22, Connectors and APIs](#), or your API documentation for more information. In addition, you can work in SQL with results from queries on tables in the `INFORMATION_SCHEMA` database, which you cannot easily do with results from `SHOW` statements. See [Chapter 20, INFORMATION_SCHEMA Tables](#).

13.7.5.1. SHOW AUTHORS Syntax

```
SHOW AUTHORS
```

The `SHOW AUTHORS` statement displays information about the people who work on MySQL. For each author, it displays `Name`, `Location`, and `Comment` values.

This statement is deprecated as of MySQL 5.5.29 and is removed in MySQL 5.6.

13.7.5.2. SHOW BINARY LOGS Syntax

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in [Section 13.4.1.1, “PURGE BINARY LOGS Syntax”](#), that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name          | File_size |
+-----+-----+
| binlog.000015    | 724935   |
| binlog.000016    | 733481   |
+-----+-----+
```

`SHOW MASTER LOGS` is equivalent to `SHOW BINARY LOGS`.

In MySQL 5.5.24 and earlier, the `SUPER` [697] privilege was required to use this statement. Starting with MySQL 5.5.25, a user with the `REPLICATION CLIENT` [697] privilege may also execute this statement.

13.7.5.3. SHOW BINLOG EVENTS Syntax

```
SHOW BINLOG EVENTS
  [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify '*log_name*', the first binary log is displayed.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See [Section 13.2.9, “SELECT Syntax”](#).



Note

Issuing a `SHOW BINLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the binary log (which includes all statements executed by the server that modify data). As an alternative to `SHOW BINLOG EVENTS`, use the `mysqlbinlog` utility to save the binary log to a text file for later examination and analysis. See [Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”](#).



Note

Some events relating to the setting of user and system variables are not included in the output from `SHOW BINLOG EVENTS`. To get complete coverage of events within a binary log, use `mysqlbinlog`.



Note

`SHOW BINLOG EVENTS` does *not* work with relay log files. You can use `SHOW RELAYLOG EVENTS` for this purpose.

13.7.5.4. SHOW CHARACTER SET Syntax

```
SHOW CHARACTER SET
  [LIKE 'pattern' | WHERE expr]
```

The `SHOW CHARACTER SET` statement shows all available character sets. The `LIKE [1111]` clause, if present, indicates which character set names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#). For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description           | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European | latin1_swedish_ci | 1      |
| latin2  | ISO 8859-2 Central   | latin2_general_ci | 1      |
| latin5  | ISO 8859-9 Turkish  | latin5_turkish_ci | 1      |
| latin7  | ISO 8859-13 Baltic  | latin7_general_ci | 1      |
+-----+-----+-----+-----+
```

The `Maxlen` column shows the maximum number of bytes required to store one character.

13.7.5.5. SHOW COLLATION Syntax

```
SHOW COLLATION
  [LIKE 'pattern' | WHERE expr]
```

This statement lists collations supported by the server. By default, the output from `SHOW COLLATION` includes all available collations. The `LIKE [1111]` clause, if present, indicates which collation names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#). For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
```

latin1_german1_ci	latin1	5			0
latin1_swedish_ci	latin1	8	Yes	Yes	0
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	0
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0

The `Collation` and `Charset` columns indicate the names of the collation and the character set with which it is associated. `Id` is the collation ID. `Default` indicates whether the collation is the default for its character set. `Compiled` indicates whether the character set is compiled into the server. `Sortlen` is related to the amount of memory required to sort strings expressed in the character set.

To see the default collation for each character set, use the following statement. `Default` is a reserved word, so to use it as an identifier, it must be quoted as such:

```
mysql> SHOW COLLATION WHERE `Default` = 'Yes';
```

Collation	Charset	Id	Default	Compiled	Sortlen
big5_chinese_ci	big5	1	Yes	Yes	1
dec8_swedish_ci	dec8	3	Yes	Yes	1
cp850_general_ci	cp850	4	Yes	Yes	1
hp8_english_ci	hp8	6	Yes	Yes	1
koi8r_general_ci	koi8r	7	Yes	Yes	1
latin1_swedish_ci	latin1	8	Yes	Yes	1

...

13.7.5.6. SHOW COLUMNS Syntax

```
SHOW [FULL] COLUMNS {FROM | IN} tbl_name [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE expr]
```

`SHOW COLUMNS` displays information about the columns in a given table. It also works for views. The `LIKE [1111]` clause, if present, indicates which column names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31](#), “Extensions to `SHOW` Statements”.

`SHOW COLUMNS` displays information only for those columns for which you have some privilege.

```
mysql> SHOW COLUMNS FROM City;
```

Field	Type	Null	Key	Default	Extra
Id	int(11)	NO	PRI	NULL	auto_increment
Name	char(35)	NO			
Country	char(3)	NO	UNI		
District	char(20)	YES	MUL		
Population	int(11)	NO		0	

5 rows in set (0.00 sec)

If the data types differ from what you expect them to be based on a `CREATE TABLE` statement, note that MySQL sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in [Section 13.1.17.2](#), “Silent Column Specification Changes”.

The `FULL` keyword causes the output to include the column collation and comments, as well as the privileges you have for each column.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. In other words, these two statements are equivalent:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;  
mysql> SHOW COLUMNS FROM mydb.mytable;
```

`SHOW COLUMNS` displays the following values for each table column:

`Field` indicates the column name.

`Type` indicates the column data type.

`Collation` indicates the collation for nonbinary string columns, or `NULL` for other columns. This value is displayed only if you use the `FULL` keyword.

The `Null` field contains `YES` if `NULL` values can be stored in the column, `NO` if not.

The `Key` field indicates whether the column is indexed:

- If `Key` is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
- If `Key` is `PRI`, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.
- If `Key` is `UNI`, the column is the first column of a `UNIQUE` index. (A `UNIQUE` index permits multiple `NULL` values, but you can tell whether the column permits `NULL` by checking the `Null` field.)
- If `Key` is `MUL`, the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

If more than one of the `Key` values applies to a given column of a table, `Key` displays the one with the highest priority, in the order `PRI`, `UNI`, `MUL`.

A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE` index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

The `Default` field indicates the default value that is assigned to the column. This is `NULL` if the column has an explicit default of `NULL`, or if the column definition has no `DEFAULT` clause.

The `Extra` field contains any additional information that is available about a given column. The value is nonempty in these cases: `auto_increment` for columns that have the `AUTO_INCREMENT` attribute; `on update CURRENT_TIMESTAMP` for `TIMESTAMP` columns that have the `ON UPDATE CURRENT_TIMESTAMP` attribute.

`Privileges` indicates the privileges you have for the column. This value is displayed only if you use the `FULL` keyword.

`Comment` indicates any comment the column has. This value is displayed only if you use the `FULL` keyword.

`SHOW FIELDS` is a synonym for `SHOW COLUMNS`. You can also list a table's columns with the `mysqlshow db_name tbl_name` command.

The `DESCRIBE` statement provides information similar to `SHOW COLUMNS`. See [Section 13.8.1, “DESCRIBE Syntax”](#).

The `SHOW CREATE TABLE`, `SHOW TABLE STATUS`, and `SHOW INDEX` statements also provide information about tables. See [Section 13.7.5, “SHOW Syntax”](#).

13.7.5.7. SHOW CONTRIBUTORS Syntax

```
SHOW CONTRIBUTORS
```

The `SHOW CONTRIBUTORS` statement displays information about the people who contribute to MySQL source or to causes that we support. For each contributor, it displays `Name`, `Location`, and `Comment` values.

This statement is deprecated as of MySQL 5.5.29 and is removed in MySQL 5.6.

13.7.5.8. SHOW CREATE DATABASE Syntax

```
SHOW CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
```

Shows the `CREATE DATABASE` statement that creates the given database. If the `SHOW` statement includes an `IF NOT EXISTS` clause, the output too includes such a clause. `SHOW CREATE SCHEMA` is a synonym for `SHOW CREATE DATABASE`.

```
mysql> SHOW CREATE DATABASE test\G
***** 1. row *****
      Database: test
Create Database: CREATE DATABASE `test`
                  /*!40100 DEFAULT CHARACTER SET latin1 */

mysql> SHOW CREATE SCHEMA test\G
***** 1. row *****
      Database: test
Create Database: CREATE DATABASE `test`
                  /*!40100 DEFAULT CHARACTER SET latin1 */
```

`SHOW CREATE DATABASE` quotes table and column names according to the value of the `sql_quote_show_create` [556] option. See [Section 5.1.4, “Server System Variables”](#).

13.7.5.9. SHOW CREATE EVENT Syntax

```
SHOW CREATE EVENT event_name
```

This statement displays the `CREATE EVENT` statement needed to re-create a given event. It requires the `EVENT` [696] privilege for the database from which the event is to be shown. For example (using the same event `e_daily` defined and then altered in [Section 13.7.5.19, “SHOW EVENTS Syntax”](#)):

```
mysql> SHOW CREATE EVENT test.e_daily\G
***** 1. row *****
      Event: e_daily
      sql_mode:
      time_zone: SYSTEM
Create Event: CREATE EVENT `e_daily`
              ON SCHEDULE EVERY 1 DAY
              STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
              ON COMPLETION NOT PRESERVE
              ENABLE
              COMMENT 'Saves total number of sessions then
                      clears the table each day'
              DO BEGIN
                INSERT INTO site_activity.totals (time, total)
                  SELECT CURRENT_TIMESTAMP, COUNT(*)
                  FROM site_activity.sessions;
                DELETE FROM site_activity.sessions;
              END
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

`character_set_client` [486] is the session value of the `character_set_client` [486] system variable when the event was created. `collation_connection` [489] is the session value of the `collation_connection` [489] system variable when the event was created. `Database Collation` is the collation of the database with which the event is associated.

Note that the output reflects the current status of the event (`ENABLE`) rather than the status with which it was created.

13.7.5.10. SHOW CREATE FUNCTION Syntax

```
SHOW CREATE FUNCTION func_name
```

This statement is similar to `SHOW CREATE PROCEDURE` but for stored functions. See [Section 13.7.5.11, “SHOW CREATE PROCEDURE Syntax”](#).

13.7.5.11. SHOW CREATE PROCEDURE Syntax

```
SHOW CREATE PROCEDURE proc_name
```

This statement is a MySQL extension. It returns the exact string that can be used to re-create the named stored procedure. A similar statement, `SHOW CREATE FUNCTION`, displays information about stored functions (see [Section 13.7.5.10, “SHOW CREATE FUNCTION Syntax”](#)).

Both statements require that you be the owner of the routine or have `SELECT` access to the `mysql.proc` table. If you do not have privileges for the routine itself, the value displayed for the `Create Procedure` or `Create Function` field will be `NULL`.

```
mysql> SHOW CREATE PROCEDURE test.simpleproc\G
***** 1. row *****
      Procedure: simpleproc
      sql_mode:
      Create Procedure: CREATE PROCEDURE `simpleproc`(OUT param1 INT)
                        BEGIN
                        SELECT COUNT(*) INTO param1 FROM t;
                        END
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci

mysql> SHOW CREATE FUNCTION test.hello\G
***** 1. row *****
      Function: hello
      sql_mode:
      Create Function: CREATE FUNCTION `hello`(s CHAR(20))
                      RETURNS CHAR(50)
                      RETURN CONCAT('Hello, ',s, '!')
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

`character_set_client` [486] is the session value of the `character_set_client` [486] system variable when the routine was created. `collation_connection` [489] is the session value of the `collation_connection` [489] system variable when the routine was created. `Database Collation` is the collation of the database with which the routine is associated.

13.7.5.12. SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows the `CREATE TABLE` statement that creates the given table. To use this statement, you must have some privilege for the table. This statement also works with views.

```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
      Table: t
Create Table: CREATE TABLE t (
  id INT(11) default NULL auto_increment,
  s char(60) default NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM
```

`SHOW CREATE TABLE` quotes table and column names according to the value of the `sql_quote_show_create` [556] option. See [Section 5.1.4, “Server System Variables”](#).

13.7.5.13. SHOW CREATE TRIGGER Syntax

```
SHOW CREATE TRIGGER trigger_name
```

This statement shows a `CREATE TRIGGER` statement that creates the given trigger.

```
mysql> SHOW CREATE TRIGGER ins_sum\G
***** 1. row *****
      Trigger: ins_sum
      sql_mode:
SQL Original Statement: CREATE DEFINER=`bob`@`localhost` TRIGGER ins_sum
                        BEFORE INSERT ON account
                        FOR EACH ROW SET @sum = @sum + NEW.amount
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

You can also obtain information about trigger objects from `INFORMATION_SCHEMA`, which contains a `TRIGGERS` table. See [Section 20.25, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

13.7.5.14. SHOW CREATE VIEW Syntax

```
SHOW CREATE VIEW view_name
```

This statement shows a `CREATE VIEW` statement that creates the given view.

```
mysql> SHOW CREATE VIEW v\G
***** 1. row *****
      View: v
      Create View: CREATE ALGORITHM=UNDEFINED
                  DEFINER=`bob`@`localhost`
                  SQL SECURITY DEFINER VIEW
                  `v` AS select 1 AS `a`,2 AS `b`
character_set_client: latin1
collation_connection: latin1_swedish_ci
```

`character_set_client` [486] is the session value of the `character_set_client` [486] system variable when the view was created. `collation_connection` [489] is the session value of the `collation_connection` [489] system variable when the view was created.

Use of `SHOW CREATE VIEW` requires the `SHOW VIEW` [697] privilege and the `SELECT` [697] privilege for the view in question.

You can also obtain information about view objects from `INFORMATION_SCHEMA`, which contains a `VIEWS` table. See [Section 20.27, “The INFORMATION_SCHEMA VIEWS Table”](#).

MySQL lets you use different `sql_mode` [556] settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` [617] SQL mode to ensure MySQL correctly interprets the

standard SQL concatenation operator, the double bar (||), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` [556] setting to a value different from `ANSI` [617] could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` [1102] function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as coll;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW CREATE VIEW test.v\G
***** 1. row *****
      View: v
      Create View: CREATE VIEW "v" AS select concat('a','b') AS "coll"
      ...
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` [556] will not affect the results from the view. However an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

13.7.5.15. SHOW DATABASES Syntax

```
SHOW {DATABASES | SCHEMAS}
      [LIKE 'pattern' | WHERE expr]
```

`SHOW DATABASES` lists the databases on the MySQL server host. `SHOW SCHEMAS` is a synonym for `SHOW DATABASES`. The `LIKE` [1111] clause, if present, indicates which database names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31](#), “Extensions to `SHOW` Statements”.

You see only those databases for which you have some kind of privilege, unless you have the global `SHOW DATABASES` privilege. You can also get this list using the `mysqlshow` command.

If the server was started with the `--skip-show-database` [463] option, you cannot use this statement at all unless you have the `SHOW DATABASES` [697] privilege.

MySQL implements databases as directories in the data directory, so this statement simply lists directories in that location. However, the output may include names of directories that do not correspond to actual databases.

13.7.5.16. SHOW ENGINE Syntax

```
SHOW ENGINE engine_name {STATUS | MUTEX}
```

`SHOW ENGINE` displays operational information about a storage engine. The following statements currently are supported:

```
SHOW ENGINE INNODB STATUS
SHOW ENGINE INNODB MUTEX
SHOW ENGINE {NDB | NDBCLUSTER} STATUS
SHOW ENGINE PERFORMANCE_SCHEMA STATUS
```

`SHOW ENGINE INNODB STATUS` displays extensive information from the standard `InnoDB` Monitor about the state of the `InnoDB` storage engine. For information about the standard monitor and other `InnoDB` Monitors that provide information about `InnoDB` processing, see [Section 14.3.14.2](#), “`SHOW ENGINE INNODB STATUS` and the `InnoDB` Monitors”.

`SHOW ENGINE INNODB MUTEX` displays InnoDB mutex statistics. The statement displays the following fields:

- `Type`

Always `InnoDB`.

- `Name`

The source file where the mutex is implemented, and the line number in the file where the mutex is created. The line number may change depending on your version of MySQL.

- `Status`

The mutex status. This field displays several values if `UNIV_DEBUG` was defined at MySQL compilation time (for example, in `include/univ.i` in the InnoDB part of the MySQL source tree). If `UNIV_DEBUG` was not defined, the statement displays only the `os_waits` value. In the latter case (without `UNIV_DEBUG`), the information on which the output is based is insufficient to distinguish regular mutexes and mutexes that protect rw-locks (which permit multiple readers or a single writer). Consequently, the output may appear to contain multiple rows for the same mutex.

- `count` indicates how many times the mutex was requested.
- `spin_waits` indicates how many times the spinlock had to run.
- `spin_rounds` indicates the number of spinlock rounds. (`spin_rounds` divided by `spin_waits` provides the average round count.)
- `os_waits` indicates the number of operating system waits. This occurs when the spinlock did not work (the mutex was not locked during the spinlock and it was necessary to yield to the operating system and wait).
- `os_yields` indicates the number of times a the thread trying to lock a mutex gave up its timeslice and yielded to the operating system (on the presumption that permitting other threads to run will free the mutex so that it can be locked).
- `os_wait_times` indicates the amount of time (in ms) spent in operating system waits, if the `timed_mutexes` [567] system variable is 1 (`ON`). If `timed_mutexes` [567] is 0 (`OFF`), timing is disabled, so `os_wait_times` is 0. `timed_mutexes` [567] is off by default.

Information from this statement can be used to diagnose system problems. For example, large values of `spin_waits` and `spin_rounds` may indicate scalability problems.

Use `SHOW ENGINE PERFORMANCE_SCHEMA STATUS` to inspect the internal operation of the Performance Schema code:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
***** 3. row *****
  Type: performance_schema
  Name: events_waits_history.row_size
  Status: 76
***** 4. row *****
  Type: performance_schema
  Name: events_waits_history.row_count
  Status: 10000
***** 5. row *****
  Type: performance_schema
  Name: events_waits_history.memory
```

```
Status: 760000
...
***** 57. row *****
Type: performance_schema
Name: performance_schema.memory
Status: 26459600
...
```

This statement is intended to help the DBA understand the effects that different Performance Schema options have on memory requirements.

`Name` values consist of two parts, which name an internal buffer and a buffer attribute, respectively. Interpret buffer names as follows:

- An internal buffer that is not exposed as a table is named within parentheses. Examples: `(pfs_cond_class).row_size`, `(pfs_mutex_class).memory`.
- An internal buffer that is exposed as a table in the `performance_schema` database is named after the table, without parentheses. Examples: `events_waits_history.row_size`, `mutex_instances.row_count`.
- A value that applies to the Performance Schema as a whole begins with `performance_schema`. Example: `performance_schema.memory`.

Buffer attributes have these meanings:

- `row_size` is the size of the internal record used by the implementation, such as the size of a row in a table. `row_size` values cannot be changed.
- `row_count` is the number of internal records, such as the number of rows in a table. `row_count` values can be changed using Performance Schema configuration options.
- For a table, `tbl_name.memory` is the product of `row_size` and `row_count`. For the Performance Schema as a whole, `performance_schema.memory` is the sum of all the memory used (the sum of all other `memory` values).

In some cases, there is a direct relationship between a Performance Schema configuration parameter and a `SHOW ENGINE` value. For example, `events_waits_history_long.row_count` corresponds to `performance_schema_events_waits_history_long_size` [2554]. In other cases, the relationship is more complex. For example, `events_waits_history.row_count` corresponds to `performance_schema_events_waits_history_size` [2554] (the number of rows per thread) multiplied by `performance_schema_max_thread_instances` [2558] (the number of threads).

If the server has the `NDBCLUSTER` storage engine enabled, `SHOW ENGINE NDB STATUS` displays cluster status information such as the number of connected data nodes, the cluster connectstring, and cluster binlog epochs, as well as counts of various Cluster API objects created by the MySQL Server when connected to the cluster. Sample output from this statement is shown here:

```
mysql> SHOW ENGINE NDB STATUS;
+-----+-----+-----+
| Type      | Name                                | Status                                |
+-----+-----+-----+
| ndbcluster | connection                          | cluster_node_id=7,
connected_host=192.168.0.103, connected_port=1186, number_of_data_nodes=4,
number_of_ready_data_nodes=3, connect_count=0 |
| ndbcluster | NdbTransaction                      | created=6, free=0, sizeof=212 |
| ndbcluster | NdbOperation                        | created=8, free=8, sizeof=660 |
| ndbcluster | NdbIndexScanOperation              | created=1, free=1, sizeof=744 |
| ndbcluster | NdbIndexOperation                  | created=0, free=0, sizeof=664 |
| ndbcluster | NdbRecAttr                          | created=1285, free=1285, sizeof=60 |
| ndbcluster | NdbApiSignal                        | created=16, free=16, sizeof=136 |
+-----+-----+-----+
```

```

| ndbcluster | NdbLabel          | created=0, free=0, sizeof=196 |
| ndbcluster | NdbBranch         | created=0, free=0, sizeof=24  |
| ndbcluster | NdbSubroutine     | created=0, free=0, sizeof=68  |
| ndbcluster | NdbCall           | created=0, free=0, sizeof=16  |
| ndbcluster | NdbBlob           | created=1, free=1, sizeof=264 |
| ndbcluster | NdbReceiver       | created=4, free=0, sizeof=68  |
| ndbcluster | binlog            | latest_epoch=155467, latest_trans_epoch=148126,
|               |                   | latest_received_binlog_epoch=0, latest_handled_binlog_epoch=0,
|               |                   | latest_applied_binlog_epoch=0 |
+-----+-----+-----+

```

The rows with `connection` and `binlog` in the `Name` column were added to the output of this statement in MySQL 5.1. The `Status` column in each of these rows provides information about the MySQL server's connection to the cluster and about the cluster binary log's status, respectively. The `Status` information is in the form of comma-delimited set of name/value pairs.

The `connection` row's `Status` column contains the name/value pairs described in the following table.

Name	Value
<code>cluster_node_id</code>	The node ID of the MySQL server in the cluster
<code>connected_host</code>	The host name or IP address of the cluster management server to which the MySQL server is connected
<code>connected_port</code>	The port used by the MySQL server to connect to the management server (<code>connected_host</code>)
<code>number_of_data_nodes</code>	The number of data nodes configured for the cluster (that is, the number of <code>[ndbd]</code> sections in the cluster <code>config.ini</code> file)
<code>number_of_ready_data_nodes</code>	The number of data nodes in the cluster that are actually running
<code>connect_count</code>	The number of times this <code>mysqld</code> has connected or reconnected to cluster data nodes

The `binlog` row's `Status` column contains information relating to MySQL Cluster Replication. The name/value pairs it contains are described in the following table.

Name	Value
<code>latest_epoch</code>	The most recent epoch most recently run on this MySQL server (that is, the sequence number of the most recent transaction run on the server)
<code>latest_trans_epoch</code>	The most recent epoch processed by the cluster's data nodes
<code>latest_received_binlog_epoch</code>	The most recent epoch received by the binlog thread
<code>latest_handled_binlog_epoch</code>	The most recent epoch processed by the binlog thread (for writing to the binlog)
<code>latest_applied_binlog_epoch</code>	The most recent epoch actually written to the binlog

See [Section 17.6, "MySQL Cluster Replication"](#), for more information.

The remaining rows from the output of `SHOW ENGINE NDB STATUS` which are most likely to prove useful in monitoring the cluster are listed here by `Name`:

- `NdbTransaction`: The number and size of `NdbTransaction` objects that have been created. An `NdbTransaction` is created each time a table schema operation (such as `CREATE TABLE` or `ALTER TABLE`) is performed on an `NDB` table.

- `NdbOperation`: The number and size of `NdbOperation` objects that have been created.
- `NdbIndexScanOperation`: The number and size of `NdbIndexScanOperation` objects that have been created.
- `NdbIndexOperation`: The number and size of `NdbIndexOperation` objects that have been created.
- `NdbRecAttr`: The number and size of `NdbRecAttr` objects that have been created. In general, one of these is created each time a data manipulation statement is performed by an SQL node.
- `NdbBlob`: The number and size of `NdbBlob` objects that have been created. An `NdbBlob` is created for each new operation involving a `BLOB` column in an `NDB` table.
- `NdbReceiver`: The number and size of any `NdbReceiver` object that have been created. The number in the `created` column is the same as the number of data nodes in the cluster to which the MySQL server has connected.



Note

`SHOW ENGINE NDB STATUS` returns an empty result if no operations involving `NDB` tables have been performed during the current session by the MySQL client accessing the SQL node on which this statement is run.

13.7.5.17. SHOW ENGINES Syntax

```
SHOW [STORAGE] ENGINES
```

`SHOW ENGINES` displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is.

```
mysql> SHOW ENGINES\G
***** 1. row *****
  Engine: MEMORY
  Support: YES
  Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
  XA: NO
  Savepoints: NO
***** 2. row *****
  Engine: MyISAM
  Support: YES
  Comment: MyISAM storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
***** 3. row *****
  Engine: InnoDB
  Support: DEFAULT
  Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
  XA: YES
  Savepoints: YES
***** 4. row *****
  Engine: EXAMPLE
  Support: YES
  Comment: Example storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
***** 5. row *****
  Engine: ARCHIVE
  Support: YES
  Comment: Archive storage engine
```

```

Transactions: NO
      XA: NO
      Savepoints: NO
***** 6. row *****
      Engine: CSV
      Support: YES
      Comment: CSV storage engine
Transactions: NO
      XA: NO
      Savepoints: NO
***** 7. row *****
      Engine: BLACKHOLE
      Support: YES
      Comment: /dev/null storage engine (anything you write »
              to it disappears)
Transactions: NO
      XA: NO
      Savepoints: NO
***** 8. row *****
      Engine: FEDERATED
      Support: YES
      Comment: Federated MySQL storage engine
Transactions: NO
      XA: NO
      Savepoints: NO
***** 9. row *****
      Engine: MRG_MYISAM
      Support: YES
      Comment: Collection of identical MyISAM tables
Transactions: NO
      XA: NO
      Savepoints: NO

```

The output from `SHOW ENGINES` may vary according to the MySQL version used and other factors. The values shown in the `Support` column indicate the server's level of support for the storage engine, as shown in the following table.

Value	Meaning
YES	The engine is supported and is active
DEFAULT	Like YES, plus this is the default engine
NO	The engine is not supported
DISABLED	The engine is supported but has been disabled

A value of `NO` means that the server was compiled without support for the engine, so it cannot be enabled at runtime.

A value of `DISABLED` occurs either because the server was started with an option that disables the engine, or because not all options required to enable it were given. In the latter case, the error log file should contain a reason indicating why the option is disabled. See [Section 5.2.2, “The Error Log”](#).

You might also see `DISABLED` for a storage engine if the server was compiled to support it, but was started with a `--skip-engine_name` option. For the `NDBCLUSTER` storage engine, `DISABLED` means the server was compiled with support for MySQL Cluster, but was not started with the `--ndbcluster` [\[2142\]](#) option.

All MySQL servers support `MyISAM` tables, because `MyISAM` is the default storage engine. It is not possible to disable `MyISAM`.

The `Transactions`, `XA`, and `Savepoints` columns indicate whether the storage engine supports transactions, XA transactions, and savepoints, respectively.

13.7.5.18. SHOW ERRORS Syntax

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

This statement is similar to [SHOW WARNINGS](#), except that it displays information only for errors, rather than for errors, warnings, and notes.

The [LIMIT](#) clause has the same syntax as for the [SELECT](#) statement. See [Section 13.2.9, “SELECT Syntax”](#).

The [SHOW COUNT\(*\) ERRORS](#) statement displays the number of errors. You can also retrieve this number from the [error_count \[498\]](#) variable:

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

[SHOW ERRORS](#) and [error_count \[498\]](#) apply only to errors, not warnings or notes. In other respects, they are similar to [SHOW WARNINGS](#) and [warning_count \[573\]](#). In particular, [SHOW ERRORS](#) cannot display information for more than [max_error_count \[520\]](#) messages, and [error_count \[498\]](#) can exceed the value of [max_error_count \[520\]](#) if the number of errors exceeds [max_error_count \[520\]](#).

For more information, see [Section 13.7.5.41, “SHOW WARNINGS Syntax”](#).

13.7.5.19. SHOW EVENTS Syntax

```
SHOW EVENTS [{FROM | IN} schema_name]
           [LIKE 'pattern' | WHERE expr]
```

This statement displays information about Event Manager events. It requires the [EVENT \[696\]](#) privilege for the database from which the events are to be shown.

In its simplest form, [SHOW EVENTS](#) lists all of the events in the current schema:

```
mysql> SELECT CURRENT_USER(), SCHEMA();
+-----+-----+
| CURRENT_USER() | SCHEMA() |
+-----+-----+
| jon@ghidora    | myschema |
+-----+-----+
1 row in set (0.00 sec)

mysql> SHOW EVENTS\G
***** 1. row *****
      Db: myschema
      Name: e_daily
      Definer: jon@ghidora
      Time zone: SYSTEM
      Type: RECURRING
      Execute at: NULL
      Interval value: 10
      Interval field: SECOND
      Starts: 2006-02-09 10:41:23
      Ends: NULL
      Status: ENABLED
      Originator: 0
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

To see events for a specific schema, use the [FROM](#) clause. For example, to see events for the `test` schema, use the following statement:

```
SHOW EVENTS FROM test;
```

The [LIKE \[1111\]](#) clause, if present, indicates which event names to match. The [WHERE](#) clause can be given to select rows using more general conditions, as discussed in [Section 20.31](#), “Extensions to [SHOW Statements](#)”.

[SHOW EVENTS](#) output has the following columns:

- **Db:** The schema (database) on which the event is defined.
- **Name:** The name of the event.
- **Time zone:** The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is [SYSTEM](#).
- **Definer:** The account of the user who created the event, in '[user_name](#)'@'[host_name](#)' format.
- **Type:** The event repetition type, either [ONE TIME](#) (transient) or [RECURRING](#) (repeating).
- **Execute At:** The date and time when a transient event is set to execute. Shown as a [DATETIME](#) value.

For a recurring event, the value of this column is always [NULL](#).

- **Interval Value:** For a recurring event, the number of intervals to wait between event executions. For a transient event, the value of this column is always [NULL](#).
- **Interval Field:** The time units used for the interval which a recurring event waits before repeating. For a transient event, the value of this column is always [NULL](#).
- **Starts:** The start date and time for a recurring event. This is displayed as a [DATETIME](#) value, and is [NULL](#) if no start date and time are defined for the event.

For a transient event, this column is always [NULL](#).

- **Ends:** The end date and time for a recurring event. This is displayed as a [DATETIME](#) value, and defaults to [NULL](#) if no end date and time is defined for the event.

For a transient event, this column is always [NULL](#).

- **Status:** The event status. One of [ENABLED](#), [DISABLED](#), or [SLAVESIDE_DISABLED](#).

[SLAVESIDE_DISABLED](#) indicates that the creation of the event occurred on another MySQL server acting as a replication master and replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave.

- **Originator:** The server ID of the MySQL server on which the event was created. Defaults to 0.
- [character_set_client \[486\]](#) is the session value of the [character_set_client \[486\]](#) system variable when the routine was created. [collation_connection \[489\]](#) is the session value of the [collation_connection \[489\]](#) system variable when the routine was created. [Database Collation](#) is the collation of the database with which the routine is associated.

For more information about [SLAVE_DISABLED](#) and the [Originator](#) column, see [Section 16.4.1.10](#), “[Replication of Invoked Features](#)”.

The event action statement is not shown in the output of [SHOW EVENTS](#). Use [SHOW CREATE EVENT](#) or the [INFORMATION_SCHEMA.EVENTS](#) table.

Times displayed by `SHOW EVENTS` are given in the event time zone, as discussed in [Section 19.4.4, “Event Metadata”](#).

The columns in the output of `SHOW EVENTS` are similar to, but not identical to the columns in the `INFORMATION_SCHEMA.EVENTS` table. See [Section 20.7, “The INFORMATION_SCHEMA EVENTS Table”](#).

13.7.5.20. SHOW FUNCTION CODE Syntax

```
SHOW FUNCTION CODE func_name
```

This statement is similar to `SHOW PROCEDURE CODE` but for stored functions. See [Section 13.7.5.28, “SHOW PROCEDURE CODE Syntax”](#).

13.7.5.21. SHOW FUNCTION STATUS Syntax

```
SHOW FUNCTION STATUS
  [LIKE 'pattern' | WHERE expr]
```

This statement is similar to `SHOW PROCEDURE STATUS` but for stored functions. See [Section 13.7.5.29, “SHOW PROCEDURE STATUS Syntax”](#).

13.7.5.22. SHOW GRANTS Syntax

```
SHOW GRANTS [FOR user]
```

This statement lists the `GRANT` statement or statements that must be issued to duplicate the privileges that are granted to a MySQL user account. The account is named using the same format as for the `GRANT` statement; for example, `'jeffrey'@'localhost'`. If you specify only the user name part of the account name, a host name part of `'%'` is used. For additional information about specifying account names, see [Section 13.7.1.3, “GRANT Syntax”](#).

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+-----+-----+
| Grants for root@localhost |
+-----+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-----+-----+
```

To list the privileges granted to the account that you are using to connect to the server, you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

If `SHOW GRANTS FOR CURRENT_USER` (or any of the equivalent syntaxes) is used in `DEFINER` context, such as within a stored procedure that is defined with `SQL SECURITY DEFINER`, the grants displayed are those of the definer and not the invoker.

`SHOW GRANTS` displays only the privileges granted explicitly to the named account. Other privileges might be available to the account, but they are not displayed. For example, if an anonymous account exists, the named account might be able to use its privileges, but `SHOW GRANTS` will not display them.

`SHOW GRANTS` requires the `SELECT [697]` privilege for the `mysql` database, except to see the privileges for the current user.

13.7.5.23. SHOW INDEX Syntax

```
SHOW {INDEX | INDEXES | KEYS}
  {FROM | IN} tbl_name
  [{FROM | IN} db_name]
```

```
[WHERE expr]
```

`SHOW INDEX` returns table index information. The format resembles that of the `SQLStatistics` call in ODBC. This statement requires some privilege for any column in the table.

`SHOW INDEX` returns the following fields:

- `Table`

The name of the table.

- `Non_unique`

0 if the index cannot contain duplicates, 1 if it can.

- `Key_name`

The name of the index. If the index is the primary key, the name is always `PRIMARY`.

- `Seq_in_index`

The column sequence number in the index, starting with 1.

- `Column_name`

The column name.

- `Collation`

How the column is sorted in the index. In MySQL, this can have values "A" (Ascending) or `NULL` (Not sorted).

- `Cardinality`

An estimate of the number of unique values in the index. This is updated by running `ANALYZE TABLE` or `myisamchk -a`. `Cardinality` is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- `Sub_part`

The number of indexed characters if the column is only partly indexed, `NULL` if the entire column is indexed.

- `Packed`

Indicates how the key is packed. `NULL` if it is not.

- `Null`

Contains `YES` if the column may contain `NULL` values and `' '` if not.

- `Index_type`

The index method used (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

- `Comment`

Information about the index not described in its own column, such as `disabled` if the index is disabled.

- `Index_comment`

Any comment provided for the index with a `COMMENT` attribute when the index was created.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#).

You can also list a table's indexes with the `mysqlshow -k db_name tbl_name` command.

13.7.5.24. SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

This statement provides status information about the binary log files of the master. It requires either the `SUPER [697]` or `REPLICATION CLIENT [697]` privilege.

Example:

```
mysql> SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.003 | 73       | test         | manual,mysql      |
+-----+-----+-----+-----+
```

13.7.5.25. SHOW OPEN TABLES Syntax

```
SHOW OPEN TABLES [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE expr]
```

`SHOW OPEN TABLES` lists the non-`TEMPORARY` tables that are currently open in the table cache. See [Section 8.4.3.1, “How MySQL Opens and Closes Tables”](#). The `FROM` clause, if present, restricts the tables shown to those present in the `db_name` database. The `LIKE [1111]` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#).

`SHOW OPEN TABLES` returns the following columns:

- `Database`

The database containing the table.

- `Table`

The table name.

- `In_use`

The number of table locks or lock requests there are for the table. For example, if one client acquires a lock for a table using `LOCK TABLE t1 WRITE`, `In_use` will be 1. If another client issues `LOCK TABLE t1 WRITE` while the table remains locked, the client will block waiting for the lock, but the lock request causes `In_use` to be 2. If the count is zero, the table is open but not currently being used. `In_use` is also increased by the `HANDLER ... OPEN` statement and decreased by `HANDLER ... CLOSE`.

- `Name_locked`

Whether the table name is locked. Name locking is used for operations such as dropping or renaming tables.

If you have no privileges for a table, it does not show up in the output from `SHOW OPEN TABLES`.

13.7.5.26. SHOW PLUGINS Syntax

```
SHOW PLUGINS
```

`SHOW PLUGINS` displays information about server plugins. Plugin information is also available in the `INFORMATION_SCHEMA.PLUGINS` table. See [Section 20.13](#), “The `INFORMATION_SCHEMA.PLUGINS` Table”.

Example of `SHOW PLUGINS` output:

```
mysql> SHOW PLUGINS\G
***** 1. row *****
  Name: binlog
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
***** 2. row *****
  Name: CSV
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
***** 3. row *****
  Name: MEMORY
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
***** 4. row *****
  Name: MyISAM
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
...
```

`SHOW PLUGINS` returns the following columns:

- **Name:** The name used to refer to the plugin in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`.
- **Status:** The plugin status, one of `ACTIVE`, `INACTIVE`, `DISABLED`, or `DELETED`.
- **Type:** The type of plugin, such as `STORAGE ENGINE`, `INFORMATION_SCHEMA`, or `AUTHENTICATION`.
- **Library:** The name of the plugin shared object file. This is the name used to refer to the plugin file in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`. This file is located in the directory named by the `plugin_dir` [537] system variable. If the library name is `NULL`, the plugin is compiled in and cannot be uninstalled with `UNINSTALL PLUGIN`.
- **License:** How the plugin is licensed; for example, `GPL`.

For plugins installed with `INSTALL PLUGIN`, the `Name` and `Library` values are also registered in the `mysql.plugin` table.

For information about plugin data structures that form the basis of the information displayed by `SHOW PLUGINS`, see [Section 23.2, “The MySQL Plugin API”](#).

13.7.5.27. SHOW PRIVILEGES Syntax

```
SHOW PRIVILEGES
```

`SHOW PRIVILEGES` shows the list of system privileges that the MySQL server supports. The exact list of privileges depends on the version of your server.

```
mysql> SHOW PRIVILEGES\G
***** 1. row *****
Privilege: Alter
  Context: Tables
  Comment: To alter the table
***** 2. row *****
Privilege: Alter routine
  Context: Functions,Procedures
  Comment: To alter or drop stored functions/procedures
***** 3. row *****
Privilege: Create
  Context: Databases,Tables,Indexes
  Comment: To create new databases and tables
***** 4. row *****
Privilege: Create routine
  Context: Databases
  Comment: To use CREATE FUNCTION/PROCEDURE
***** 5. row *****
Privilege: Create temporary tables
  Context: Databases
  Comment: To use CREATE TEMPORARY TABLE
...

```

Privileges belonging to a specific user are displayed by the `SHOW GRANTS` statement. See [Section 13.7.5.22, “SHOW GRANTS Syntax”](#), for more information.

13.7.5.28. SHOW PROCEDURE CODE Syntax

```
SHOW PROCEDURE CODE proc_name
```

This statement is a MySQL extension that is available only for servers that have been built with debugging support. It displays a representation of the internal implementation of the named stored procedure. A similar statement, `SHOW FUNCTION CODE`, displays information about stored functions (see [Section 13.7.5.20, “SHOW FUNCTION CODE Syntax”](#)).

Both statements require that you be the owner of the routine or have `SELECT` access to the `mysql.proc` table.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one “instruction” in the routine. The first column is `Pos`, which is an ordinal number beginning with 0. The second column is `Instruction`, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE p1 ()
-> BEGIN
->   DECLARE fanta INT DEFAULT 55;
->   DROP TABLE t2;
->   LOOP
->     INSERT INTO t3 VALUES (fanta);
->   END LOOP;
-> END//
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SHOW PROCEDURE CODE p1//
+-----+
| Pos | Instruction |
+-----+
| 0 | set fanta@0 55 |
| 1 | stmt 9 "DROP TABLE t2" |
| 2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
| 3 | jump 2 |
+-----+
4 rows in set (0.00 sec)
```

In this example, the nonexecutable `BEGIN` and `END` statements have disappeared, and for the `DECLARE variable_name` statement, only the executable part appears (the part where the default is assigned). For each statement that is taken from source, there is a code word `stmt` followed by a type (9 means `DROP`, 5 means `INSERT`, and so on). The final row contains an instruction `jump 2`, meaning `GOTO instruction #2`.

13.7.5.29. SHOW PROCEDURE STATUS Syntax

```
SHOW PROCEDURE STATUS
[LIKE 'pattern' | WHERE expr]
```

This statement is a MySQL extension. It returns characteristics of a stored procedure, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW FUNCTION STATUS`, displays information about stored functions (see [Section 13.7.5.21](#), “`SHOW FUNCTION STATUS Syntax`”).

The `LIKE [1111]` clause, if present, indicates which procedure or function names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31](#), “`Extensions to SHOW Statements`”.

```
mysql> SHOW PROCEDURE STATUS LIKE 'spl'\G
***** 1. row *****
      Db: test
      Name: spl
      Type: PROCEDURE
      Definer: testuser@localhost
      Modified: 2004-08-03 15:29:37
      Created: 2004-08-03 15:29:37
      Security_type: DEFINER
      Comment:
character_set_client: latin1
collation_connection: latin1_swedish_ci
      Database Collation: latin1_swedish_ci
```

`character_set_client [486]` is the session value of the `character_set_client [486]` system variable when the routine was created. `collation_connection [489]` is the session value of the `collation_connection [489]` system variable when the routine was created. `Database Collation` is the collation of the database with which the routine is associated.

You can also get information about stored routines from the `ROUTINES` table in `INFORMATION_SCHEMA`. See [Section 20.17](#), “`The INFORMATION_SCHEMA ROUTINES Table`”.

13.7.5.30. SHOW PROCESSLIST Syntax

```
SHOW [FULL] PROCESSLIST
```

`SHOW PROCESSLIST` shows you which threads are running. You can also get this information from the `INFORMATION_SCHEMA PROCESSLIST` table or the `mysqladmin processlist` command. If you have the `PROCESS [696]` privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). If you do not use the `FULL` keyword, only the first 100 characters of each statement are shown in the `Info` field.

This statement is very useful if you get the “too many connections” error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the [SUPER \[697\]](#) privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Threads can be killed with the [KILL](#) statement. See [Section 13.7.6.4, “KILL Syntax”](#).

Here is an example of [SHOW PROCESSLIST](#) output:

```
mysql> SHOW FULL PROCESSLIST\G
***** 1. row *****
Id: 1
User: system user
Host:
db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
***** 2. row *****
Id: 2
User: system user
Host:
db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave
      I/O thread to update it
Info: NULL
***** 3. row *****
Id: 3112
User: replikator
Host: artemis:2204
db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
***** 4. row *****
Id: 3113
User: replikator
Host: iconnect2:45781
db: NULL
Command: Binlog Dump
Time: 2086
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
***** 5. row *****
Id: 3123
User: stefan
Host: localhost
db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
5 rows in set (0.00 sec)
```

The columns produced by [SHOW PROCESSLIST](#) have the following meanings:

- [Id](#)

The connection identifier.

- [User](#)

The MySQL user who issued the statement. If this is `system user`, it refers to a nonclient thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. `unauthenticated user` refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet been done. `event_scheduler` refers to the thread that monitors scheduled events. For `system user`, there is no host specified in the `Host` column.

- `Host`

The host name of the client issuing the statement (except for `system user` where there is no host). `SHOW PROCESSLIST` reports the host name for TCP/IP connections in `host_name:client_port` format to make it easier to determine which client is doing what.

- `db`

The default database, if one is selected, otherwise `NULL`.

- `Command`

The type of command the thread is executing. For descriptions for thread commands, see [Section 8.12.5, “Examining Thread Information”](#). The value of this column corresponds to the `COM_XXX` commands of the client/server protocol and `Com_XXX` status variables. See [Section 5.1.6, “Server Status Variables”](#)

- `Time`

The time in seconds that the thread has been in its current state.

- `State`

An action, event, or state that indicates what the thread is doing. Descriptions for `State` values can be found at [Section 8.12.5, “Examining Thread Information”](#).

Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

For the `SHOW PROCESSLIST` statement, the value of `State` is `NULL`.

- `Info`

The statement the thread is executing, or `NULL` if it is not executing any statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `Info` value shows the `SELECT` statement.

13.7.5.31. SHOW PROFILE Syntax

```
SHOW PROFILE [type [, type] ... ]
  [FOR QUERY n]
  [LIMIT row_count [OFFSET offset]]
```

type:

```
ALL
| BLOCK IO
| CONTEXT SWITCHES
| CPU
| IPC
| MEMORY
| PAGE FAULTS
```

```
| SOURCE
| SWAPS
```

The `SHOW PROFILE` and `SHOW PROFILES` statements display profiling information that indicates resource usage for statements executed during the course of the current session.

Profiling is controlled by the `profiling` [539] session variable, which has a default value of 0 (`OFF`). Profiling is enabled by setting `profiling` [539] to 1 or `ON`:

```
mysql> SET profiling = 1;
```

`SHOW PROFILES` displays a list of the most recent statements sent to the server. The size of the list is controlled by the `profiling_history_size` [539] session variable, which has a default value of 15. The maximum value is 100. Setting the value to 0 has the practical effect of disabling profiling.

All statements are profiled except `SHOW PROFILE` and `SHOW PROFILES`, so you will find neither of those statements in the profile list. Malformed statements are profiled. For example, `SHOW PROFILING` is an illegal statement, and a syntax error occurs if you try to execute it, but it will show up in the profiling list.

`SHOW PROFILE` displays detailed information about a single statement. Without the `FOR QUERY n` clause, the output pertains to the most recently executed statement. If `FOR QUERY n` is included, `SHOW PROFILE` displays information for statement `n`. The values of `n` correspond to the `Query_ID` values displayed by `SHOW PROFILES`.

The `LIMIT row_count` clause may be given to limit the output to `row_count` rows. If `LIMIT` is given, `OFFSET offset` may be added to begin the output `offset` rows into the full set of rows.

By default, `SHOW PROFILE` displays `Status` and `Duration` columns. The `Status` values are like the `State` values displayed by `SHOW PROCESSLIST`, although there might be some minor differences in interpretation for the two statements for some status values (see [Section 8.12.5, “Examining Thread Information”](#)).

Optional `type` values may be specified to display specific additional types of information:

- `ALL` displays all information
- `BLOCK IO` displays counts for block input and output operations
- `CONTEXT SWITCHES` displays counts for voluntary and involuntary context switches
- `CPU` displays user and system CPU usage times
- `IPC` displays counts for messages sent and received
- `MEMORY` is not currently implemented
- `PAGE FAULTS` displays counts for major and minor page faults
- `SOURCE` displays the names of functions from the source code, together with the name and line number of the file in which the function occurs
- `SWAPS` displays swap counts

Profiling is enabled per session. When a session ends, its profiling information is lost.

```
mysql> SELECT @@profiling;
+-----+
| @@profiling |
+-----+
|             0 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SET profiling = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE t1 (id INT);
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROFILES;
+-----+-----+-----+
| Query_ID | Duration | Query |
+-----+-----+-----+
| 0 | 0.000088 | SET PROFILING = 1 |
| 1 | 0.000136 | DROP TABLE IF EXISTS t1 |
| 2 | 0.011947 | CREATE TABLE t1 (id INT) |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SHOW PROFILE;
+-----+-----+
| Status | Duration |
+-----+-----+
| checking permissions | 0.000040 |
| creating table | 0.000056 |
| After create | 0.011363 |
| query end | 0.000375 |
| freeing items | 0.000089 |
| logging slow query | 0.000019 |
| cleaning up | 0.000005 |
+-----+-----+
7 rows in set (0.00 sec)

mysql> SHOW PROFILE FOR QUERY 1;
+-----+-----+
| Status | Duration |
+-----+-----+
| query end | 0.000107 |
| freeing items | 0.000008 |
| logging slow query | 0.000015 |
| cleaning up | 0.000006 |
+-----+-----+
4 rows in set (0.00 sec)

mysql> SHOW PROFILE CPU FOR QUERY 2;
+-----+-----+-----+-----+
| Status | Duration | CPU_user | CPU_system |
+-----+-----+-----+-----+
| checking permissions | 0.000040 | 0.000038 | 0.000002 |
| creating table | 0.000056 | 0.000028 | 0.000028 |
| After create | 0.011363 | 0.000217 | 0.001571 |
| query end | 0.000375 | 0.000013 | 0.000028 |
| freeing items | 0.000089 | 0.000010 | 0.000014 |
| logging slow query | 0.000019 | 0.000009 | 0.000010 |
| cleaning up | 0.000005 | 0.000003 | 0.000002 |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```



Note

Profiling is only partially functional on some architectures. For values that depend on the `getrusage()` system call, `NULL` is returned on systems such as Windows that do not support the call. In addition, profiling is per process and not per thread. This means that activity on threads within the server other than your own may affect the timing information that you see.

You can also get profiling information from the `PROFILING` table in `INFORMATION_SCHEMA`. See [Section 20.15, “The `INFORMATION_SCHEMA PROFILING` Table”](#). For example, the following queries produce the same result:

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

13.7.5.32. SHOW PROFILES Syntax

```
SHOW PROFILES
```

The `SHOW PROFILES` statement, together with `SHOW PROFILE`, displays profiling information that indicates resource usage for statements executed during the course of the current session. For more information, see [Section 13.7.5.31, “SHOW PROFILE Syntax”](#).

13.7.5.33. SHOW RELAYLOG EVENTS Syntax

```
SHOW RELAYLOG EVENTS
  [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the relay log of a replication slave. If you do not specify `'log_name'`, the first relay log is displayed. This statement has no effect on the master.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See [Section 13.2.9, “SELECT Syntax”](#).



Note

Issuing a `SHOW RELAYLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the relay log (including all statements modifying data that have been received by the slave).



Note

Some events relating to the setting of user and system variables are not included in the output from `SHOW RELAYLOG EVENTS`. To get complete coverage of events within a relay log, use `mysqlbinlog`.

13.7.5.34. SHOW SLAVE HOSTS Syntax

```
SHOW SLAVE HOSTS
```

Displays a list of replication slaves currently registered with the master. (Before MySQL 5.5.3, only slaves started with the `--report-host=host_name` [\[1880\]](#) option are visible in this list.)

The list is displayed on any server (not just the master server). The output looks like this:

```
mysql> SHOW SLAVE HOSTS;
+-----+-----+-----+-----+
| Server_id | Host      | Port | Master_id |
+-----+-----+-----+-----+
| 192168010 | iconnect2 | 3306 | 192168011 |
| 1921680101 | athena    | 3306 | 192168011 |
+-----+-----+-----+-----+
```

- `Server_id`: The unique server ID of the slave server, as configured in the server's option file, or on the command line with `--server-id=value` [\[1858\]](#).

- **Host:** The host name of the slave server, as configured in the server's option file, or on the command line with `--report-host=host_name` [1880]. Note that this can differ from the machine name as configured in the operating system.
- **Port:** The port the slave server is listening on.

In MySQL 5.5.23 and later, a zero in this column means that the slave port (`--report-port` [1881]) was not set. Prior to MySQL 5.5.23, 3306 was used as the default in such cases (Bug #13333431).

- **Master_id:** The unique server ID of the master server that the slave server is replicating from.

Some MySQL versions report another variable, `Rpl_recovery_rank` [1887]. This variable was never used, and was removed in MySQL 5.5.3. (Bug #13963)

13.7.5.35. SHOW SLAVE STATUS Syntax

```
SHOW SLAVE STATUS
```

This statement provides status information on essential parameters of the slave threads. It requires either the `SUPER` [697] or `REPLICATION CLIENT` [697] privilege.

If you issue this statement using the `mysql` client, you can use a `\G` statement terminator rather than a semicolon to obtain a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: localhost
      Master_User: root
      Master_Port: 3306
      Connect_Retry: 3
      Master_Log_File: gbichot-bin.005
      Read_Master_Log_Pos: 79
      Relay_Log_File: gbichot-relay-bin.005
      Relay_Log_Pos: 548
      Relay_Master_Log_File: gbichot-bin.005
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
      Replicate_Wild_Ignore_Table:
      Last_Errno: 0
      Last_Error:
      Skip_Counter: 0
      Exec_Master_Log_Pos: 79
      Relay_Log_Space: 552
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
      Master_SSL_Allowed: No
      Master_SSL_CA_File:
      Master_SSL_CA_Path:
      Master_SSL_Cert:
      Master_SSL_Cipher:
      Master_SSL_Key:
      Seconds_Behind_Master: 8
      Master_SSL_Verify_Server_Cert: No
      Last_IO_Errno: 0
      Last_IO_Error:
      Last_SQL_Errno: 0
      Last_SQL_Error:
      Replicate_Ignore_Server_Ids: 0
```

```
Master_Server_Id: 1
```

The following list describes the fields returned by `SHOW SLAVE STATUS`. For additional information about interpreting their meanings, see [Section 8.12.5.6, “Replication Slave I/O Thread States”](#).

- `Slave_IO_State`

A copy of the `State` field of the `SHOW PROCESSLIST` output for the slave I/O thread. This tells you what the thread is doing: trying to connect to the master, waiting for events from the master, reconnecting to the master, and so on. For a listing of possible states, see [Section 8.12.5.6, “Replication Slave I/O Thread States”](#).

- `Master_Host`

The master host that the slave is connected to.

- `Master_User`

The user name of the account used to connect to the master.

- `Master_Port`

The port used to connect to the master.

- `Connect_Retry`

The number of seconds between connect retries (default 60). This can be set with the `CHANGE MASTER TO` statement.

- `Master_Log_File`

The name of the master binary log file from which the I/O thread is currently reading.

- `Read_Master_Log_Pos`

The position in the current master binary log file up to which the I/O thread has read.

- `Relay_Log_File`

The name of the relay log file from which the SQL thread is currently reading and executing.

- `Relay_Log_Pos`

The position in the current relay log file up to which the SQL thread has read and executed.

- `Relay_Master_Log_File`

The name of the master binary log file containing the most recent event executed by the SQL thread.

- `Slave_IO_Running`

Whether the I/O thread is started and has connected successfully to the master. Internally, the state of this thread is represented by one of the following three values:

- **`MYSQL_SLAVE_NOT_RUN`**. The slave I/O thread is not running. For this state, `Slave_IO_Running` is `No`.
- **`MYSQL_SLAVE_RUN_NOT_CONNECT`**. The slave I/O thread is running, but is not connected to a replication master. For this state, `Slave_IO_Running` depends on the server version as shown in the following table.

MySQL Version	Slave_IO_Running
4.1 (4.1.13 and earlier); 5.0 (5.0.11 and earlier)	Yes
4.1 (4.1.14 and later); 5.0 (5.0.12 and later)	No
5.1 (5.1.45 and earlier); 5.4	No
5.1 (5.1.46 and later); 5.5	Connecting

- **MYSQL_SLAVE_RUN_CONNECT.** The slave I/O thread is running, and is connected to a replication master. For this state, `Slave_IO_Running` is `Yes`.

The value of the `Slave_running` [608] system status variable corresponds with this value.

- `Slave_SQL_Running`

Whether the SQL thread is started.

- `Replicate_Do_DB`, `Replicate_Ignore_DB`

The lists of databases that were specified with the `--replicate-do-db` [1875] and `--replicate-ignore-db` [1877] options, if any.

- `Replicate_Do_Table`, `Replicate_Ignore_Table`, `Replicate_Wild_Do_Table`, `Replicate_Wild_Ignore_Table`

The lists of tables that were specified with the `--replicate-do-table` [1878], `--replicate-ignore-table` [1878], `--replicate-wild-do-table` [1879], and `--replicate-wild-ignore-table` [1880] options, if any.

- `Last_Errno`, `Last_Error`

These columns are aliases for `Last_SQL_Errno` and `Last_SQL_Error`.

Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.



Note

When the slave SQL thread receives an error, it reports the error first, then stops the SQL thread. This means that there is a small window of time during which `SHOW SLAVE STATUS` shows a nonzero value for `Last_SQL_Errno` even though `Slave_SQL_Running` still displays `Yes`.

- `Skip_Counter`

The current value of the `sql_slave_skip_counter` [1890] system variable. See [Section 13.4.2.4, "SET GLOBAL sql_slave_skip_counter Syntax"](#).

- `Exec_Master_Log_Pos`

The position in the current master binary log file to which the SQL thread has read and executed, marking the start of the next transaction or event to be processed. You can use this value with the `CHANGE MASTER TO` statement's `MASTER_LOG_POS` option when starting a new slave from an existing slave, so that the new slave reads from this point. The coordinates given by (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) in the master's binary log correspond to the coordinates given by (`Relay_Log_File`, `Relay_Log_Pos`) in the relay log.

- `Relay_Log_Space`

The total combined size of all existing relay log files.

- `Until_Condition`, `Until_Log_File`, `Until_Log_Pos`

The values specified in the `UNTIL` clause of the `START SLAVE` statement.

`Until_Condition` has these values:

- `None` if no `UNTIL` clause was specified
- `Master` if the slave is reading until a given position in the master's binary log
- `Relay` if the slave is reading until a given position in its relay log

`Until_Log_File` and `Until_Log_Pos` indicate the log file name and position that define the coordinates at which the SQL thread stops executing.

- `Master_SSL_Allowed`, `Master_SSL_CA_File`, `Master_SSL_CA_Path`, `Master_SSL_Cert`, `Master_SSL_Cipher`, `Master_SSL_Key`, `Master_SSL_Verify_Server_Cert`

These fields show the SSL parameters used by the slave to connect to the master, if any.

`Master_SSL_Allowed` has these values:

- `Yes` if an SSL connection to the master is permitted
- `No` if an SSL connection to the master is not permitted
- `Ignored` if an SSL connection is permitted but the slave server does not have SSL support enabled

The values of the other SSL-related fields correspond to the values of the `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_CIPHER`, `MASTER_SSL_KEY`, and `MASTER_SSL_VERIFY_SERVER_CERT` options to the `CHANGE MASTER TO` statement. See [Section 13.4.2.1, “CHANGE MASTER TO Syntax”](#).

- `Seconds_Behind_Master`

This field is an indication of how “late” the slave is:

- When the slave SQL thread is actively processing updates, this field is the number of seconds that have elapsed since the timestamp of the most recent event on the master executed by that thread.
- When the SQL thread has caught up to the slave I/O thread and is idle waiting for more events from the I/O thread, this field is zero.

In essence, this field measures the time difference in seconds between the slave SQL thread and the slave I/O thread.

If the network connection between master and slave is fast, the slave I/O thread is very close to the master, so this field is a good approximation of how late the slave SQL thread is compared to the master. If the network is slow, this is *not* a good approximation; the slave SQL thread may quite often be caught up with the slow-reading slave I/O thread, so `Seconds_Behind_Master` often shows a value of 0, even if the I/O thread is late compared to the master. In other words, *this column is useful only for fast networks*.

This time difference computation works even if the master and slave do not have identical clock times, provided that the difference, computed when the slave I/O thread starts, remains constant from then

on. Any changes—including NTP updates—can lead to clock skews that can make calculation of `Seconds_Behind_Master` less reliable.

This field is `NULL` (undefined or unknown) if the slave SQL thread is not running, or if the slave I/O thread is not running or is not connected to the master. For example, if the slave I/O thread is running but is not connected to the master and is sleeping for the number of seconds given by the `CHANGE MASTER TO` statement or `--master-connect-retry` [1884] option (default 60) before reconnecting, the value is `NULL`. This is because the slave cannot know what the master is doing, and so cannot say reliably how late it is.

The value of `Seconds_Behind_Master` is based on the timestamps stored in events, which are preserved through replication. This means that if a master M1 is itself a slave of M0, any event from M1's binary log that originates from M0's binary log has M0's timestamp for that event. This enables MySQL to replicate `TIMESTAMP` successfully. However, the problem for `Seconds_Behind_Master` is that if M1 also receives direct updates from clients, the `Seconds_Behind_Master` value randomly fluctuates because sometimes the last event from M1 originates from M0 and sometimes is the result of a direct update on M1.

- `Last_IO_Errno`, `Last_IO_Error`

The error number and error message of the last error that caused the I/O thread to stop. An error number of 0 and message of the empty string mean “no error.” If the `Last_IO_Error` value is not empty, the error values also appear in the slave's error log.

Prior to MySQL 5.5, `Last_IO_Error` and `Last_IO_Errno` were not set in the event that replication failed due to exceeding `max_allowed_packet` [519] (Bug #42914).

Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

- `Last_SQL_Errno`, `Last_SQL_Error`

The error number and error message of the last error that caused the SQL thread to stop. An error number of 0 and message of the empty string mean “no error.” If the `Last_SQL_Error` value is not empty, the error values also appear in the slave's error log.

Example:

```
Last_SQL_Errno: 1051
Last_SQL_Error: error 'Unknown table 'z'' on query 'drop table z'
```

The message indicates that the table `z` existed on the master and was dropped there, but it did not exist on the slave, so `DROP TABLE` failed on the slave. (This might occur, for example, if you forget to copy the table to the slave when setting up replication.)

Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

- `Replicate_Ignore_Server_Ids`

Beginning with MySQL 5.5, you can tell a slave to ignore events from 0 or more masters using the `IGNORE_SERVER_IDS` option in a `CHANGE MASTER TO` statement. This is normally of interest only when using a circular or other multi-master replication setup.

The message shown for `Replicate_Ignore_Server_Ids` consists of a space-delimited list of one or more numbers, the first value indicating the number of servers to be ignored; if not 0 (the default), this server-count value is followed by the actual server IDs. For example, if a `CHANGE MASTER TO` statement containing the `IGNORE_SERVER_IDS = (2,6,9)` option has been issued to tell a slave to ignore masters having the server ID 2, 6, or 9, that information appears as shown here:

```
Replicate_Ignore_Server_Ids: 3 2 6 9
```

- `Master_Server_Id`

The `server_id` [551] value from the master.

13.7.5.36. SHOW STATUS Syntax

```
SHOW [GLOBAL | SESSION] STATUS
     [LIKE 'pattern' | WHERE expr]
```

`SHOW STATUS` provides server status information. This information also can be obtained using the `mysqladmin extended-status` command. The `LIKE` [1111] clause, if present, indicates which variable names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#). This statement does not require any privilege. It requires only the ability to connect to the server.

Partial output is shown here. The list of names and values may be different for your server. The meaning of each variable is given in [Section 5.1.6, “Server Status Variables”](#).

```
mysql> SHOW STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Aborted_clients | 0 |
| Aborted_connects | 0 |
| Bytes_received | 155372598 |
| Bytes_sent | 1176560426 |
| Connections | 30023 |
| Created_tmp_disk_tables | 0 |
| Created_tmp_tables | 8340 |
| Created_tmp_files | 60 |
| ... |
| Open_tables | 1 |
| Open_files | 2 |
| Open_streams | 0 |
| Opened_tables | 44600 |
| Questions | 2026873 |
| ... |
| Table_locks_immediate | 1920382 |
| Table_locks_waited | 0 |
| Threads_cached | 0 |
| Threads_created | 30022 |
| Threads_connected | 1 |
| Threads_running | 1 |
| Uptime | 80380 |
+-----+-----+
```

With a `LIKE` [1111] clause, the statement displays only rows for those variables with names that match the pattern:

```
mysql> SHOW STATUS LIKE 'Key%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Key_blocks_used | 14955 |
| Key_read_requests | 96854827 |
| Key_reads | 162040 |
| Key_write_requests | 7589728 |
| Key_writes | 3813196 |
+-----+-----+
```

With the `GLOBAL` modifier, `SHOW STATUS` displays the status values for all connections to MySQL. With `SESSION`, it displays the status values for the current connection. If no modifier is present, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`.

Some status variables have only a global value. For these, you get the same value for both `GLOBAL` and `SESSION`. The scope for each status variable is listed at [Section 5.1.6, “Server Status Variables”](#).

Each invocation of the `SHOW STATUS` statement uses an internal temporary table and increments the global `Created_tmp_tables [598]` value.

13.7.5.37. SHOW TABLE STATUS Syntax

```
SHOW TABLE STATUS [{FROM | IN} db_name]
                    [LIKE 'pattern' | WHERE expr]
```

`SHOW TABLE STATUS` works like `SHOW TABLES`, but provides a lot of information about each non-`TEMPORARY` table. You can also get this list using the `mysqlshow --status db_name` command. The `LIKE [1111]` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#).

This statement also displays information about views.

`SHOW TABLE STATUS` returns the following fields:

- `Name`
The name of the table.
- `Engine`
The storage engine for the table. See [Chapter 14, Storage Engines](#).
- `Version`
The version number of the table's `.frm` file.
- `Row_format`
The row-storage format (`Fixed`, `Dynamic`, `Compressed`, `Redundant`, `Compact`). For `MyISAM` tables, (`Dynamic` corresponds to what `myisamchk -dvv` reports as `Packed`). The format of `InnoDB` tables is reported as `Redundant` or `Compact`. For the `Barracuda` file format of the `InnoDB Plugin`, the format may be `Compressed` or `Dynamic`.
- `Rows`
The number of rows. Some storage engines, such as `MyISAM`, store the exact count. For other storage engines, such as `InnoDB`, this value is an approximation, and may vary from the actual value by as much as 40 to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

The `Rows` value is `NULL` for tables in the `INFORMATION_SCHEMA` database.
- `Avg_row_length`
The average row length.
- `Data_length`

The length of the data file.

- `Max_data_length`

The maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

- `Index_length`

The length of the index file.

- `Data_free`

The number of allocated but unused bytes.

This information is also shown for `InnoDB` tables (previously, it was in the `Comment` value). `InnoDB` tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of completely free 1MB extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

For partitioned tables, this value is only an estimate and may not be absolutely correct. A more accurate method of obtaining this information in such cases is to query the `INFORMATION_SCHEMA.PARTITIONS` table, as shown in this example:

```
SELECT      SUM(DATA_FREE)
FROM        INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'mydb'
AND TABLE_NAME   = 'mytable';
```

For more information, see [Section 20.12, “The INFORMATION_SCHEMA PARTITIONS Table”](#).

- `Auto_increment`

The next `AUTO_INCREMENT` value.

- `Create_time`

When the table was created.

- `Update_time`

When the data file was last updated. For some storage engines, this value is `NULL`. For example, `InnoDB` stores multiple tables in its `system tablespace` and the data file timestamp does not apply. Even with `file-per-table` mode with each `InnoDB` table in a separate `.ibd` file, `change buffering` can delay the write to the data file, so the file modification time is different from the time of the last insert, update, or delete. For `MyISAM`, the data file timestamp is used; however, on Windows the timestamp is not updated by updates so the value is inaccurate.

- `Check_time`

When the table was last checked. Not all storage engines update this time, in which case the value is always `NULL`.

- `Collation`

The table's character set and collation.

- `Checksum`

The live checksum value (if any).

- `Create_options`

Extra options used with `CREATE TABLE`. The original options supplied when `CREATE TABLE` is called are retained and the options reported here may differ from the active table settings and options.

- `Comment`

The comment used when creating the table (or information as to why MySQL could not access the table information).

For `MEMORY` tables, the `Data_length`, `Max_data_length`, and `Index_length` values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

For `NDBCLUSTER` tables, the output of this statement shows appropriate values for the `Avg_row_length` and `Data_length` columns, with the exception that `BLOB` columns are not taken into account

For views, all the fields displayed by `SHOW TABLE STATUS` are `NULL` except that `Name` indicates the view name and `Comment` says `view`.

13.7.5.38. SHOW TABLES Syntax

```
SHOW [FULL] TABLES [{FROM | IN} db_name]
                    [LIKE 'pattern' | WHERE expr]
```

`SHOW TABLES` lists the non-`TEMPORARY` tables in a given database. You can also get this list using the `mysqlshow db_name` command. The `LIKE [1111]` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#).

This statement also lists any views in the database. The `FULL` modifier is supported such that `SHOW FULL TABLES` displays a second output column. Values for the second column are `BASE TABLE` for a table and `VIEW` for a view.

If you have no privileges for a base table or view, it does not show up in the output from `SHOW TABLES` or `mysqlshow db_name`.

13.7.5.39. SHOW TRIGGERS Syntax

```
SHOW TRIGGERS [{FROM | IN} db_name]
              [LIKE 'pattern' | WHERE expr]
```

`SHOW TRIGGERS` lists the triggers currently defined for tables in a database (the default database unless a `FROM` clause is given). This statement returns results only for databases and tables for which you have the `TRIGGER [697]` privilege. The `LIKE [1111]` clause, if present, indicates which table names to match and causes the statement to display triggers for those tables. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#).

For the trigger `ins_sum` as defined in [Section 19.3, “Using Triggers”](#), the output of this statement is as shown here:

```
mysql> SHOW TRIGGERS LIKE 'acc%\G
***** 1. row *****
      Trigger: ins_sum
      Event: INSERT
      Table: account
      Statement: SET @sum = @sum + NEW.amount
```

```

Timing: BEFORE
Created: NULL
sql_mode:
Definer: myname@localhost
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci

```

`character_set_client` [486] is the session value of the `character_set_client` [486] system variable when the trigger was created. `collation_connection` [489] is the session value of the `collation_connection` [489] system variable when the trigger was created. `Database Collation` is the collation of the database with which the trigger is associated.



Note

When using a `LIKE` [1111] clause with `SHOW TRIGGERS`, the expression to be matched (*expr*) is compared with the name of the table on which the trigger is declared, and not with the name of the trigger:

```

mysql> SHOW TRIGGERS LIKE 'ins%';
Empty set (0.01 sec)

```

A brief explanation of the columns in the output of this statement is shown here:

- `Trigger`
The name of the trigger.
- `Event`
The event that causes trigger activation: one of 'INSERT', 'UPDATE', or 'DELETE'.
- `Table`
The table for which the trigger is defined.
- `Statement`
The statement to be executed when the trigger is activated. This is the same as the text shown in the `ACTION_STATEMENT` column of `INFORMATION_SCHEMA.TRIGGERS`.
- `Timing`
One of the two values 'BEFORE' or 'AFTER'.
- `Created`
Currently, the value of this column is always `NULL`.
- `sql_mode` [556]
The SQL mode in effect when the trigger executes.
- `Definer`
The account that created the trigger.

See also [Section 20.25, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

13.7.5.40. SHOW VARIABLES Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES
```

```
[LIKE 'pattern' | WHERE expr]
```

`SHOW VARIABLES` shows the values of MySQL system variables. This information also can be obtained using the `mysqladmin variables` command. The `LIKE [1111]` clause, if present, indicates which variable names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 20.31, “Extensions to SHOW Statements”](#). This statement does not require any privilege. It requires only the ability to connect to the server.

With the `GLOBAL` modifier, `SHOW VARIABLES` displays the values that are used for new connections to MySQL. As of MySQL 5.5.3, if a variable has no global value, no value is displayed. Before 5.5.3, the session value is displayed. With `SESSION`, `SHOW VARIABLES` displays the values that are in effect for the current connection. If no modifier is present, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`.

`SHOW VARIABLES` is subject to a version-dependent display-width limit. For variables with very long values that are not completely displayed, use `SELECT` as a workaround. For example:

```
SELECT @@GLOBAL.innodb_data_file_path;
```

If the default system variable values are unsuitable, you can set them using command options when `mysqld` starts, and most can be changed at runtime with the `SET` statement. See [Section 5.1.5, “Using System Variables”](#), and [Section 13.7.4, “SET Syntax”](#).

Partial output is shown here. The list of names and values may be different for your server. [Section 5.1.4, “Server System Variables”](#), describes the meaning of each variable, and [Section 8.11.2, “Tuning Server Parameters”](#), provides information about tuning them.

```
mysql> SHOW VARIABLES;
```

Variable_name	Value
auto_increment_increment	1
auto_increment_offset	1
autocommit	ON
automatic_sp_privileges	ON
back_log	50
basedir	/home/jon/bin/mysql-5.5
big_tables	OFF
binlog_cache_size	32768
binlog_direct_non_transactional_updates	OFF
binlog_format	STATEMENT
binlog_stmt_cache_size	32768
bulk_insert_buffer_size	8388608
...	
max_allowed_packet	1048576
max_binlog_cache_size	18446744073709547520
max_binlog_size	1073741824
max_binlog_stmt_cache_size	18446744073709547520
max_connect_errors	10
max_connections	151
max_delayed_threads	20
max_error_count	64
max_heap_table_size	16777216
max_insert_delayed_threads	20
max_join_size	18446744073709551615
...	
thread_handling	one-thread-per-connection
thread_stack	262144
time_format	%H:%i:%s
time_zone	SYSTEM
timed_mutexes	OFF
timestamp	1316689732
tmp_table_size	16777216

tmpdir	/tmp
transaction_alloc_block_size	8192
transaction_prealloc_size	4096
tx_isolation	REPEATABLE-READ
unique_checks	ON
updatable_views_with_limit	YES
version	5.5.17-log
version_comment	Source distribution
version_compile_machine	x86_64
version_compile_os	Linux
wait_timeout	28800
warning_count	0

With a [LIKE \[1111\]](#) clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a [LIKE \[1111\]](#) clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the “%” wildcard character in a [LIKE \[1111\]](#) clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because “_” is a wildcard that matches any single character, you should escape it as “_” to match it literally. In practice, this is rarely necessary.

13.7.5.41. SHOW WARNINGS Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

[SHOW WARNINGS](#) shows information about the conditions (errors, warnings, and notes) that resulted from the last statement in the current session that generated messages. It shows nothing if the last statement used a table and generated no messages. (That is, a statement that uses a table but generates no messages clears the message list.) Statements that do not use tables and do not generate messages have no effect on the message list.

Warnings are generated for DML statements such as [INSERT](#), [UPDATE](#), and [LOAD DATA INFILE](#) as well as DDL statements such as [CREATE TABLE](#) and [ALTER TABLE](#).

[SHOW WARNINGS](#) is also used following [EXPLAIN EXTENDED](#), to display the extra information generated by [EXPLAIN](#) when the [EXTENDED](#) keyword is used. See [Section 8.8.3, “EXPLAIN EXTENDED Output Format”](#).

The [LIMIT](#) clause has the same syntax as for the [SELECT](#) statement. See [Section 13.2.9, “SELECT Syntax”](#).

A related statement, [SHOW ERRORS](#), shows only the error conditions (it excludes warnings and notes). See [Section 13.7.5.18, “SHOW ERRORS Syntax”](#).

The [SHOW COUNT\(*\) WARNINGS](#) statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the [warning_count \[573\]](#) system variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

Here is a simple example that shows a syntax warning for [CREATE TABLE](#) and conversion warnings for [INSERT](#):

```

mysql> CREATE TABLE t1
  > (a TINYINT NOT NULL, b CHAR(4))
  > TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
        'ENGINE=storage_engine' instead
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'),
  -> (NULL,'test'), (300,'Open Source');
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 4

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 1
***** 2. row *****
Level: Warning
Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row 2
***** 3. row *****
Level: Warning
Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
***** 4. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)

```

The `max_error_count` [520] system variable controls the maximum number of error, warning, and note messages for which the server stores information, and thus the number of messages that `SHOW WARNINGS` displays. By default, `max_error_count` [520] is 64. To change the number of messages the server can store, change the value of `max_error_count` [520].

The value of `warning_count` [573] is not limited by `max_error_count` [520] if the number of messages generated exceeds `max_error_count` [520].

In the following example, the `ALTER TABLE` statement produces three warning messages (as shown by the value of `warning_count` [573]), but only one is stored because `max_error_count` [520] has been set to 1:

```

mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_error_count | 64   |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET max_error_count=1;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 3

mysql> SELECT @@warning_count;
+-----+

```

```

| @@warning_count |
+-----+
|          3 |
+-----+
1 row in set (0.01 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)

```

To disable warnings, set `max_error_count` [520] to 0. In this case, `warning_count` [573] still indicates how many warnings have occurred, but none of the messages are stored.

The following `DROP TABLE` statement results in a note:

```

mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.01 sec)
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note | 1051 | Unknown table 'no_such_table' |
+-----+-----+-----+

```

If the `sql_notes` [556] system variable is set to 0, notes do not increment `warning_count` [573] and the server does not record them.

The MySQL server sends back a count indicating the total number of errors, warnings, and notes resulting from the last statement. From the C API, this value can be obtained by calling `mysql_warning_count()`. See Section 22.8.3.72, “`mysql_warning_count()`”.

13.7.6. Other Administrative Statements

13.7.6.1. BINLOG Syntax

```
BINLOG 'str'
```

`BINLOG` is an internal-use statement. It is generated by the `mysqlbinlog` program as the printable representation of certain events in binary log files. (See Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”.) The `'str'` value is a base 64-encoded string that the server decodes to determine the data change indicated by the corresponding event. This statement requires the `SUPER` [697] privilege.

13.7.6.2. CACHE INDEX Syntax

```

CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  [PARTITION (partition_list | ALL)]
  IN key_cache_name

tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]

partition_list:
  partition_name[, partition_name][, ...]

```

The `CACHE INDEX` statement assigns table indexes to a specific key cache. It is used only for `MyISAM` tables. After the indexes have been assigned, they can be preloaded into the cache if desired with `LOAD INDEX INTO CACHE`.

The following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+-----+-----+-----+-----+
| Table | Op           | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+-----+-----+-----+-----+
```

The syntax of `CACHE INDEX` enables you to specify that only particular indexes from a table should be assigned to the cache. The current implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters can be accessed as members of a structured system variable. See [Section 5.1.5.1, "Structured System Variables"](#).

A key cache must exist before you can assign indexes to it:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it become assigned to the default key cache again.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

In MySQL 5.5, this statement is also supported for partitioned `MyISAM` tables. You can assign one or more indexes for one, several, or all partitions to a given key cache. For example, you can do the following:

```
CREATE TABLE pt (c1 INT, c2 VARCHAR(50), INDEX i(c1))
PARTITION BY HASH(c1)
PARTITIONS 4;

SET GLOBAL kc_fast.key_buffer_size = 128 * 1024;
SET GLOBAL kc_slow.key_buffer_size = 128 * 1024;

CACHE INDEX pt PARTITION (p0) IN kc_fast;
CACHE INDEX pt PARTITION (p1, p3) IN kc_slow;
```

The previous set of statements performs the following actions:

- Creates a partitioned table with 4 partitions; these partitions are automatically named `p0`, ..., `p3`; this table has an index named `i` on column `c1`.
- Creates 2 key caches named `kc_fast` and `kc_slow`
- Assigns the index for partition `p0` to the `kc_fast` key cache and the index for partitions `p1` and `p3` to the `kc_slow` key cache; the index for the remaining partition (`p2`) uses the server's default key cache.

If you wish instead to assign the indexes for all partitions in table `pt` to a single key cache named `kc_all`, you can use either one of the following 2 statements:

```
CACHE INDEX pt PARTITION (ALL) IN kc_all;

CACHE INDEX pt IN kc_all;
```

The two statements just shown are equivalent, and issuing either one of them has exactly the same effect. In other words, if you wish to assign indexes for all partitions of a partitioned table to the same key cache, then the `PARTITION (ALL)` clause is optional.

When assigning indexes for multiple partitions to a key cache, the partitions do not have to be contiguous, and you are not required to list their names in any particular order. Indexes for any partitions that are not explicitly assigned to a key cache automatically use the server's default key cache.

As of MySQL 5.5, index preloading is also supported for partitioned `MyISAM` tables. For more information, see [Section 13.7.6.5, “LOAD INDEX INTO CACHE Syntax”](#).

13.7.6.3. FLUSH Syntax

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL]
      flush_option [, flush_option] ...
```

The `FLUSH` statement has several variant forms that clear or reload various internal caches, flush tables, or acquire locks. To execute `FLUSH`, you must have the `RELOAD [696]` privilege. Specific flush options might require additional privileges, as described later.

By default, the server writes `FLUSH` statements to the binary log so that they replicate to replication slaves. To suppress logging, use the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.



Note

`FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK` (with or without a table list) are not written to the binary log in any case because they would cause problems if replicated to a slave.

The `FLUSH` statement causes an implicit commit. See [Section 13.3.3, “Statements That Cause an Implicit Commit”](#).

The `RESET` statement is similar to `FLUSH`. See [Section 13.7.6.6, “RESET Syntax”](#), for information about using the `RESET` statement with replication.

`flush_option` can be any of the following items.

- `DES_KEY_FILE`

Reloads the DES keys from the file that was specified with the `--des-key-file [442]` option at server startup time.

- `HOSTS`

Empties the host cache. You should flush the host cache if some of your hosts change IP address or if the error message `Host 'host_name' is blocked` occurs. (See [Section C.5.2.6, “Host 'host_name' is blocked”](#).) When more than `max_connect_errors [519]` errors occur successively for a given host while connecting to the MySQL server, MySQL assumes that something is wrong and blocks the host from further connection requests. Flushing the host cache enables further connection attempts from the host. The default value of `max_connect_errors [519]` is 10. To avoid this error message, start the server with `max_connect_errors [519]` set to a large value.

- `[log_type] LOGS`

With no `log_type` option, `FLUSH LOGS` closes and reopens all log files. If binary logging is enabled, the sequence number of the binary log file is incremented by one relative to the previous file. On Unix, this is the same thing as sending a `SIGHUP` signal to the `mysqld` server (except on some Mac OS X 10.3 versions where `mysqld` ignores `SIGHUP` and `SIGQUIT`).

Prior to MySQL 5.5.7, if you flush the logs using `FLUSH LOGS` and `mysqld` is writing the error log to a file (for example, if it was started with the `--log-error [448]` option), log file renaming may occur, as described in [Section 5.2.2, “The Error Log”](#).

With a `log_type` option, only the specified log type is flushed. These `log_type` options are permitted:

- `BINARY` closes and reopens the binary log files.
- `ENGINE` closes and reopens any flushable logs for installed storage engines. Currently, this causes `InnoDB` to flush its logs to disk and perform a checkpoint.
- `ERROR` closes and reopens the error log file.
- `GENERAL` closes and reopens the general query log file.
- `RELAY` closes and reopens the relay log files.
- `SLOW` closes and reopens the slow query log file.

The `log_type` options were added in MySQL 5.5.3.

- `MASTER`

Deletes all binary logs, resets the binary log index file and creates a new binary log. `FLUSH MASTER` is deprecated in favor of `RESET MASTER`. `FLUSH MASTER` is still accepted in MySQL 5.5 for backward compatibility, but is removed in MySQL 5.6. See [Section 13.4.1.2, “RESET MASTER Syntax”](#).

- `PRIVILEGES`

Reloads the privileges from the grant tables in the `mysql` database. On Unix, this also occurs if the server receives a `SIGHUP` signal.

The server caches information in memory as a result of `GRANT`, `CREATE USER`, `CREATE SERVER`, and `INSTALL PLUGIN` statements. This memory is not released by the corresponding `REVOKE`, `DROP USER`, `DROP SERVER`, and `UNINSTALL PLUGIN` statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with `FLUSH PRIVILEGES`.

- `QUERY CACHE`

Defragment the query cache to better utilize its memory. `FLUSH QUERY CACHE` does not remove any queries from the cache, unlike `FLUSH TABLES` or `RESET QUERY CACHE`.

- `SLAVE`

Resets all replication slave parameters, including relay log files and replication position in the master's binary logs. `FLUSH SLAVE` is deprecated in favor of `RESET SLAVE`. `FLUSH SLAVE` is still accepted in MySQL 5.5 for backward compatibility, but is removed in MySQL 5.6. See [Section 13.4.2.3, “RESET SLAVE Syntax”](#).

- `STATUS`

This option adds the current thread's session status variable values to the global values and resets the session values to zero. Some global variables may be reset to zero as well. It also resets the counters for key caches (default and named) to zero and sets `Max_used_connections [604]` to the current number of open connections. This is something you should use only when debugging a query. See [Section 1.7, “How to Report Bugs or Problems”](#).

- [TABLES](#)

[FLUSH TABLES](#) flushes tables, and, depending on the variant used, acquires locks. The permitted syntax is discussed later in this section.

- [USER_RESOURCES](#)

Resets all per-hour user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. [FLUSH USER_RESOURCES](#) does not apply to the limit on maximum simultaneous connections. See [Section 6.3.4, “Setting Account Resource Limits”](#).

The [mysqladmin](#) utility provides a command-line interface to some flush operations, using commands such as [flush-hosts](#), [flush-logs](#), [flush-privileges](#), [flush-status](#), and [flush-tables](#). See [Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”](#).



Note

It is not possible to issue [FLUSH](#) statements within stored functions or triggers. However, you may use [FLUSH](#) in stored procedures, so long as these are not called from stored functions or triggers. See [Section E.1, “Restrictions on Stored Programs”](#).

[FLUSH TABLES](#) Syntax

[FLUSH TABLES](#) has several forms, described following. As of MySQL 5.5.3, if any variant of the [TABLES](#) option is used in a [FLUSH](#) statement, it must be the only option used. [FLUSH TABLE](#) is a synonym for [FLUSH TABLES](#), except that [TABLE](#) does not work with the [WITH READ LOCK](#) variants prior to MySQL 5.5.3.

- [FLUSH TABLES](#)

Closes all open tables, forces all tables in use to be closed, and flushes the query cache. [FLUSH TABLES](#) also removes all query results from the query cache, like the [RESET QUERY CACHE](#) statement.

As of MySQL 5.5.3, [FLUSH TABLES](#) is not permitted when there is an active [LOCK TABLES ... READ](#). To flush and lock tables, use [FLUSH TABLES tbl_name ... WITH READ LOCK](#) instead.

- [FLUSH TABLES tbl_name \[, tbl_name\] ...](#)

With a list of one or more comma-separated table names, this statement is like [FLUSH TABLES](#) with no names except that the server flushes only the named tables. No error occurs if a named table does not exist.

- [FLUSH TABLES WITH READ LOCK](#)

Closes all open tables and locks all tables for all databases with a global read lock. This is a very convenient way to get backups if you have a file system such as Veritas or ZFS that can take snapshots in time. Use [UNLOCK TABLES](#) to release the lock.

[FLUSH TABLES WITH READ LOCK](#) acquires a global read lock and not table locks, so it is not subject to the same behavior as [LOCK TABLES](#) and [UNLOCK TABLES](#) with respect to table locking and implicit commits:

- [UNLOCK TABLES](#) implicitly commits any active transaction only if any tables currently have been locked with [LOCK TABLES](#). The commit does not occur for [UNLOCK TABLES](#) following [FLUSH TABLES WITH READ LOCK](#) because the latter statement does not acquire table locks.

- Beginning a transaction causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

`FLUSH TABLES WITH READ LOCK` does not prevent the server from inserting rows into the log tables (see Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”).

- `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK`

This statement flushes and acquires read locks for the named tables. The statement first acquires exclusive metadata locks for the tables, so it waits for transactions that have those tables open to complete. Then the statement flushes the tables from the table cache, reopens the tables, acquires table locks (like `LOCK TABLES ... READ`), and downgrades the metadata locks from exclusive to shared. After the statement acquires locks and downgrades the metadata locks, other sessions can read but not modify the tables.

Because this statement acquires table locks, you must have the `LOCK TABLES [696]` privilege for each table, in addition to the `RELOAD [696]` privilege that is required to use any `FLUSH` statement.

This statement applies only to existing base tables. If a name refers to a base table, that table is used. If it refers to a `TEMPORARY` table, it is ignored. If a name applies to a view, an `ER_WRONG_OBJECT [4002]` error occurs. Otherwise, an `ER_NO_SUCH_TABLE [3989]` error occurs.

Use `UNLOCK TABLES` to release the locks, `LOCK TABLES` to release the locks and acquire other locks, or `START TRANSACTION` to release the locks and begin a new transaction.

This variant of `FLUSH` enables tables to be flushed and locked in a single operation. It provides a workaround for the restriction as of MySQL 5.5.3 that `FLUSH TABLES` is not permitted when there is an active `LOCK TABLES ... READ`.

This statement does not perform an implicit `UNLOCK TABLES`, so an error results if you use the statement while there is any active `LOCK TABLES` or use it a second time without first releasing the locks acquired.

If a flushed table was opened with `HANDLER`, the handler is implicitly flushed and loses its position.

This variant of `FLUSH` is available as of MySQL 5.5.3.

13.7.6.4. `KILL` Syntax

```
KILL [CONNECTION | QUERY] thread_id
```

Each connection to `mysqld` runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` statement and kill a thread with the `KILL thread_id` statement.

`KILL` permits an optional `CONNECTION` or `QUERY` modifier:

- `KILL CONNECTION` is the same as `KILL` with no modifier: It terminates the connection associated with the given *thread_id*.
- `KILL QUERY` terminates the statement that the connection is currently executing, but leaves the connection itself intact.

If you have the `PROCESS [696]` privilege, you can see all threads. If you have the `SUPER [697]` privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

You can also use the `mysqladmin processlist` and `mysqladmin kill` commands to examine and kill threads.



Note

You cannot use `KILL` with the Embedded MySQL Server library because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

When you use `KILL`, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die because the kill flag is checked only at specific intervals:

- In `SELECT`, `ORDER BY` and `GROUP BY` loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.
- During `ALTER TABLE`, the kill flag is checked before each block of rows are read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.
- During `UPDATE` or `DELETE` operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. Note that if you are not using transactions, the changes are not rolled back.
- `GET_LOCK()` [1199] aborts and returns `NULL`.
- An `INSERT DELAYED` thread quickly flushes (inserts) all rows it has in memory and then terminates.
- If the thread is in the table lock handler (state: `Locked`), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.



Warning

Killing a `REPAIR TABLE` or `OPTIMIZE TABLE` operation on a `MyISAM` table results in a table that is corrupted and unusable. Any reads or writes to such a table fail until you optimize or repair it again (without interruption).

13.7.6.5. `LOAD INDEX INTO CACHE` Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
  [PARTITION (partition_list | ALL)]
  [[INDEX|KEY] (index_name[, index_name] ...)]
  [IGNORE LEAVES]

partition_list:
  partition_name[, partition_name][, ...]
```

The `LOAD INDEX INTO CACHE` statement preloads a table index into the key cache to which it has been assigned by an explicit `CACHE INDEX` statement, or into the default key cache otherwise.

`LOAD INDEX INTO CACHE` is used only for `MyISAM` tables. In MySQL 5.5, it is also supported for partitioned `MyISAM` tables; in addition, indexes on partitioned tables can be preloaded for one, several, or all partitions.

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded.

`IGNORE LEAVES` is also supported for partitioned `MyISAM` tables.

The following statement preloads nodes (index blocks) of indexes for the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+-----+-----+-----+-----+
| Table   | Op           | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+-----+-----+-----+-----+
```

This statement preloads all index blocks from `t1`. It preloads only blocks for the nonleaf nodes from `t2`.

The syntax of `LOAD INDEX INTO CACHE` enables you to specify that only particular indexes from a table should be preloaded. The current implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

In MySQL 5.5, it is possible to preload indexes on specific partitions of partitioned `MyISAM` tables. For example, of the following 2 statements, the first preloads indexes for partition `p0` of a partitioned table `pt`, while the second preloads the indexes for partitions `p1` and `p3` of the same table:

```
LOAD INDEX INTO CACHE pt PARTITION (p0);
LOAD INDEX INTO CACHE pt PARTITION (p1, p3);
```

To preload the indexes for all partitions in table `pt`, you can use either one of the following 2 statements:

```
LOAD INDEX INTO CACHE pt PARTITION (ALL);
LOAD INDEX INTO CACHE pt;
```

The two statements just shown are equivalent, and issuing either one of them has exactly the same effect. In other words, if you wish to preload indexes for all partitions of a partitioned table, then the `PARTITION (ALL)` clause is optional.

When preloading indexes for multiple partitions, the partitions do not have to be contiguous, and you are not required to list their names in any particular order.

`LOAD INDEX INTO CACHE ... IGNORE LEAVES` fails unless all indexes in a table have the same block size. You can determine index block sizes for a table by using `myisamchk -dv` and checking the `Blocksize` column.

13.7.6.6. RESET Syntax

```
RESET reset_option [, reset_option] ...
```

The `RESET` statement is used to clear the state of various server operations. You must have the `RELOAD [696]` privilege to execute `RESET`.

`RESET` acts as a stronger version of the `FLUSH` statement. See [Section 13.7.6.3, "FLUSH Syntax"](#).

The `RESET` statement causes an implicit commit. See [Section 13.3.3, "Statements That Cause an Implicit Commit"](#).

`reset_option` can be any of the following:

- `MASTER`

Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

- `QUERY CACHE`

Removes all query results from the query cache.

- `SLAVE`

Makes the slave forget its replication position in the master binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one.

13.8. MySQL Utility Statements

13.8.1. `DESCRIBE` Syntax

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

`DESCRIBE` provides information about the columns in a table. It is a shortcut for `SHOW COLUMNS FROM`. These statements also display information for views. (See [Section 13.7.5.6, “SHOW COLUMNS Syntax”](#).)

`col_name` can be a column name, or a string containing the SQL “%” and “_” wildcard characters to obtain output only for the columns with names matching the string. There is no need to enclose the string within quotation marks unless it contains spaces or other special characters.

```
mysql> DESCRIBE City;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| Id         | int(11)   | NO   | PRI | NULL    | auto_increment |
| Name      | char(35)  | NO   |     |         |              |
| Country   | char(3)   | NO   |     |         |              |
| District  | char(20)  | YES  | MUL |         |              |
| Population | int(11)   | NO   |     | 0       |              |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

The description for `SHOW COLUMNS` provides more information about the output columns (see [Section 13.7.5.6, “SHOW COLUMNS Syntax”](#)).

If the data types differ from what you expect them to be based on a `CREATE TABLE` statement, note that MySQL sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in [Section 13.1.17.2, “Silent Column Specification Changes”](#).

The `DESCRIBE` statement is provided for compatibility with Oracle.

The `SHOW CREATE TABLE`, `SHOW TABLE STATUS`, and `SHOW INDEX` statements also provide information about tables. See [Section 13.7.5, “SHOW Syntax”](#).

13.8.2. `EXPLAIN` Syntax

```
EXPLAIN [explain_type] SELECT select_options
```

```
explain_type:
  EXTENDED
  | PARTITIONS
```

Or:

```
EXPLAIN tbl_name
```

The `EXPLAIN` statement can be used either as a way to obtain information about how MySQL executes a statement, or as a synonym for `DESCRIBE`:

- When you precede a `SELECT` statement with the keyword `EXPLAIN`, MySQL displays information from the optimizer about the query execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. `EXPLAIN EXTENDED` can be used to obtain additional information.

For information about using `EXPLAIN` and `EXPLAIN EXTENDED` to obtain query execution plan information, see [Section 8.8.1, “Optimizing Queries with EXPLAIN”](#).

- `EXPLAIN PARTITIONS` is useful only when examining queries involving partitioned tables. For details, see [Section 18.3.4, “Obtaining Information About Partitions”](#).
- `EXPLAIN tbl_name` is synonymous with `DESCRIBE tbl_name` or `SHOW COLUMNS FROM tbl_name`. For information about `DESCRIBE` and `SHOW COLUMNS`, see [Section 13.8.1, “DESCRIBE Syntax”](#), and [Section 13.7.5.6, “SHOW COLUMNS Syntax”](#).

13.8.3. HELP Syntax

```
HELP 'search_string'
```

The `HELP` statement returns online information from the MySQL Reference manual. Its proper operation requires that the help tables in the `mysql` database be initialized with help topic information (see [Section 5.1.10, “Server-Side Help”](#)).

The `HELP` statement searches the help tables for the given search string and displays the result of the search. The search string is not case sensitive.

The `HELP` statement understands several types of search strings:

- At the most general level, use `contents` to retrieve a list of the top-level help categories:

```
HELP 'contents'
```

- For a list of topics in a given help category, such as `Data Types`, use the category name:

```
HELP 'data types'
```

- For help on a specific help topic, such as the `ASCII() [1101]` function or the `CREATE TABLE` statement, use the associated keyword or keywords:

```
HELP 'ascii'
HELP 'create table'
```

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string will return a list of items or the help information for a single help topic. However, you can tell what kind of response `HELP` returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the `mysql` client, but note that `mysql` itself reformats `HELP` result sets in a different way.

- Empty result set

No match could be found for the search string.

- Result set containing a single row with three columns

This means that the search string yielded a hit for the help topic. The result has three columns:

- `name`: The topic name.
- `description`: Descriptive help text for the topic.
- `example`: Usage example or examples. This column might be blank.

Example: `HELP 'replace'`

Yields:

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```

- Result set containing multiple rows with two columns

This means that the search string matched many help topics. The result set indicates the help topic names:

- `name`: The help topic name.
- `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

Example: `HELP 'status'`

Yields:

name	is_it_category
SHOW	N
SHOW ENGINE	N
SHOW MASTER STATUS	N
SHOW PROCEDURE STATUS	N
SHOW SLAVE STATUS	N
SHOW STATUS	N
SHOW TABLE STATUS	N

- Result set containing multiple rows with three columns

This means the search string matches a category. The result set contains category entries:

- `source_category_name`: The help category name.
- `name`: The category or topic name
- `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

Example: `HELP 'functions'`

Yields:

source_category_name	name	is_it_category
Functions	CREATE FUNCTION	N

Functions	DROP FUNCTION	N
Functions	Bit Functions	Y
Functions	Comparison operators	Y
Functions	Control flow functions	Y
Functions	Date and Time Functions	Y
Functions	Encryption Functions	Y
Functions	Information Functions	Y
Functions	Logical operators	Y
Functions	Miscellaneous Functions	Y
Functions	Numeric Functions	Y
Functions	String Functions	Y

13.8.4. USE Syntax

USE *db_name*

The `USE db_name` statement tells MySQL to use the *db_name* database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another `USE` statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;    # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;    # selects from db2.mytable
```

Making a particular database the default by means of the `USE` statement does not preclude you from accessing tables in other databases. The following example accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
WHERE author.editor_id = db2.editor.editor_id;
```

Chapter 14. Storage Engines

Table of Contents

14.1. Setting the Storage Engine	1527
14.2. Overview of MySQL Storage Engine Architecture	1527
14.2.1. Pluggable Storage Engine Architecture	1528
14.2.2. The Common Database Server Layer	1528
14.3. The InnoDB Storage Engine	1529
14.3.1. InnoDB as the Default MySQL Storage Engine	1531
14.3.2. Configuring InnoDB	1534
14.3.3. Using Per-Table Tablespace	1538
14.3.4. InnoDB Startup Options and System Variables	1542
14.3.5. Creating and Using InnoDB Tables	1576
14.3.6. Adding, Removing, or Resizing InnoDB Data and Log Files	1591
14.3.7. Backing Up and Recovering an InnoDB Database	1592
14.3.8. Moving or Copying InnoDB Tables to Another Machine	1596
14.3.9. The InnoDB Transaction Model and Locking	1596
14.3.10. InnoDB Multi-Versioning	1609
14.3.11. InnoDB Table and Index Structures	1610
14.3.12. InnoDB Disk I/O and File Space Management	1613
14.3.13. InnoDB Error Handling	1615
14.3.14. InnoDB Performance Tuning and Troubleshooting	1620
14.3.15. Limits on InnoDB Tables	1632
14.4. New Features of InnoDB 1.1	1635
14.4.1. Introduction to InnoDB 1.1	1635
14.4.2. Fast Index Creation in the InnoDB Storage Engine	1638
14.4.3. InnoDB Data Compression	1642
14.4.4. InnoDB File-Format Management	1651
14.4.5. How InnoDB Stores Variable-Length Columns	1656
14.4.6. InnoDB INFORMATION_SCHEMA tables	1658
14.4.7. InnoDB Performance and Scalability Enhancements	1666
14.4.8. Changes for Flexibility, Ease of Use and Reliability	1678
14.4.9. Installing the InnoDB Storage Engine	1683
14.4.10. Upgrading the InnoDB Storage Engine	1684
14.4.11. Downgrading the InnoDB Storage Engine	1684
14.4.12. InnoDB Storage Engine Change History	1684
14.4.13. Third-Party Software	1685
14.4.14. List of Parameters Changed in InnoDB 1.1 and InnoDB Plugin 1.0	1688
14.5. The MyISAM Storage Engine	1691
14.5.1. MyISAM Startup Options	1693
14.5.2. Space Needed for Keys	1695
14.5.3. MyISAM Table Storage Formats	1695
14.5.4. MyISAM Table Problems	1698
14.6. The MEMORY Storage Engine	1699
14.7. The CSV Storage Engine	1703
14.7.1. Repairing and Checking CSV Tables	1704
14.7.2. CSV Limitations	1705
14.8. The ARCHIVE Storage Engine	1705
14.9. The BLACKHOLE Storage Engine	1706
14.10. The MERGE Storage Engine	1709
14.10.1. MERGE Table Advantages and Disadvantages	1711

14.10.2. MERGE Table Problems	1712
14.11. The FEDERATED Storage Engine	1714
14.11.1. FEDERATED Storage Engine Overview	1714
14.11.2. How to Create FEDERATED Tables	1715
14.11.3. FEDERATED Storage Engine Notes and Tips	1718
14.11.4. FEDERATED Storage Engine Resources	1719
14.12. The EXAMPLE Storage Engine	1719
14.13. Other Storage Engines	1720

MySQL supports several storage engines that act as handlers for different table types. MySQL storage engines include both those that handle transaction-safe tables and those that handle nontransaction-safe tables.

MySQL Server uses a pluggable storage engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

To determine which storage engines your server supports by using the [SHOW ENGINES](#) statement. The value in the [Support](#) column indicates whether an engine can be used. A value of [YES](#), [NO](#), or [DEFAULT](#) indicates that an engine is available, not available, or available and currently set as the default storage engine.

```
mysql> SHOW ENGINES\G
***** 1. row *****
    Engine: FEDERATED
    Support: NO
    Comment: Federated MySQL storage engine
Transactions: NULL
    XA: NULL
    Savepoints: NULL
***** 2. row *****
    Engine: MRG_MYISAM
    Support: YES
    Comment: Collection of identical MyISAM tables
Transactions: NO
    XA: NO
    Savepoints: NO
***** 3. row *****
    Engine: MyISAM
    Support: DEFAULT
    Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
    XA: NO
    Savepoints: NO
...

```

This chapter describes each of the MySQL storage engines except for [NDBCLUSTER](#), which is covered in [Chapter 17, MySQL Cluster NDB 7.2](#). It also contains a description of the pluggable storage engine architecture (see [Section 14.2, "Overview of MySQL Storage Engine Architecture"](#)).

For information about storage engine support offered in commercial MySQL Server binaries, see [MySQL Enterprise Server 5.1](#), on the MySQL Web site. The storage engines available might depend on which edition of Enterprise Server you are using.

For answers to some commonly asked questions about MySQL storage engines, see [Section B.2, "MySQL 5.5 FAQ: Storage Engines"](#).

MySQL 5.5 supported storage engines

- [InnoDB](#): A transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. [InnoDB](#) row-level locking (without escalation to coarser

granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. **InnoDB** stores user data in clustered indexes to reduce I/O for common queries based on primary keys. To maintain data integrity, **InnoDB** also supports **FOREIGN KEY** referential-integrity constraints. **InnoDB** is the default storage engine as of MySQL 5.5.5.

- **MyISAM**: The MySQL storage engine that is used the most in Web, data warehousing, and other application environments. **MyISAM** is supported in all MySQL configurations, and is the default storage engine prior to MySQL 5.5.5.
- **Memory**: Stores all data in RAM for extremely fast access in environments that require quick lookups of reference and other like data. This engine was formerly known as the **HEAP** engine.
- **Merge**: Enables a MySQL DBA or developer to logically group a series of identical **MyISAM** tables and reference them as one object. Good for VLDB environments such as data warehousing.
- **Archive**: Provides the perfect solution for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.
- **Federated**: Offers the ability to link separate MySQL servers to create one logical database from many physical servers. Very good for distributed or data mart environments.
- **CSV**: The CSV storage engine stores data in text files using comma-separated values format. You can use the CSV engine to easily exchange data between other software and applications that can import and export in CSV format.
- **Blackhole**: The Blackhole storage engine accepts but does not store data and retrievals always return an empty set. The functionality can be used in distributed database design where data is automatically replicated, but not stored locally.
- **Example**: The Example storage engine is “stub” engine that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them. The purpose of this engine is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

It is important to remember that you are not restricted to using the same storage engine for an entire server or schema: you can use a different storage engine for each table in your schema.

Choosing a Storage Engine

The various storage engines provided with MySQL are designed with different use cases in mind. To use the pluggable storage architecture effectively, it is good to have an idea of the advantages and disadvantages of the various storage engines. The following table provides an overview of some storage engines provided with MySQL:

Table 14.1. Storage Engines Feature Summary

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Storage limits	256TB	RAM	64TB	None	384EB
Transactions	No	No	Yes	No	Yes
Locking granularity	Table	Table	Row	Table	Row
MVCC	No	No	Yes	No	No
Geospatial data type support	Yes	No	Yes	Yes	Yes

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Geospatial indexing support	Yes	No	No	No	No
B-tree indexes	Yes	Yes	Yes	No	Yes
Hash indexes	No	Yes	No ^a	No	Yes
Full-text search indexes	Yes	No	Yes ^b	No	No
Clustered indexes	No	No	Yes	No	No
Data caches	No	N/A	Yes	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Compressed data	Yes ^c	No	Yes ^d	Yes	No
Encrypted data ^e	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Replication support ^f	Yes	Yes	Yes	Yes	Yes
Foreign key support	No	No	Yes	No	No
Backup / point-in-time recovery ^g	Yes	Yes	Yes	Yes	Yes
Query cache support	Yes	Yes	Yes	Yes	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

^aInnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.

^bInnoDB support for FULLTEXT indexes is available in MySQL 5.6.4 and higher.

^cCompressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.

^dCompressed InnoDB tables require the InnoDB Barracuda file format.

^eImplemented in the server (via encryption functions), rather than in the storage engine.

^fImplemented in the server, rather than in the storage engine.

^gImplemented in the server, rather than in the storage engine.

14.1. Setting the Storage Engine

When you create a new table, you can specify which storage engine to use by adding an `ENGINE` table option to the `CREATE TABLE` statement:

```
CREATE TABLE t (i INT) ENGINE = INNODB;
```

If you omit the `ENGINE` option, the default storage engine is used. The default engine is `InnoDB` as of MySQL 5.5.5 (`MyISAM` before 5.5.5). You can specify the default engine by using the `--default-storage-engine` [441] server startup option, or by setting the `default-storage-engine` [441] option in the `my.cnf` configuration file.

You can set the default storage engine to be used during the current session by setting the `default_storage_engine` [494] variable:

```
SET default_storage_engine=MYISAM;
```

When MySQL is installed on Windows using the MySQL Configuration Wizard, the `InnoDB` or `MyISAM` storage engine can be selected as the default. See [Section 2.3.6.5, “The Database Usage Dialog”](#).

To convert a table from one storage engine to another, use an `ALTER TABLE` statement that indicates the new engine:

```
ALTER TABLE t ENGINE = MYISAM;
```

See [Section 13.1.17, “CREATE TABLE Syntax”](#), and [Section 13.1.7, “ALTER TABLE Syntax”](#).

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine. This behavior is convenient when you want to copy tables between MySQL servers that support different storage engines. (For example, in a replication setup, perhaps your master server supports transactional storage engines for increased safety, but the slave servers use only nontransactional storage engines for greater speed.)

This automatic substitution of the default storage engine for unavailable engines can be confusing for new MySQL users. A warning is generated whenever a storage engine is automatically changed. To prevent this from happening if the desired engine is unavailable, enable the `NO_ENGINE_SUBSTITUTION` [614] SQL mode. In this case, an error occurs instead of a warning and the table is not created or altered if the desired engine is unavailable. See [Section 5.1.7, “Server SQL Modes”](#).

For new tables, MySQL always creates an `.frm` file to hold the table and column definitions. The table's index and data may be stored in one or more other files, depending on the storage engine. The server creates the `.frm` file above the storage engine level. Individual storage engines create any additional files required for the tables that they manage. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in [Section 9.2.3, “Mapping of Identifiers to File Names”](#).

A database may contain tables of different types. That is, tables need not all be created with the same storage engine.

14.2. Overview of MySQL Storage Engine Architecture

The MySQL pluggable storage engine architecture enables a database professional to select a specialized storage engine for a particular application need while being completely shielded from the need to manage any specific application coding requirements. The MySQL server architecture isolates the application programmer and DBA from all of the low-level implementation details at the storage level, providing a consistent and easy application model and API. Thus, although there are different capabilities across different storage engines, the application is shielded from these differences.

The pluggable storage engine architecture provides a standard set of management and support services that are common among all underlying storage engines. The storage engines themselves are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level.

This efficient and modular architecture provides huge benefits for those wishing to specifically target a particular application need—such as data warehousing, transaction processing, or high availability situations—while enjoying the advantage of utilizing a set of interfaces and services that are independent of any one storage engine.

The application programmer and DBA interact with the MySQL database through Connector APIs and service layers that are above the storage engines. If application changes bring about requirements that demand the underlying storage engine change, or that one or more storage engines be added to support new needs, no significant coding or process changes are required to make things work. The MySQL server architecture shields the application from the underlying complexity of the storage engine by presenting a consistent and easy-to-use API that applies across storage engines.

14.2.1. Pluggable Storage Engine Architecture

MySQL Server uses a pluggable storage engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

Plugging in a Storage Engine

Before a storage engine can be used, the storage engine plugin shared library must be loaded into MySQL using the `INSTALL PLUGIN` statement. For example, if the `EXAMPLE` engine plugin is named `example` and the shared library is named `ha_example.so`, you load it with the following statement:

```
mysql> INSTALL PLUGIN example SONAME 'ha_example.so';
```

To install a pluggable storage engine, the plugin file must be located in the MySQL plugin directory, and the user issuing the `INSTALL PLUGIN` statement must have `INSERT [696]` privilege for the `mysql.plugin` table.

The shared library must be located in the MySQL server plugin directory, the location of which is given by the `plugin_dir [537]` system variable.

Unplugging a Storage Engine

To unplug a storage engine, use the `UNINSTALL PLUGIN` statement:

```
mysql> UNINSTALL PLUGIN example;
```

If you unplug a storage engine that is needed by existing tables, those tables become inaccessible, but will still be present on disk (where applicable). Ensure that there are no tables using a storage engine before you unplug the storage engine.

14.2.2. The Common Database Server Layer

A MySQL pluggable storage engine is the component in the MySQL database server that is responsible for performing the actual data I/O operations for a database as well as enabling and enforcing certain feature sets that target a specific application need. A major benefit of using specific storage engines is that you are only delivered the features needed for a particular application, and therefore you have less system overhead in the database, with the end result being more efficient and higher database performance. This is one of the reasons that MySQL has always been known to have such high performance, matching or beating proprietary monolithic databases in industry standard benchmarks.

From a technical perspective, what are some of the unique supporting infrastructure components that are in a storage engine? Some of the key feature differentiations include:

- *Concurrency*: Some applications have more granular lock requirements (such as row-level locks) than others. Choosing the right locking strategy can reduce overhead and therefore improve overall performance. This area also includes support for capabilities such as multi-version concurrency control or “snapshot” read.
- *Transaction Support*: Not every application needs transactions, but for those that do, there are very well defined requirements such as ACID compliance and more.
- *Referential Integrity*: The need to have the server enforce relational database referential integrity through DDL defined foreign keys.
- *Physical Storage*: This involves everything from the overall page size for tables and indexes as well as the format used for storing data to physical disk.
- *Index Support*: Different application scenarios tend to benefit from different index strategies. Each storage engine generally has its own indexing methods, although some (such as B-tree indexes) are common to nearly all engines.
- *Memory Caches*: Different applications respond better to some memory caching strategies than others, so although some memory caches are common to all storage engines (such as those used for user connections or MySQL's high-speed Query Cache), others are uniquely defined only when a particular storage engine is put in play.
- *Performance Aids*: This includes multiple I/O threads for parallel operations, thread concurrency, database checkpointing, bulk insert handling, and more.
- *Miscellaneous Target Features*: This may include support for geospatial operations, security restrictions for certain data manipulation operations, and other similar features.

Each set of the pluggable storage engine infrastructure components are designed to offer a selective set of benefits for a particular application. Conversely, avoiding a set of component features helps reduce unnecessary overhead. It stands to reason that understanding a particular application's set of requirements and selecting the proper MySQL storage engine can have a dramatic impact on overall system efficiency and performance.

14.3. The **InnoDB** Storage Engine

InnoDB is a high-reliability and high-performance storage engine for MySQL. Starting with MySQL 5.5, it is the default MySQL storage engine. Key advantages of **InnoDB** include:

- Its design follows the **ACID** model, with **transactions** featuring **commit**, **rollback**, and **crash-recovery** capabilities to protect user data.
- Row-level **locking** and Oracle-style **consistent reads** increase multi-user concurrency and performance.
- **InnoDB** tables arrange your data on disk to optimize common queries based on **primary keys**. Each **InnoDB** table has a primary key index called the **clustered index** that organizes the data to minimize I/O for primary key lookups.
- To maintain data integrity, **InnoDB** also supports **FOREIGN KEY** referential-integrity constraints.
- You can freely mix **InnoDB** tables with tables from other MySQL storage engines, even within the same statement. For example, you can use a join operation to combine data from **InnoDB** and **MEMORY** tables in a single query.

To determine whether your server supports InnoDB use the `SHOW ENGINES` statement. See [Section 13.7.5.17, “SHOW ENGINES Syntax”](#).

Table 14.2. InnoDB Storage Engine Features

Storage limits	64TB	Transactions	Yes	Locking granularity	Row
MVCC	Yes	Geospatial data type support	Yes	Geospatial indexing support	No
B-tree indexes	Yes	Hash indexes	No ^a	Full-text search indexes	Yes ^b
Clustered indexes	Yes	Data caches	Yes	Index caches	Yes
Compressed data	Yes ^c	Encrypted data^d	Yes	Cluster database support	No
Replication support^e	Yes	Foreign key support	Yes	Backup / point-in-time recovery^f	Yes
Query cache support	Yes	Update statistics for data dictionary	Yes		

^aInnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.

^bInnoDB support for FULLTEXT indexes is available in MySQL 5.6.4 and higher.

^cCompressed InnoDB tables require the InnoDB Barracuda file format.

^dImplemented in the server (via encryption functions), rather than in the storage engine.

^eImplemented in the server, rather than in the storage engine.

^fImplemented in the server, rather than in the storage engine.

InnoDB has been designed for maximum performance when processing large data volumes. Its CPU efficiency is probably not matched by any other disk-based relational database engine.

The InnoDB storage engine maintains its own buffer pool for caching data and indexes in main memory. InnoDB stores its tables and indexes in a tablespace, which may consist of several files (or raw disk partitions). This is different from, for example, MyISAM tables where each table is stored using separate files. InnoDB tables can be very large even on operating systems where file size is limited to 2GB.

InnoDB is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see <http://www.mysql.com/company/legal/licensing/>.

Additional Resources

- A forum dedicated to the InnoDB storage engine is available at <http://forums.mysql.com/list.php?22>.
- The InnoDB storage engine in MySQL 5.5 releases includes a number performance improvements that in MySQL 5.1 were only available by installing the InnoDB Plugin. This latest InnoDB (now known as InnoDB 1.1) offers new features, improved performance and scalability, enhanced reliability and new capabilities for flexibility and ease of use. Among the top features are Fast Index Creation, table and index compression, file format management, new `INFORMATION_SCHEMA` tables, capacity tuning, multiple background I/O threads, multiple buffer pools, and group commit.

For information about these features, see [Section 14.4, “New Features of InnoDB 1.1”](#).

- The MySQL Enterprise Backup product lets you back up a running MySQL database, including InnoDB and MyISAM tables, with minimal disruption to operations while producing a consistent snapshot of the database. When MySQL Enterprise Backup is copying InnoDB tables, reads and writes to both InnoDB and MyISAM tables can continue. During the copying of MyISAM and other non-InnoDB tables, reads (but not writes) to those tables are permitted. In addition, MySQL Enterprise Backup can create

compressed backup files, and back up subsets of [InnoDB](#) tables. In conjunction with the MySQL binary log, you can perform point-in-time recovery. MySQL Enterprise Backup is included as part of the MySQL Enterprise subscription.

For a more complete description of MySQL Enterprise Backup, see [MySQL Enterprise Backup User's Guide \(Version 3.8\)](#).

14.3.1. InnoDB as the Default MySQL Storage Engine

MySQL has a well-earned reputation for being easy-to-use and delivering performance and scalability. In previous versions of MySQL, MyISAM was the default storage engine. In our experience, most users never changed the default settings. With MySQL 5.5, InnoDB becomes the default storage engine. Again, we expect most users will not change the default settings. But, because of InnoDB, the default settings deliver the benefits users expect from their RDBMS: ACID Transactions, Referential Integrity, and Crash Recovery. Let's explore how using InnoDB tables improves your life as a MySQL user, DBA, or developer.

Trends in Storage Engine Usage

In the first years of MySQL growth, early web-based applications didn't push the limits of concurrency and availability. In 2010, hard drive and memory capacity and the performance/price ratio have all gone through the roof. Users pushing the performance boundaries of MySQL care a lot about reliability and crash recovery. MySQL databases are big, busy, robust, distributed, and important.

InnoDB hits the sweet spot of these top user priorities. The trend of storage engine usage has shifted in favor of the more efficient InnoDB. With MySQL 5.5, the time is right to make InnoDB the default storage engine.

Consequences of InnoDB as Default MySQL Storage Engine

Starting from MySQL 5.5.5, the default storage engine for new tables is InnoDB. This change applies to newly created tables that don't specify a storage engine with a clause such as `ENGINE=MyISAM`. (Given this change of default behavior, MySQL 5.5 might be a logical point to evaluate whether your tables that do use MyISAM could benefit from switching to InnoDB.)

The `mysql` and `information_schema` databases, that implement some of the MySQL internals, still use MyISAM. In particular, you cannot switch the grant tables to use InnoDB.

Benefits of InnoDB Tables

If you use MyISAM tables but aren't tied to them for technical reasons, you'll find many things more convenient when you use InnoDB tables in MySQL 5.5:

- If your server crashes because of a hardware or software issue, regardless of what was happening in the database at the time, you don't need to do anything special after restarting the database. InnoDB automatically finalizes any changes that were committed before the time of the crash, and undoes any changes that were in process but not committed. Just restart and continue where you left off.
- The InnoDB buffer pool caches table and index data as the data is accessed. Frequently used data is processed directly from memory. This cache applies to so many types of information, and speeds up processing so much, that dedicated database servers assign up to 80% of their physical memory to the InnoDB buffer pool.
- If you split up related data into different tables, you can set up foreign keys that enforce referential integrity. Update or delete data, and the related data in other tables is updated or deleted automatically. Try to insert data into a secondary table without corresponding data in the primary table, and it gets kicked out automatically.

- If data becomes corrupted on disk or in memory, a checksum mechanism alerts you to the bogus data before you use it.
- When you design your database with appropriate primary key columns for each table, operations involving those columns are automatically optimized. It is very fast to reference the primary key columns in `WHERE` clauses, `ORDER BY` clauses, `GROUP BY` clauses, and join operations.
- Inserts, updates, deletes are optimized by an automatic mechanism called change buffering. InnoDB not only allows concurrent read and write access to the same table, it caches changed data to streamline disk I/O.
- Performance benefits are not limited to giant tables with long-running queries. When the same rows are accessed over and over from a table, a feature called the Adaptive Hash Index takes over to make these lookups even faster, as if they came out of a hash table.

Best Practices for InnoDB Tables

If you have been using InnoDB for a long time, you already know about features like transactions and foreign keys. If not, read about them throughout this chapter. To make a long story short:

- Specify a primary key for every table using the most frequently queried column or columns, or an auto-increment value if there isn't an obvious primary key.
- Embrace the idea of joins, where data is pulled from multiple tables based on identical ID values from those tables. For fast join performance, define foreign keys on the join columns, and declare those columns with the same datatype in each table. The foreign keys also propagate deletes or updates to all affected tables, and prevent insertion of data in a child table if the corresponding IDs are not present in the parent table.
- Turn off autocommit. Committing hundreds of times a second puts a cap on performance (limited by the write speed of your storage device).
- Bracket sets of related changes, logical units of work, with `START TRANSACTION` and `COMMIT` statements. While you don't want to commit too often, you also don't want to issue huge batches of `INSERT`, `UPDATE`, or `DELETE` statements that run for hours without committing.
- Stop using `LOCK TABLE` statements. InnoDB can handle multiple sessions all reading and writing to the same table at once, without sacrificing reliability or high performance. To get exclusive write access to a set of rows, use the `SELECT ... FOR UPDATE` syntax to lock just the rows you intend to update.
- Enable the `innodb_file_per_table` option to put the data and indexes for individual tables into separate files, instead of in a single giant system tablespace. (This setting is required to use some of the other features, such as table compression and fast truncation.)
- Evaluate whether your data and access patterns benefit from the new InnoDB table compression feature (`ROW_FORMAT=COMPRESSED` on the `CREATE TABLE` statement. You can compress InnoDB tables without sacrificing read/write capability.
- Run your server with the option `--sql_mode=NO_ENGINE_SUBSTITUTION` to prevent tables being created with a different storage engine if there is an issue with the one specified in the `ENGINE=` clause of `CREATE TABLE`.

Recent Improvements for InnoDB Tables (from the Plugin Era)

If you have experience with InnoDB, but not the recent incarnation known as the InnoDB Plugin, read about the latest enhancements in [Section 14.4, “New Features of InnoDB 1.1”](#). To make a long story short:

- You can compress tables and associated indexes.
- You can create and drop indexes with much less performance or availability impact than before.
- Truncating a table is very fast, and can free up disk space for the operating system to reuse, rather than freeing up space within the system tablespace that only InnoDB could reuse.
- The storage layout for table data is more efficient for BLOBs and long text fields.
- You can monitor the internal workings of the storage engine by querying `INFORMATION_SCHEMA` tables.
- You can monitor the performance details of the storage engine by querying `performance_schema` tables.
- There are many many performance improvements. In particular, crash recovery, the automatic process that makes all data consistent when the database is restarted, is fast and reliable. (Now much much faster than long-time InnoDB users are used to.) The bigger the database, the more dramatic the speedup.

Most new performance features are automatic, or at most require setting a value for a configuration option. For details, see [Section 14.4.7, “InnoDB Performance and Scalability Enhancements”](#). For InnoDB-specific tuning techniques you can apply in your application code, see [Section 8.5, “Optimizing for InnoDB Tables”](#). Advanced users can review [Section 14.3.4, “InnoDB Startup Options and System Variables”](#).

Testing and Benchmarking with InnoDB as Default Storage Engine

Even before completing your upgrade to MySQL 5.5, you can preview whether your database server or application works correctly with InnoDB as the default storage engine. To set up InnoDB as the default storage engine with an earlier MySQL release, either specify on the command line `--default-storage-engine=InnoDB`, or add to your `my.cnf` file `default-storage-engine=innodb` in the `[mysqld]` section, then restart the server.

Since changing the default storage engine only affects new tables as they are created, run all your application installation and setup steps to confirm that everything installs properly. Then exercise all the application features to make sure all the data loading, editing, and querying features work. If a table relies on some MyISAM-specific feature, you'll receive an error; add the `ENGINE=MyISAM` clause to the `CREATE TABLE` statement to avoid the error. (For example, tables that rely on full-text search must be MyISAM tables rather than InnoDB ones.)

If you didn't make a deliberate decision about the storage engine, and you just want to preview how certain tables work when they're created under InnoDB, issue the command `ALTER TABLE table_name ENGINE=InnoDB;` for each table. Or, to run test queries and other statements without disturbing the original table, make a copy like so:

```
CREATE TABLE InnoDB_Table (...) ENGINE=InnoDB AS SELECT * FROM MyISAM_Table;
```

Since there are so many performance enhancements in the InnoDB that is part of MySQL 5.5, to get a true idea of the performance with a full application under a realistic workload, install the real MySQL 5.5 and run benchmarks.

Test the full application lifecycle, from installation, through heavy usage, and server restart. Kill the server process while the database is busy to simulate a power failure, and verify that the data is recovered successfully when you restart the server.

Test any replication configurations, especially if you use different MySQL versions and options on the master and the slaves.

Verifying that InnoDB is the Default Storage Engine

To know what the status of InnoDB is, whether you're doing what-if testing with an older MySQL or comprehensive testing with MySQL 5.5:

- Issue the command `SHOW VARIABLES LIKE 'have_innodb'`; to confirm that InnoDB is available at all. If the result is `NO`, you have a `mysqld` binary that was compiled without InnoDB support and you need to get a different one. If the result is `DISABLED`, go back through your startup options and configuration file and get rid of any `skip-innodb` option.
- Issue the command `SHOW ENGINES`; to see all the different MySQL storage engines. Look for `DEFAULT` in the InnoDB line.

14.3.2. Configuring InnoDB

The first decisions to make about InnoDB configuration involve how to lay out InnoDB data files, and how much memory to allocate for the InnoDB storage engine. You record these choices either by recording them in a configuration file that MySQL reads at startup, or by specifying them as command-line options in a startup script. The full list of options, descriptions, and allowed parameter values is at [Section 14.3.4, “InnoDB Startup Options and System Variables”](#).

Overview of InnoDB Tablespace and Log Files

Two important disk-based resources managed by the InnoDB storage engine are its tablespace data files and its log files. If you specify no InnoDB configuration options, MySQL creates an auto-extending 10MB data file named `ibdata1` and two 5MB log files named `ib_logfile0` and `ib_logfile1` in the MySQL data directory. To get good performance, explicitly provide InnoDB parameters as discussed in the following examples. Naturally, edit the settings to suit your hardware and requirements.

The examples shown here are representative. See [Section 14.3.4, “InnoDB Startup Options and System Variables”](#) for additional information about InnoDB-related configuration parameters.

Considerations for Storage Devices

In some cases, database performance improves if the data is not all placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. The example illustrates how to do this. It places the two data files on different disks and places the log files on the third disk. InnoDB fills the tablespace beginning with the first data file. You can also use raw disk partitions (raw devices) as InnoDB data files, which may speed up I/O. See [Section 14.3.3.1, “Using Raw Devices for the Shared Tablespace”](#).



Caution

InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. **However, it cannot do so** if the underlying operating system or hardware does not work as advertised. Many operating systems or disk subsystems may delay or reorder write operations to improve performance. On some operating systems, the very `fsync()` system call that should wait until all unwritten data for a file has been flushed might actually return before the data has been flushed to stable storage. Because of this, an operating system crash or a power outage may destroy recently committed data, or in the worst case, even corrupt the database because of write operations having been reordered. If data integrity is important to you, perform some “pull-the-plug” tests before using anything in production. On Mac OS X 10.3 and up, InnoDB uses a special `fcntl()` file flush method. Under Linux, it is advisable to **disable the write-back cache**.

On ATA/SATA disk drives, a command such `hdparm -W0 /dev/hda` may work to disable the write-back cache. **Beware that some drives or disk controllers may be unable to disable the write-back cache.**



Caution

If reliability is a consideration for your data, do not configure InnoDB to use data files or log files on NFS volumes. Potential problems vary according to OS and version of NFS, and include such issues as lack of protection from conflicting writes, and limitations on maximum file sizes.

Specifying the Location and Size for InnoDB Tablespace Files

To set up the InnoDB tablespace files, use the `innodb_data_file_path` [1552] option in the `[mysqld]` section of the `my.cnf` option file. On Windows, you can use `my.ini` instead. The value of `innodb_data_file_path` [1552] should be a list of one or more data file specifications. If you name more than one data file, separate them by semicolon (“;”) characters:

```
innodb_data_file_path=datafile_spec1[:datafile_spec2]...
```

For example, the following setting explicitly creates a tablespace having the same characteristics as the default:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

This setting configures a single 10MB data file named `ibdata1` that is auto-extending. No location for the file is given, so by default, InnoDB creates it in the MySQL data directory.

Sizes are specified using `K`, `M`, or `G` suffix letters to indicate units of KB, MB, or GB.

A tablespace containing a fixed-size 50MB data file named `ibdata1` and a 50MB auto-extending file named `ibdata2` in the data directory can be configured like this:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The full syntax for a data file specification includes the file name, its size, and several optional attributes:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

The `autoextend` and `max` attributes can be used only for the last data file in the `innodb_data_file_path` [1552] line.

If you specify the `autoextend` option for the last data file, InnoDB extends the data file if it runs out of free space in the tablespace. The increment is 8MB at a time by default. To modify the increment, change the `innodb_autoextend_increment` [1548] system variable.

If the disk becomes full, you might want to add another data file on another disk. For tablespace reconfiguration instructions, see [Section 14.3.6, “Adding, Removing, or Resizing InnoDB Data and Log Files”](#).

InnoDB is not aware of the file system maximum file size, so be cautious on file systems where the maximum file size is a small value such as 2GB. To specify a maximum size for an auto-extending data file, use the `max` attribute following the `autoextend` attribute. Use the `max` attribute only in cases where constraining disk usage is of critical importance, because exceeding the maximum size causes a fatal error, possibly including a crash. The following configuration permits `ibdata1` to grow up to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

InnoDB creates tablespace files in the MySQL data directory by default. To specify a location explicitly, use the `innodb_data_home_dir` [1552] option. For example, to use two files named `ibdata1` and `ibdata2` but create them in the `/ibdata` directory, configure InnoDB like this:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```



Note

InnoDB does not create directories, so make sure that the `/ibdata` directory exists before you start the server. This is also true of any log file directories that you configure. Use the Unix or DOS `mkdir` command to create any necessary directories.

Make sure that the MySQL server has the proper access rights to create files in the data directory. More generally, the server must have access rights in any directory where it needs to create data files or log files.

InnoDB forms the directory path for each data file by textually concatenating the value of `innodb_data_home_dir` [1552] to the data file name, adding a path name separator (slash or backslash) between values if necessary. If the `innodb_data_home_dir` [1552] option is not specified in `my.cnf` at all, the default value is the “dot” directory `.`, which means the MySQL data directory. (The MySQL server changes its current working directory to its data directory when it begins executing.)

If you specify `innodb_data_home_dir` [1552] as an empty string, you can specify absolute paths for the data files listed in the `innodb_data_file_path` [1552] value. The following example is equivalent to the preceding one:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

Specifying InnoDB Configuration Options

Sample `my.cnf` file for small systems. Suppose that you have a computer with 512MB RAM and one hard disk. The following example shows possible configuration parameters in `my.cnf` or `my.ini` for InnoDB, including the `autoextend` attribute. The example suits most users, both on Unix and Windows, who do not want to distribute InnoDB data files and log files onto several disks. It creates an auto-extending data file `ibdata1` and two InnoDB log files `ib_logfile0` and `ib_logfile1` in the MySQL data directory.

```
[mysqld]
# You can write your other MySQL server options here
# ...
# Data files must be able to hold your data and indexes.
# Make sure that you have enough free disk space.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory
innodb_buffer_pool_size=256M
innodb_additional_mem_pool_size=20M
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=64M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Note that data files must be less than 2GB in some file systems. The combined size of the log files must be less than 4GB. The combined size of data files must be at least 10MB.

Setting Up the InnoDB System Tablespace

When you create an InnoDB system tablespace for the first time, it is best that you start the MySQL server from the command prompt. InnoDB then prints the information about the database creation to the screen, so you can see what is happening. For example, on Windows, if `mysqld` is located in `C:\Program Files\MySQL\MySQL Server 5.5\bin`, you can start it like this:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --console
```

If you do not send server output to the screen, check the server's error log to see what InnoDB prints during the startup process.

For an example of what the information displayed by InnoDB should look like, see [Section 14.3.3.2, "Creating the InnoDB Tablespace"](#).

Editing the MySQL Configuration File

You can place InnoDB options in the `[mysqld]` group of any option file that your server reads when it starts. The locations for option files are described in [Section 4.2.3.3, "Using Option Files"](#).

If you installed MySQL on Windows using the installation and configuration wizards, the option file will be the `my.ini` file located in your MySQL installation directory. See [Section 2.3.3, "Installing MySQL on Microsoft Windows Using MySQL Installer"](#).

If your PC uses a boot loader where the `C:` drive is not the boot drive, your only option is to use the `my.ini` file in your Windows directory (typically `C:\WINDOWS`). You can use the `SET` command at the command prompt in a console window to print the value of `WINDIR`:

```
C:\> SET WINDIR
windir=C:\WINDOWS
```

To make sure that `mysqld` reads options only from a specific file, use the `--defaults-file` [\[248\]](#) option as the first option on the command line when starting the server:

```
mysqld --defaults-file=your_path_to_my.cnf
```

Sample `my.cnf` file for large systems. Suppose that you have a Linux computer with 2GB RAM and three 60GB hard disks at directory paths `/`, `/dr2` and `/dr3`. The following example shows possible configuration parameters in `my.cnf` for InnoDB.

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /db/ibdata1:2000M:/dr2/db/ibdata2:2000M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
```

```
#
# Uncomment the next line if you want to use it
#innodb_thread_concurrency=5
```

Determining the Maximum Memory Allocation for InnoDB



Warning

On 32-bit GNU/Linux x86, be careful not to set memory usage too high. `glibc` may permit the process heap to grow over thread stacks, which crashes your server. It is a risk if the value of the following expression is close to or exceeds 2GB:

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL binaries provided by Oracle Corporation.) and in the worst case also uses `sort_buffer_size + read_buffer_size` additional memory.

Tuning other `mysqld` server parameters. The following values are typical and suit most users:

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Set key_buffer to 5 - 50% of your RAM depending on how much
# you use MyISAM tables, but keep key_buffer_size + InnoDB
# buffer pool size < 80% of your RAM
key_buffer_size=value
```

On Linux, if the kernel is enabled for large page support, `InnoDB` can use large pages to allocate memory for its buffer pool and additional memory pool. See [Section 8.11.4.2, “Enabling Large Page Support”](#).

Turning Off InnoDB

Oracle recommends `InnoDB` as the preferred storage engine for typical database applications, from single-user wikis and blogs running on a local system, to high-end applications pushing the limits of performance. In MySQL 5.5, `InnoDB` is the default storage engine for new tables.

If you do not want to use `InnoDB` tables, start the server with the `--innodb=OFF` [1546] or `--skip-innodb` [1546] option to disable the `InnoDB` storage engine. In this case, because the default storage engine is `InnoDB`, the server will not start unless you also use `--default-storage-engine` [441] to set the default to some other engine.

14.3.3. Using Per-Table Tablespaces

By default, all `InnoDB` tables and indexes are stored in the `system tablespace`. As an alternative, you can store each `InnoDB` table and its indexes in its own file. This feature is called “multiple tablespaces” because each table that is created when this setting is in effect has its own tablespace.

Using multiple tablespaces is useful in a number of situations:

- You can back up or restore a single table quickly without interrupting the use of other `InnoDB` tables, using the MySQL Enterprise Backup product. See [Backing Up and Restoring a Single .ibd File](#) for the procedure and restrictions.
- Storing specific tables on separate physical disks, for I/O optimization or backup purposes.

- Restoring backups of single tables quickly without interrupting the use of other [InnoDB](#) tables.
- Using [compressed row format](#) to compress table data.
- Reclaiming disk space when truncating a table.

Enabling and Disabling Multiple Tablespaces

To enable multiple tablespaces, start the server with the `--innodb_file_per_table [1555]` option. For example, add a line to the `[mysqld]` section of `my.cnf`:

```
[mysqld]
innodb_file_per_table
```

With multiple tablespaces enabled, [InnoDB](#) stores each newly created table in its own `tbl_name.ibd` file in the appropriate database directory. Unlike the [MyISAM](#) storage engine, with its separate `tbl_name.MYD` and `tbl_name.MYI` files for indexes and data, [InnoDB](#) stores the data and the indexes together in a single `.ibd` file. The `tbl_name.frm` file is still created as usual.

If you remove the `innodb_file_per_table [1555]` line from `my.cnf` and restart the server, [InnoDB](#) creates any new tables inside the shared tablespace files.

You can always access both tables in the system tablespace and tables in their own tablespaces, regardless of the file-per-table setting. To move a table from the system tablespace to its own tablespace, or vice versa, you can change the `innodb_file_per_table [1555]` setting and issue the command:

```
-- Move table from system tablespace to its own tablespace.
SET GLOBAL innodb_file_per_table=1;
ALTER TABLE table_name ENGINE=InnoDB;
-- Move table from its own tablespace to system tablespace.
SET GLOBAL innodb_file_per_table=0;
ALTER TABLE table_name ENGINE=InnoDB;
```



Note

[InnoDB](#) always needs the shared tablespace because it puts its internal data dictionary and undo logs there. The `.ibd` files are not sufficient for [InnoDB](#) to operate.

When a table is moved out of the system tablespace into its own `.ibd` file, the data files that make up the system tablespace remain the same size. The space formerly occupied by the table can be reused for new [InnoDB](#) data, but is not reclaimed for use by the operating system. When moving large [InnoDB](#) tables out of the system tablespace, where disk space is limited, you might prefer to turn on `innodb_file_per_table` and then recreate the entire instance using the `mysqldump` command.

Portability Considerations for .ibd Files

You cannot freely move `.ibd` files between database directories as you can with [MyISAM](#) table files. The table definition stored in the [InnoDB](#) shared tablespace includes the database name. The transaction IDs and log sequence numbers stored in the tablespace files also differ between databases.

To move an `.ibd` file and the associated table from one database to another, use a `RENAME TABLE` statement:

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

If you have a “clean” backup of an `.ibd` file, you can restore it to the MySQL installation from which it originated as follows:

1. The table must not have been dropped or truncated since you copied the `.ibd` file, because doing so changes the table ID stored inside the tablespace.
2. Issue this `ALTER TABLE` statement to delete the current `.ibd` file:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

3. Copy the backup `.ibd` file to the proper database directory.
4. Issue this `ALTER TABLE` statement to tell InnoDB to use the new `.ibd` file for the table:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

In this context, a “clean” `.ibd` file backup is one for which the following requirements are satisfied:

- There are no uncommitted modifications by transactions in the `.ibd` file.
- There are no unmerged insert buffer entries in the `.ibd` file.
- Purge has removed all delete-marked index records from the `.ibd` file.
- `mysqld` has flushed all modified pages of the `.ibd` file from the buffer pool to the file.

You can make a clean backup `.ibd` file using the following method:

1. Stop all activity from the `mysqld` server and commit all transactions.
2. Wait until `SHOW ENGINE INNODB STATUS` shows that there are no active transactions in the database, and the main thread status of InnoDB is `Waiting for server activity`. Then you can make a copy of the `.ibd` file.

Another method for making a clean copy of an `.ibd` file is to use the MySQL Enterprise Backup product:

1. Use MySQL Enterprise Backup to back up the InnoDB installation.
2. Start a second `mysqld` server on the backup and let it clean up the `.ibd` files in the backup.

14.3.3.1. Using Raw Devices for the Shared Tablespace

You can use raw disk partitions as data files in the system tablespace. Using a raw disk, you can perform nonbuffered I/O on Windows and on some Unix systems without file system overhead. Perform tests with and without raw partitions to verify whether this change actually improves performance on your system.

When you create a new data file, put the keyword `newraw` immediately after the data file size in `innodb_data_file_path` [1552]. The partition must be at least as large as the size that you specify. Note that 1MB in InnoDB is 1024 × 1024 bytes, whereas 1MB in disk specifications usually means 1,000,000 bytes.

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

The next time you start the server, InnoDB notices the `newraw` keyword and initializes the new partition. However, do not create or change any InnoDB tables yet. Otherwise, when you next restart the server, InnoDB reinitializes the partition and your changes are lost. (As a safety measure InnoDB prevents users from modifying data when any partition with `newraw` is specified.)

After InnoDB has initialized the new partition, stop the server, change `newraw` in the data file specification to `raw`:

```
[mysqld]
```

```
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Graw:/dev/hdd2:2Graw
```

Then restart the server and [InnoDB](#) permits changes to be made.

On Windows, you can allocate a disk partition as a data file like this:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

The `//./` corresponds to the Windows syntax of `\\.\` for accessing physical drives.

When you use a raw disk partition, be sure that it has permissions that enable read and write access by the account used for running the MySQL server. For example, if you run the server as the `mysql` user, the partition must permit read and write access to `mysql`. If you run the server with the `--memlock` [\[453\]](#) option, the server must be run as `root`, so the partition must permit access to `root`.

14.3.3.2. Creating the [InnoDB](#) Tablespace

Suppose that you have installed MySQL and have edited your option file so that it contains the necessary [InnoDB](#) configuration parameters. Before starting MySQL, verify that the directories you have specified for [InnoDB](#) data files and log files exist and that the MySQL server has access rights to those directories. [InnoDB](#) does not create directories, only files. Check also that you have enough disk space for the data and log files.

It is best to run the MySQL server `mysqld` from the command prompt when you first start the server with [InnoDB](#) enabled, not from `mysqld_safe` or as a Windows service. When you run from a command prompt you see what `mysqld` prints and what is happening. On Unix, just invoke `mysqld`. On Windows, start `mysqld` with the `--console` [\[438\]](#) option to direct the output to the console window.

When you start the MySQL server after initially configuring [InnoDB](#) in your option file, [InnoDB](#) creates your data files and log files, and prints something like this:

```
InnoDB: The first specified datafile /home/heikki/data/ibdata1
did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file /home/heikki/data/ibdata1 size to 134217728
InnoDB: Database physically writes the file full: wait...
InnoDB: datafile /home/heikki/data/ibdata2 did not exist:
new to be created
InnoDB: Setting file /home/heikki/data/ibdata2 size to 262144000
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file /home/heikki/data/logs/ib_logfile0 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile0 size
to 5242880
InnoDB: Log file /home/heikki/data/logs/ib_logfile1 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile1 size
to 5242880
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
InnoDB: Started
mysqld: ready for connections
```

At this point [InnoDB](#) has initialized its tablespace and log files. You can connect to the MySQL server with the usual MySQL client programs like `mysql`. When you shut down the MySQL server with `mysqladmin shutdown`, the output is like this:

```
010321 18:33:34 mysqld: Normal shutdown
```

```
010321 18:33:34 mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

You can look at the data file and log directories and you see the files created there. When MySQL is started again, the data files and log files have been created already, so the output is much briefer:

```
InnoDB: Started
mysqld: ready for connections
```

If you add the `innodb_file_per_table [1555]` option to `my.cnf`, InnoDB stores each table in its own `.ibd` file in the same MySQL database directory where the `.frm` file is created. See [Section 14.3.3, “Using Per-Table Tablespaces”](#).

14.3.3.3. Troubleshooting InnoDB I/O Problems

The troubleshooting steps for InnoDB I/O problems depend on when the problem occurs: during startup of the MySQL server, or during normal operations when a DML or DDL statement fails due to problems at the file system level.

Initialization Problems

If something goes wrong when InnoDB attempts to initialize its tablespace or its log files, delete all files created by InnoDB: all `ibdata` files and all `ib_logfile` files. If you already created some InnoDB tables, also delete the corresponding `.frm` files for these tables, and any `.ibd` files if you are using multiple tablespaces, from the MySQL database directories. Then try the InnoDB database creation again. For easiest troubleshooting, start the MySQL server from a command prompt so that you see what is happening.

Runtime Problems

If InnoDB prints an operating system error during a file operation, usually the problem has one of the following solutions:

- Make sure the InnoDB data file directory and the InnoDB log directory exist.
- Make sure `mysqld` has access rights to create files in those directories.
- Make sure `mysqld` can read the proper `my.cnf` or `my.ini` option file, so that it starts with the options that you specified.
- Make sure the disk is not full and you are not exceeding any disk quota.
- Make sure that the names you specify for subdirectories and data files do not clash.
- Doublecheck the syntax of the `innodb_data_home_dir [1552]` and `innodb_data_file_path [1552]` values. In particular, any `MAX` value in the `innodb_data_file_path [1552]` option is a hard limit, and exceeding that limit causes a fatal error.

14.3.4. InnoDB Startup Options and System Variables

This section describes the InnoDB-related command options and system variables. System variables that are true or false can be enabled at server startup by naming them, or disabled by using a `--skip-` prefix. For example, to enable or disable InnoDB checksums, you can use `--innodb_checksums [1551]` or `--skip-innodb_checksums [1551]` on the command line, or `innodb_checksums [1551]` or `skip-innodb_checksums` in an option file. System variables that take a numeric value can be specified as `--var_name=value` on the command line or as `var_name=value` in option files. For more information on specifying options and system variables, see [Section 4.2.3, “Specifying Program Options”](#). Many of the system variables can be changed at runtime (see [Section 5.1.5.2, “Dynamic System Variables”](#)).

Certain options control the locations and layout of the [InnoDB](#) data files. [Section 14.3.2, “Configuring InnoDB”](#) explains how to use these options. Many other options, that you might not use initially, help to tune [InnoDB](#) performance characteristics based on machine capacity and your database [workload](#). The performance-related options are explained in [Section 14.3.14, “InnoDB Performance Tuning and Troubleshooting”](#) and [Section 14.4.7, “InnoDB Performance and Scalability Enhancements”](#).

Table 14.3. InnoDB Option/Variable Reference

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
foreign_key_checks [500]			Yes		Both	Yes
have_innodb [504]			Yes		Global	No
ignore-builtin-innodb [1546]	Yes	Yes			Global	No
- Variable: ignore_builtin_innodb			Yes		Global	No
innodb [1546]	Yes	Yes				
innodb_adaptive_flushing [1547]	Yes	Yes	Yes		Global	Yes
innodb_adaptive_hash_index [1547]	Yes	Yes	Yes		Global	Yes
innodb_additional_mem_pool_size [1548]	Yes	Yes	Yes		Global	No
innodb_autoextend_increment [1548]	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode [1548]	Yes	Yes	Yes		Global	No
innodb_buffer_pool_instances [1548]	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_pages_data [600]				Yes	Global	No
Innodb_buffer_pool_pages_dirty [600]				Yes	Global	No
Innodb_buffer_pool_pages_flushed [600]				Yes	Global	No
Innodb_buffer_pool_pages_free [600]				Yes	Global	No
Innodb_buffer_pool_pages_latched [600]				Yes	Global	No
Innodb_buffer_pool_pages_misc [600]				Yes	Global	No
Innodb_buffer_pool_pages_total [600]				Yes	Global	No
Innodb_buffer_pool_read_ahead [600]				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted [600]				Yes	Global	No
Innodb_buffer_pool_read_requests [600]				Yes	Global	No
Innodb_buffer_pool_reads [600]				Yes	Global	No
innodb_buffer_pool_size [1549]	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_wait_free [601]				Yes	Global	No
Innodb_buffer_pool_write_requests [601]				Yes	Global	No
innodb_change_buffering [1550]	Yes	Yes	Yes		Global	Yes
innodb_checksums [1551]	Yes	Yes	Yes		Global	No
innodb_commit_concurrency [1551]	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets [1552]	Yes	Yes	Yes		Global	Yes
innodb_data_file_path [1552]	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs [601]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_data_home_dir [1552]	Yes	Yes	Yes		Global	No
InnoDB_data_pending_fsyncs [601]				Yes	Global	No
InnoDB_data_pending_reads [601]				Yes	Global	No
InnoDB_data_pending_writes [601]				Yes	Global	No
InnoDB_data_read [601]				Yes	Global	No
InnoDB_data_reads [601]				Yes	Global	No
InnoDB_data_writes [601]				Yes	Global	No
InnoDB_data_written [601]				Yes	Global	No
InnoDB_dblwr_pages_written [601]				Yes	Global	No
InnoDB_dblwr_writes [601]				Yes	Global	No
innodb_doublewrite [1553]	Yes	Yes	Yes		Global	No
innodb_fast_shutdown [1553]	Yes	Yes	Yes		Global	Yes
innodb_file_format [1554]	Yes	Yes	Yes		Global	Yes
innodb_file_format_check [1554]	Yes	Yes	Yes		Global	No
innodb_file_format_max [1555]	Yes	Yes	Yes		Global	Yes
innodb_file_per_table [1555]	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_commit [1556]	Yes	Yes	Yes		Global	Yes
innodb_flush_method [1557]	Yes	Yes	Yes		Global	No
innodb_force_load_corrupted [1558]	Yes	Yes	Yes		Global	No
innodb_force_recovery [1558]	Yes	Yes	Yes		Global	No
InnoDB_have_atomic_builtins [601]				Yes	Global	No
innodb_io_capacity [1559]	Yes	Yes	Yes		Global	Yes
innodb_large_prefix [1560]	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout [1560]	Yes	Yes	Yes		Both	Yes
innodb_locks_unsafe_for_binlog [1561]	Yes	Yes	Yes		Global	No
innodb_log_buffer_size [1563]	Yes	Yes	Yes		Global	No
innodb_log_file_size [1564]	Yes	Yes	Yes		Global	No
innodb_log_files_in_group [1564]	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir [1564]	Yes	Yes	Yes		Global	No
InnoDB_log_waits [602]				Yes	Global	No
InnoDB_log_write_requests [602]				Yes	Global	No
InnoDB_log_writes [602]				Yes	Global	No
innodb_max_dirty_pages_pct [1565]	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag [1565]	Yes	Yes	Yes		Global	Yes
innodb_mirrored_log_groups [1566]	Yes	Yes	Yes		Global	No
innodb_old_blocks [1566]	Yes	Yes	Yes		Global	Yes
innodb_old_blocks_time [1566]	Yes	Yes	Yes		Global	Yes
innodb_open_files [1567]	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
InnoDB_os_log_fsyncs [602]				Yes	Global	No
InnoDB_os_log_pending_fsyncs [602]				Yes	Global	No
InnoDB_os_log_pending_writes [602]				Yes	Global	No
InnoDB_os_log_written [602]				Yes	Global	No
InnoDB_page_size [602]				Yes	Global	No
InnoDB_pages_created [602]				Yes	Global	No
InnoDB_pages_read [602]				Yes	Global	No
InnoDB_pages_written [602]				Yes	Global	No
innodb_purge_batch_size [1567]	Yes	Yes	Yes		Global	No
innodb_purge_threads [1567]	Yes	Yes	Yes		Global	No
innodb_read_ahead_threshold [1568]	Yes		Yes		Global	Yes
innodb_read_io_threads [1568]	Yes	Yes	Yes		Global	No
innodb_replication_enable [1569]	Yes	Yes	Yes		Global	Yes
innodb_rollback_on_timeout [1569]	Yes	Yes	Yes		Global	No
innodb_rollback_segments [1570]	Yes	Yes	Yes		Global	Yes
InnoDB_row_lock_current_waits [602]				Yes	Global	No
InnoDB_row_lock_time [602]				Yes	Global	No
InnoDB_row_lock_time_avg [602]				Yes	Global	No
InnoDB_row_lock_time_max [602]				Yes	Global	No
InnoDB_row_lock_waits [603]				Yes	Global	No
InnoDB_rows_deleted [603]				Yes	Global	No
InnoDB_rows_inserted [603]				Yes	Global	No
InnoDB_rows_read [603]				Yes	Global	No
InnoDB_rows_updated [603]				Yes	Global	No
innodb_spin_wait_delay [1570]	Yes	Yes	Yes		Global	Yes
innodb_stats_method [1570]	Yes	Yes	Yes		Global	Yes
innodb_stats_on_metadata [1571]	Yes	Yes	Yes		Global	Yes
innodb_stats_sample_pages [1571]	Yes	Yes	Yes		Global	Yes
innodb-status-file [1546]	Yes	Yes				
innodb_strict_mode [1572]	Yes	Yes	Yes		Both	Yes
innodb_support_xa [1572]	Yes	Yes	Yes		Both	Yes
innodb_sync_spin_iters [1573]	Yes	Yes	Yes		Global	Yes
innodb_table_locks [1573]	Yes	Yes	Yes		Both	Yes
innodb_thread_concurrency [1573]	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay [1574]	Yes	Yes	Yes		Global	Yes
InnoDB_truncated_status_writes [603]				Yes	Global	No
innodb_use_native_aio [1574]	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_use_sys_malloc [1575]	Yes	Yes	Yes		Global	No
innodb_version [1575]			Yes		Global	No
innodb_write_io_threads [1575]	Yes	Yes	Yes		Global	No
timed_mutexes [567]	Yes	Yes	Yes		Global	Yes
unique_checks [571]			Yes		Both	Yes

InnoDB Command Options

- [--ignore-builtin-innodb](#) [1546]

Command-Line Format	<code>--ignore-builtin-innodb</code>	
Option-File Format	<code>ignore-builtin-innodb</code>	
Option Sets Variable	Yes, <code>ignore_builtin_innodb</code> [1547]	
Variable Name	<code>ignore-builtin-innodb</code>	
Variable Scope	Global	
Dynamic Variable	No	
Deprecated	5.2.22	
	Permitted Values	
	Type	<code>boolean</code>

In MySQL 5.1, this option caused the server to behave as if the built-in InnoDB were not present, which enabled `InnoDB Plugin` to be used instead. In MySQL 5.5, InnoDB is the default storage engine and `InnoDB Plugin` is not used, so this option has no effect. As of MySQL 5.5.22, it is deprecated and its use results in a warning.

- [--innodb\[=value\]](#) [1546]

Controls loading of the InnoDB storage engine, if the server was compiled with InnoDB support. This option has a tristate format, with possible values of `OFF`, `ON`, or `FORCE`. See [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

To disable InnoDB, use [--innodb=OFF](#) [1546] or [--skip-innodb](#) [1546]. In this case, because the default storage engine is InnoDB, the server will not start unless you also use [--default-storage-engine](#) [441] to set the default to some other engine.

- [--innodb-status-file](#) [1546]

Command-Line Format	<code>--innodb-status-file</code>	
Option-File Format	<code>innodb-status-file</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Controls whether InnoDB creates a file named `innodb_status.<pid>` in the MySQL data directory. If enabled, InnoDB periodically writes the output of `SHOW ENGINE INNODB STATUS` to this file.

By default, the file is not created. To create it, start `mysqld` with the `--innodb-status-file=1` [1546] option. The file is deleted during normal shutdown.

- `--skip-innodb` [1546]

Disable the InnoDB storage engine. See the description of `--innodb` [1546].

InnoDB System Variables

- `ignore_builtin_innodb`

Whether the server was started with the `--ignore-builtin-innodb` [1546] option, which causes the server to behave as if the built-in InnoDB is not present. For more information, see the description of `--ignore-builtin-innodb` [1546] under “InnoDB Command Options” earlier in this section.

- `innodb_adaptive_flushing`

Command-Line Format	<code>--innodb_adaptive_flushing=#</code>	
Option-File Format	<code>innodb_adaptive_flushing</code>	
Option Sets Variable	Yes, <code>innodb_adaptive_flushing</code> [1547]	
Variable Name	<code>innodb_adaptive_flushing</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

Specifies whether to dynamically adjust the rate of flushing [dirty pages](#) in the [InnoDB buffer pool](#) based on the workload. Adjusting the flush rate dynamically is intended to avoid bursts of I/O activity. This setting is enabled by default.

- `innodb_adaptive_hash_index`

Command-Line Format	<code>--innodb_adaptive_hash_index=#</code>	
Option-File Format	<code>innodb_adaptive_hash_index</code>	
Option Sets Variable	Yes, <code>innodb_adaptive_hash_index</code> [1547]	
Variable Name	<code>innodb_adaptive_hash_index</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

Whether the [InnoDB adaptive hash index](#) is enabled or disabled. The adaptive hash index feature is useful for some workloads, and not for others; conduct benchmarks with it both enabled and disabled, using realistic workloads. See [Section 14.3.11.4, “Adaptive Hash Indexes”](#) for details. This variable is enabled by default. Use `--skip-innodb_adaptive_hash_index` at server startup to disable it.

- `innodb_additional_mem_pool_size`

Command-Line Format	<code>--innodb_additional_mem_pool_size=#</code>	
Option-File Format	<code>innodb_additional_mem_pool_size</code>	
Option Sets Variable	Yes, <code>innodb_additional_mem_pool_size</code> [1548]	
Variable Name	<code>innodb_additional_mem_pool_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
Deprecated	5.6.3	
	Permitted Values	
	Type	numeric
	Default	8388608
	Range	2097152 .. 4294967295

The size in bytes of a memory pool InnoDB uses to store [data dictionary](#) information and other internal data structures. The more tables you have in your application, the more memory you need to allocate here. If InnoDB runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log. The default value is 8MB.

- `innodb_autoextend_increment`

Command-Line Format	<code>--innodb_autoextend_increment=#</code>	
Option-File Format	<code>innodb_autoextend_increment</code>	
Option Sets Variable	Yes, <code>innodb_autoextend_increment</code> [1548]	
Variable Name	<code>innodb_autoextend_increment</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	64
	Range	1 .. 1000

The increment size (in MB) for extending the size of an auto-extending shared tablespace file when it becomes full. The default value is 8. This variable does not affect the per-table tablespace files that are created if you use `innodb_file_per_table=1` [1555]. Those files are auto-extending regardless of the value of `innodb_autoextend_increment` [1548]. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

- `innodb_autoinc_lock_mode`

Command-Line Format	<code>--innodb_autoinc_lock_mode=#</code>	
Option-File Format	<code>innodb_autoinc_lock_mode</code>	
Option Sets Variable	Yes, <code>innodb_autoinc_lock_mode</code> [1548]	
Variable Name	<code>innodb_autoinc_lock_mode</code>	
Variable Scope	Global	

Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	1
	Valid Values	0 1 2

The locking mode to use for generating auto-increment values. The permissible values are 0, 1, or 2, for “traditional”, “consecutive”, or “interleaved” lock mode, respectively. [Section 14.3.5.3, “AUTO_INCREMENT Handling in InnoDB”](#), describes the characteristics of these modes.

This variable has a default of 1 (“consecutive” lock mode).

- `innodb_buffer_pool_instances`

Version Introduced	5.5.4
Command-Line Format	<code>--innodb_buffer_pool_instances=#</code>
Option-File Format	<code>innodb_buffer_pool_instances</code>
Option Sets Variable	Yes, <code>innodb_buffer_pool_instances</code> [1549]
Variable Name	<code>innodb_buffer_pool_instances</code>
Variable Scope	Global
Dynamic Variable	No

The number of regions that the InnoDB buffer pool is divided into. For systems with buffer pools in the multi-gigabyte range, dividing the buffer pool into separate instances can improve concurrency, by reducing contention as different threads read and write to cached pages. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pool instances randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool, and is protected by its own buffer pool mutex.

This option takes effect only when you set the `innodb_buffer_pool_size` to a size of 1 gigabyte or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of `innodb_buffer_pool_instances` [\[1549\]](#) and `innodb_buffer_pool_size` [\[1549\]](#) so that each buffer pool instance is at least 1 gigabyte.

- `innodb_buffer_pool_size`

Command-Line Format	<code>--innodb_buffer_pool_size=#</code>
Option-File Format	<code>innodb_buffer_pool_size</code>
Option Sets Variable	Yes, <code>innodb_buffer_pool_size</code> [1549]
Variable Name	<code>innodb_buffer_pool_size</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Platform Bit Size	32
	Type	numeric
	Default	134217728
	Range	1048576 .. 2**32-1

The size in bytes of the memory buffer InnoDB uses to cache data and indexes of its tables. The default value is 128MB. The maximum value depends on the CPU architecture; the maximum is 4294967295 ($2^{32}-1$) on 32-bit systems and 18446744073709551615 ($2^{64}-1$) on 64-bit systems. On 32-bit systems, the CPU architecture and operating system may impose a lower practical maximum size than the stated maximum. When the size of the buffer pool is greater than 1GB, setting `innodb_buffer_pool_instances` [1549] to a value greater than 1 can improve the scalability on a busy server.

The larger you set this value, the less disk I/O is needed to access data in tables. On a dedicated database server, you may set this to up to 80% of the machine physical memory size. Be prepared to scale back this value if these other issues occur:

- Competition for physical memory might cause paging in the operating system.
- InnoDB reserves additional memory for buffers and control structures, so that the total allocated space is approximately 10% greater than the specified size.
- The address space must be contiguous, which can be an issue on Windows systems with DLLs that load at specific addresses.
- The time to initialize the buffer pool is roughly proportional to its size. On large installations, this initialization time may be significant. For example, on a modern Linux x86_64 server, initialization of a 10GB buffer pool takes approximately 6 seconds. See Section 8.9.1, “The InnoDB Buffer Pool”.
- `innodb_change_buffering`

Command-Line Format	<code>--innodb_change_buffering=#</code>	
Option-File Format	<code>innodb_change_buffering</code>	
Option Sets Variable	Yes, <code>innodb_change_buffering</code> [1550]	
Variable Name	<code>innodb_change_buffering</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values (<= 5.5.3)	
	Type	enumeration
	Default	inserts
	Valid Values	inserts none
	Permitted Values (>= 5.5.4)	
	Type	enumeration
	Default	all
	Valid Values	inserts

	deletes
	purges
	changes
	all
	none

Whether InnoDB performs change buffering, an optimization that delays write operations to secondary indexes so that the I/O operations can be performed sequentially. The permitted values are `inserts` (buffer insert operations), `deletes` (buffer delete operations; strictly speaking, the writes that mark index records for later deletion during a purge operation), `changes` (buffer insert and delete-marking operations), `purges` (buffer purge operations, the writes when deleted index entries are finally garbage-collected), `all` (buffer insert, delete-marking, and purge operations) and `none` (do not buffer any operations). The default is `all`. For details, see [Section 14.4.7.4, “Controlling InnoDB Change Buffering”](#).

- `innodb_checksums`

Command-Line Format	<code>--innodb_checksums</code>
Option-File Format	<code>innodb_checksums</code>
Option Sets Variable	Yes, <code>innodb_checksums</code> [1551]
Variable Name	<code>innodb_checksums</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>boolean</code>
	Default <code>ON</code>

InnoDB can use checksum validation on all pages read from the disk to ensure extra fault tolerance against broken hardware or data files. This validation is enabled by default. However, under some rare circumstances (such as when running benchmarks) this extra safety feature is unneeded and can be disabled with `--skip-innodb-checksums` [\[1551\]](#).

- `innodb_commit_concurrency`

Command-Line Format	<code>--innodb_commit_concurrency=#</code>
Option-File Format	<code>innodb_commit_concurrency</code>
Option Sets Variable	Yes, <code>innodb_commit_concurrency</code> [1551]
Variable Name	<code>innodb_commit_concurrency</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>0</code>
	Range <code>0 .. 1000</code>

The number of threads that can commit at the same time. A value of 0 (the default) permits any number of transactions to commit simultaneously.

The value of `innodb_commit_concurrency` [1551] cannot be changed at runtime from zero to nonzero or vice versa. The value can be changed from one nonzero value to another.

- `innodb_concurrency_tickets`

Command-Line Format	<code>--innodb_concurrency_tickets=#</code>
Option-File Format	<code>innodb_concurrency_tickets</code>
Option Sets Variable	Yes, <code>innodb_concurrency_tickets</code> [1552]
Variable Name	<code>innodb_concurrency_tickets</code>
Variable Scope	Global
Dynamic Variable	Yes

The number of threads that can enter InnoDB concurrently is determined by the `innodb_thread_concurrency` [1573] variable. A thread is placed in a queue when it tries to enter InnoDB if the number of threads has already reached the concurrency limit. When a thread is permitted to enter InnoDB, it is given a number of “free tickets” equal to the value of `innodb_concurrency_tickets` [1552], and the thread can enter and leave InnoDB freely until it has used up its tickets. After that point, the thread again becomes subject to the concurrency check (and possible queuing) the next time it tries to enter InnoDB. The default value is 500.

- `innodb_data_file_path`

Command-Line Format	<code>--innodb_data_file_path=name</code>
Option-File Format	<code>innodb_data_file_path</code>
Variable Name	<code>innodb_data_file_path</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>file name</code>

The paths to individual data files and their sizes. The full directory path to each data file is formed by concatenating `innodb_data_home_dir` [1552] to each path specified here. The file sizes are specified in KB, MB, or GB (1024MB) by appending `K`, `M`, or `G` to the size value. The sum of the sizes of the files must be at least 10MB. If you do not specify `innodb_data_file_path` [1552], the default behavior is to create a single 10MB auto-extending data file named `ibdata1`. The size limit of individual files is determined by your operating system. You can set the file size to more than 4GB on those operating systems that support big files. You can also use raw disk partitions as data files. For detailed information on configuring InnoDB tablespace files, see [Section 14.3.2, “Configuring InnoDB”](#).

- `innodb_data_home_dir`

Command-Line Format	<code>--innodb_data_home_dir=path</code>
Option-File Format	<code>innodb_data_home_dir</code>
Option Sets Variable	Yes, <code>innodb_data_home_dir</code> [1552]
Variable Name	<code>innodb_data_home_dir</code>

Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The common part of the directory path for all InnoDB data files in the shared tablespace. This setting does not affect the location of per-file tablespaces when `innodb_file_per_table` [1555] is enabled. The default value is the MySQL data directory. If you specify the value as an empty string, you can use absolute file paths in `innodb_data_file_path` [1552].

- `innodb_doublewrite`

Command-Line Format	<code>--innodb-doublewrite</code>	
Option-File Format	<code>innodb_doublewrite</code>	
Option Sets Variable	Yes, <code>innodb_doublewrite</code> [1553]	
Variable Name	<code>innodb_doublewrite</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean

If this variable is enabled (the default), InnoDB stores all data twice, first to the doublewrite buffer, and then to the actual data files. This variable can be turned off with `--skip-innodb_doublewrite` [1553] for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures.

- `innodb_fast_shutdown`

Command-Line Format	<code>--innodb_fast_shutdown[=#]</code>	
Option-File Format	<code>innodb_fast_shutdown</code>	
Option Sets Variable	Yes, <code>innodb_fast_shutdown</code> [1553]	
Variable Name	<code>innodb_fast_shutdown</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1
	Valid Values	0 1 2

The InnoDB shutdown mode. If the value is 0, InnoDB does a **slow shutdown**, a full **purge** and an insert buffer merge before shutting down. If the value is 1 (the default), InnoDB skips these operations at shutdown, a process known as a **fast shutdown**. If the value is 2, InnoDB flushes its logs and shuts

down cold, as if MySQL had crashed; no committed transactions are lost, but the [crash recovery](#) operation makes the next startup take longer.

The slow shutdown can take minutes, or even hours in extreme cases where substantial amounts of data are still buffered. Use the slow shutdown technique before upgrading or downgrading between MySQL major releases, so that all data files are fully prepared in case the upgrade process updates the file format.

Use `innodb_fast_shutdown=2` in emergency or troubleshooting situations, to get the absolute fastest shutdown if data is at risk of corruption.

- `innodb_file_format`

Command-Line Format	<code>--innodb_file_format=#</code>	
Option-File Format	<code>innodb_file_format</code>	
Option Sets Variable	Yes, <code>innodb_file_format</code> [1554]	
Variable Name	<code>innodb_file_format</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values (>= 5.5.0, <= 5.5.6)	
	Type	<code>string</code>
	Default	<code>Barracuda</code>
	Valid Values	<code>Antelope</code> <code>Barracuda</code>
	Permitted Values (>= 5.5.7)	
	Type	<code>string</code>
	Default	<code>Antelope</code>
	Valid Values	<code>Antelope</code> <code>Barracuda</code>

The file format to use for new InnoDB tables. Currently, `Antelope` and `Barracuda` are supported. This applies only for tables that have their own tablespace, so for it to have an effect, `innodb_file_per_table` [1555] must be enabled. The `Barracuda` file format is required for certain InnoDB features such as table compression.

- `innodb_file_format_check`

Command-Line Format	<code>--innodb_file_format_check=#</code>	
Option-File Format	<code>innodb_file_format_check</code>	
Option Sets Variable	Yes, <code>innodb_file_format_check</code> [1554]	
Variable Name	<code>innodb_file_format_check</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values (<= 5.5.0)	
	Type	<code>string</code>

	Default	Antelope
	Permitted Values (>= 5.5.4)	
	Type	string
	Default	Barracuda
	Permitted Values (>= 5.5.5)	
	Type	boolean
	Default	ON

As of MySQL 5.5.5, this variable can be set to 1 or 0 at server startup to enable or disable whether InnoDB checks the file format tag in the shared tablespace (for example, [Antelope](#) or [Barracuda](#)). If the tag is checked and is higher than that supported by the current version of InnoDB, an error occurs and InnoDB does not start. If the tag is not higher, InnoDB sets the value of `innodb_file_format_max` [1555] to the file format tag.

Before MySQL 5.5.5, this variable can be set to 1 or 0 at server startup to enable or disable whether InnoDB checks the file format tag in the shared tablespace. If the tag is checked and is higher than that supported by the current version of InnoDB, an error occurs and InnoDB does not start. If the tag is not higher, InnoDB sets the value of `innodb_file_format_check` [1554] to the file format tag, which is the value seen at runtime.



Note

Despite the default value sometimes being displayed as `ON` or `OFF`, always use the numeric values 1 or 0 to turn this option on or off in your configuration file or command line.

- `innodb_file_format_max`

Version Introduced	5.5.5	
Command-Line Format	<code>--innodb_file_format_max=#</code>	
Option-File Format	<code>innodb_file_format_max</code>	
Option Sets Variable	Yes, <code>innodb_file_format_max</code> [1555]	
Variable Name	<code>innodb_file_format_max</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	Antelope
	Valid Values	Antelope Barracuda

At server startup, InnoDB sets the value of `innodb_file_format_max` [1555] to the file format tag in the shared tablespace (for example, [Antelope](#) or [Barracuda](#)). If the server creates or opens a table with a “higher” file format, it sets the value of `innodb_file_format_max` [1555] to that format.

This variable was added in MySQL 5.5.5.

- `innodb_file_per_table`

Command-Line Format	<code>--innodb_file_per_table</code>	
Option-File Format	<code>innodb_file_per_table</code>	
Variable Name	<code>innodb_file_per_table</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values (>= 5.5.0, <= 5.5.6)	
	Type	<code>boolean</code>
	Default	<code>ON</code>

If `innodb_file_per_table` [1555] is disabled (the default), InnoDB creates tables in the [system tablespace](#). If `innodb_file_per_table` [1555] is enabled, InnoDB creates each new table using its own `.ibd` file for storing data and indexes, rather than in the system tablespace. See [Section 14.3.3, “Using Per-Table Tablespaces”](#) for information about the features, such as InnoDB table compression, that only work for tables stored in separate tablespaces.

- `innodb_flush_log_at_trx_commit`

Command-Line Format	<code>--innodb_flush_log_at_trx_commit[=#]</code>	
Option-File Format	<code>innodb_flush_log_at_trx_commit</code>	
Option Sets Variable	Yes, <code>innodb_flush_log_at_trx_commit</code> [1556]	
Variable Name	<code>innodb_flush_log_at_trx_commit</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>enumeration</code>
	Default	<code>1</code>
	Valid Values	<code>0</code> <code>1</code> <code>2</code>

If the value of `innodb_flush_log_at_trx_commit` [1556] is 0, the log buffer is written out to the log file once per second and the flush to disk operation is performed on the log file, but nothing is done at a transaction commit. When the value is 1 (the default), the log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file. When the value is 2, the log buffer is written out to the file at each commit, but the flush to disk operation is not performed on it. However, the flushing on the log file takes place once per second also when the value is 2. Note that the once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues.

The default value of 1 is required for full ACID compliance. You can achieve better performance by setting the value different from 1, but then you can lose up to one second worth of transactions in a crash. With a value of 0, any `mysqld` process crash can erase the last second of transactions. With a value of 2, only an operating system crash or a power outage can erase the last second of transactions. InnoDB's [crash recovery](#) works regardless of the value.

For the greatest possible durability and consistency in a replication setup using InnoDB with transactions, use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in your master server `my.cnf` file.



Caution

Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell `mysqld` that the flush has taken place, even though it has not. Then the durability of transactions is not guaranteed even with the setting 1, and in the worst case a power outage can even corrupt the InnoDB database. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try using the Unix command `hdparm` to disable the caching of disk writes in hardware caches, or use some other command specific to the hardware vendor.

- `innodb_flush_method`

Command-Line Format	<code>--innodb_flush_method=name</code>	
Option-File Format	<code>innodb_flush_method</code>	
Option Sets Variable	Yes, <code>innodb_flush_method</code> [1557]	
Variable Name	<code>innodb_flush_method</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type (solaris)	<code>enumeration</code>
	Default	<code>fdatasync</code>
	Valid Values	<code>O_DSYNC</code> <code>O_DIRECT</code>

By default, InnoDB uses the `fsync()` system call to flush both the data and log files. If `innodb_flush_method` [1557] option is set to `O_DSYNC`, InnoDB uses `O_SYNC` to open and flush the log files, and `fsync()` to flush the data files. If `O_DIRECT` is specified (available on some GNU/Linux versions, FreeBSD, and Solaris), InnoDB uses `O_DIRECT` (or `directio()` on Solaris) to open the data files, and uses `fsync()` to flush both the data and log files. Note that InnoDB uses `fsync()` instead of `fdatasync()`, and it does not use `O_DSYNC` by default because there have been problems with it on many varieties of Unix. This variable is relevant only for Unix. On Windows, the flush method is always `async_unbuffered` and cannot be changed.

Depending on hardware configuration, setting `innodb_flush_method` to `O_DIRECT` can either have either a positive or negative effect on performance. Benchmark your particular configuration to decide which setting to use. The mix of read and write operations in your workload can also affect which setting performs better for you. For example, on a system with a hardware RAID controller and battery-backed write cache, `O_DIRECT` can help to avoid double buffering between the InnoDB buffer pool and the operating system's filesystem cache. On some systems where InnoDB data and log files are located on a SAN, the default value or `O_DSYNC` might be faster for a read-heavy workload with mostly `SELECT` statements. Always test this parameter with the same type of hardware and workload that reflects your production environment.

Formerly, a value of `fdatasync` also specified the default behavior. This value was removed, due to confusion that a value of `fdatasync` caused `fsync()` system calls rather than `fdatasync()` for flushing. To obtain the default value now, do not set any value for `innodb_flush_method` [1557] at startup.

- `innodb_force_load_corrupted` [1558]

Version Introduced	5.5.18	
Command-Line Format	<code>--innodb_force_load_corrupted</code>	
Option-File Format	<code>innodb_force_load_corrupted</code>	
Option Sets Variable	Yes, <code>innodb_force_load_corrupted</code> [1558]	
Variable Name	<code>innodb_force_load_corrupted</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Lets InnoDB load tables at startup that are marked as corrupted. Use only during troubleshooting, to recover data that is otherwise inaccessible. When troubleshooting is complete, turn this setting back off and restart the server.

- `innodb_force_recovery`

Command-Line Format	<code>--innodb_force_recovery=#</code>	
Option-File Format	<code>innodb_force_recovery</code>	
Option Sets Variable	Yes, <code>innodb_force_recovery</code> [1558]	
Variable Name	<code>innodb_force_recovery</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>enumeration</code>
	Default	<code>0</code>
	Valid Values	<code>0</code> <code>1</code> <code>2</code> <code>3</code> <code>4</code> <code>5</code> <code>6</code>

The crash recovery mode. Possible values are from 0 to 6. The meanings of these values are described in [Section 14.3.7.2, “Forcing InnoDB Recovery”](#).



Warning

Only set this variable greater than 0 in an emergency situation, to dump your tables from a corrupt database. As a safety measure, InnoDB prevents any changes to its data when this variable is greater than 0. This restriction also prohibits some queries that use `WHERE` or `ORDER BY` clauses, because high values can prevent queries from using indexes.

- `innodb_io_capacity`

Command-Line Format	<code>--innodb_io_capacity=#</code>
Option-File Format	<code>innodb_io_capacity</code>
Option Sets Variable	Yes, <code>innodb_io_capacity</code> [1559]
Variable Name	<code>innodb_io_capacity</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Platform Bit Size 32
	Type <code>numeric</code>
	Default 200
	Range 100 .. 2**32-1
	Permitted Values
	Platform Bit Size 64
	Type <code>numeric</code>
	Default 200
	Range 100 .. 2**64-1

An upper limit on the I/O activity performed by the InnoDB background tasks, such as flushing pages from the [buffer pool](#) and merging data from the [insert buffer](#). The default value is 200. For busy systems capable of higher I/O rates, you can set a higher value at server startup, to help the server handle the background maintenance work associated with a high rate of row changes. For systems with individual 5400 RPM or 7200 RPM drives, you might lower the value to the former default of 100.

This parameter should be set to approximately the number of I/O operations that the system can perform per second. Ideally, keep this setting as low as practical, but not so low that these background activities fall behind. If the value is too high, data is removed from the buffer pool and insert buffer too quickly to provide significant benefit from the caching.

The value represents an estimated proportion of the I/O operations per second (IOPS) available to older-generation disk drives that could perform about 100 IOPS. The current default of 200 reflects that modern storage devices are capable of much higher I/O rates.

In general, you can increase the value as a function of the number of drives used for InnoDB I/O, particularly fast drives capable of high numbers of IOPS. For example, systems that use multiple disks or solid-state disks for InnoDB are likely to benefit from the ability to control this parameter.

Although you can specify a very high number, in practice such large values cease to have any benefit; for example, a value of one million would be considered very high.

You can set the `innodb_io_capacity` value to any number 100 or greater, and the default value is 200. You can set the value of this parameter in the MySQL option file (`my.cnf` or `my.ini`) or change it dynamically with the `SET GLOBAL` command, which requires the `SUPER` privilege.

See [Section 14.4.7.11, “Controlling the Master Thread I/O Rate”](#) for more guidelines about this option. For general information about InnoDB I/O performance, see [Section 8.5.7, “Optimizing InnoDB Disk I/O”](#).

- `innodb_large_prefix`

Version Introduced	5.5.14	
Command-Line Format	<code>--innodb_large_prefix</code>	
Option-File Format	<code>innodb_large_prefix</code>	
Option Sets Variable	Yes, <code>innodb_large_prefix</code> [1560]	
Variable Name	<code>innodb_large_prefix</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Enable this option to allow index key prefixes longer than 767 bytes (up to 3072 bytes), for InnoDB tables that use the `DYNAMIC` and `COMPRESSED` row formats. (Creating such tables also requires the option values `innodb_file_format=barracuda` [\[1554\]](#) and `innodb_file_per_table=true` [\[1555\]](#).) See [Section 14.3.15, “Limits on InnoDB Tables”](#) for the relevant maximums associated with index key prefixes under various settings.

For tables using the `REDUNDANT` and `COMPACT` row formats, this option does not affect the allowed key prefix length. It does introduce a new error possibility. When this setting is enabled, attempting to create an index prefix with a key length greater than 3072 for a `REDUNDANT` or `COMPACT` table causes an error `ER_INDEX_COLUMN_TOO_LONG` (1727).

- `innodb_lock_wait_timeout`

Command-Line Format	<code>--innodb_lock_wait_timeout=#</code>	
Option-File Format	<code>innodb_lock_wait_timeout</code>	
Option Sets Variable	Yes, <code>innodb_lock_wait_timeout</code> [1560]	
Variable Name	<code>innodb_lock_wait_timeout</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	

	Type	numeric
	Default	50
	Range	1 .. 1073741824

The timeout in seconds an InnoDB transaction waits for a row lock before giving up. The default value is 50 seconds. A transaction that tries to access a row that is locked by another InnoDB transaction waits at most this many seconds for write access to the row before issuing the following error:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

When a lock wait timeout occurs, the current statement is rolled back (not the entire transaction). To have the entire transaction roll back, start the server with the `--innodb_rollback_on_timeout` [1569] option. See also [Section 14.3.13, “InnoDB Error Handling”](#).

You might decrease this value for highly interactive applications or OLTP systems, to display user feedback quickly or put the update into a queue for processing later. You might increase this value for long-running back-end operations, such as a transform step in a data warehouse that waits for other large insert or update operations to finish.

`innodb_lock_wait_timeout` [1560] applies to InnoDB row locks only. A MySQL table lock does not happen inside InnoDB and this timeout does not apply to waits for table locks.

The lock wait timeout value does not apply to deadlocks, because InnoDB detects them immediately and rolls back one of the deadlocked transactions.

- `innodb_locks_unsafe_for_binlog`

Command-Line Format	<code>--innodb_locks_unsafe_for_binlog</code>	
Option-File Format	<code>innodb_locks_unsafe_for_binlog</code>	
Option Sets Variable	Yes, <code>innodb_locks_unsafe_for_binlog</code> [1561]	
Variable Name	<code>innodb_locks_unsafe_for_binlog</code>	
Variable Scope	Global	
Dynamic Variable	No	
Deprecated	5.6.3	
	Permitted Values	
	Type	boolean
	Default	OFF

This variable affects how InnoDB uses gap locking for searches and index scans. Normally, InnoDB uses an algorithm called *next-key locking* that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record *R* in an index, another session cannot insert a new index record in the gap immediately before *R* in the index order. See [Section 14.3.9.4, “InnoDB Record, Gap, and Next-Key Locks”](#).

By default, the value of `innodb_locks_unsafe_for_binlog` [1561] is 0 (disabled), which means that gap locking is enabled: InnoDB uses next-key locks for searches and index scans. To enable the

variable, set it to 1. This causes gap locking to be disabled: InnoDB uses only index-record locks for searches and index scans.

Enabling `innodb_locks_unsafe_for_binlog` [1561] does not disable the use of gap locking for foreign-key constraint checking or duplicate-key checking.

The effect of enabling `innodb_locks_unsafe_for_binlog` [1561] is similar to but not identical to setting the transaction isolation level to `READ COMMITTED` [1402]:

- Enabling `innodb_locks_unsafe_for_binlog` [1561] is a global setting and affects all sessions, whereas the isolation level can be set globally for all sessions, or individually per session.
- `innodb_locks_unsafe_for_binlog` [1561] can be set only at server startup, whereas the isolation level can be set at startup or changed at runtime.

`READ COMMITTED` [1402] therefore offers finer and more flexible control than `innodb_locks_unsafe_for_binlog` [1561]. For additional details about the effect of isolation level on gap locking, see [Section 13.3.6, “SET TRANSACTION Syntax”](#).

Enabling `innodb_locks_unsafe_for_binlog` [1561] may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled. Suppose that there is an index on the `id` column of the `child` table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where `id` is greater than 100. If the locks set on the index records in that range do not lock out inserts made in the gaps, another session can insert a new row into the table. Consequently, if you were to execute the same `SELECT` again within the same transaction, you would see a new row in the result set returned by the query. This also means that if new items are added to the database, InnoDB does not guarantee serializability. Therefore, if `innodb_locks_unsafe_for_binlog` [1561] is enabled, InnoDB guarantees at most an isolation level of `READ COMMITTED` [1402]. (Conflict serializability is still guaranteed.) For additional information about phantoms, see [Section 14.3.9.5, “Avoiding the Phantom Problem Using Next-Key Locking”](#).

Enabling `innodb_locks_unsafe_for_binlog` [1561] has additional effects:

- For `UPDATE` or `DELETE` statements, InnoDB holds locks only for rows that it updates or deletes. Record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition. This greatly reduces the probability of deadlocks, but they can still happen.
- For `UPDATE` statements, if a row is already locked, InnoDB performs a “semi-consistent” read, returning the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`. If the row matches (must be updated), MySQL reads the row again and this time InnoDB either locks it or waits for a lock on it.

Consider the following example, beginning with this table:

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2), (2,3), (3,2), (4,3), (5,2);
COMMIT;
```

In this case, table has no indexes, so searches and index scans use the hidden clustered index for record locking (see [Section 14.3.11.1, “Clustered and Secondary Indexes”](#)).

Suppose that one client performs an `UPDATE` using these statements:

```
SET autocommit = 0;
UPDATE t SET b = 5 WHERE b = 3;
```

Suppose also that a second client performs an `UPDATE` by executing these statements following those of the first client:

```
SET autocommit = 0;
UPDATE t SET b = 4 WHERE b = 2;
```

As `InnoDB` executes each `UPDATE`, it first acquires an exclusive lock for each row, and then determines whether to modify it. If `InnoDB` does not modify the row and `innodb_locks_unsafe_for_binlog` [1561] is enabled, it releases the lock. Otherwise, `InnoDB` retains the lock until the end of the transaction. This affects transaction processing as follows.

If `innodb_locks_unsafe_for_binlog` [1561] is disabled, the first `UPDATE` acquires x-locks and does not release any of them:

```
x-lock(1,2); retain x-lock
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); retain x-lock
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); retain x-lock
```

The second `UPDATE` blocks as soon as it tries to acquire any locks (because first update has retained locks on all rows), and does not proceed until the first `UPDATE` commits or rolls back:

```
x-lock(1,2); block and wait for first UPDATE to commit or roll back
```

If `innodb_locks_unsafe_for_binlog` [1561] is enabled, the first `UPDATE` acquires x-locks and releases those for rows that it does not modify:

```
x-lock(1,2); unlock(1,2)
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); unlock(3,2)
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); unlock(5,2)
```

For the second `UPDATE`, `InnoDB` does a “semi-consistent” read, returning the latest committed version of each row to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`:

```
x-lock(1,2); update(1,2) to (1,4); retain x-lock
x-lock(2,3); unlock(2,3)
x-lock(3,2); update(3,2) to (3,4); retain x-lock
x-lock(4,3); unlock(4,3)
x-lock(5,2); update(5,2) to (5,4); retain x-lock
```

- `innodb_log_buffer_size`

Command-Line Format	<code>--innodb_log_buffer_size=#</code>
Option-File Format	<code>innodb_log_buffer_size</code>
Option Sets Variable	Yes, <code>innodb_log_buffer_size</code> [1563]
Variable Name	<code>innodb_log_buffer_size</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	numeric
	Default	8388608
	Range	262144 .. 4294967295

The size in bytes of the buffer that InnoDB uses to write to the log files on disk. The default value is 8MB. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have big transactions, making the log buffer larger saves disk I/O.

- `innodb_log_file_size`

Command-Line Format	<code>--innodb_log_file_size=#</code>	
Option-File Format	<code>innodb_log_file_size</code>	
Option Sets Variable	Yes, <code>innodb_log_file_size</code> [1564]	
Variable Name	<code>innodb_log_file_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	5242880
	Range	108576 .. 4294967295

The size in bytes of each log file in a log group. The combined size of log files must be less than 4GB. The default value is 5MB. Sensible values range from 1MB to $1/N$ -th of the size of the buffer pool, where N is the number of log files in the group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But larger log files also mean that recovery is slower in case of a crash.

- `innodb_log_files_in_group`

Command-Line Format	<code>--innodb_log_files_in_group=#</code>	
Option-File Format	<code>innodb_log_files_in_group</code>	
Option Sets Variable	Yes, <code>innodb_log_files_in_group</code> [1564]	
Variable Name	<code>innodb_log_files_in_group</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	2
	Range	2 .. 100

The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default (and recommended) value is 2.

- `innodb_log_group_home_dir`

Command-Line Format	<code>--innodb_log_group_home_dir=path</code>
----------------------------	---

Option-File Format	<code>innodb_log_group_home_dir</code>	
Option Sets Variable	Yes, <code>innodb_log_group_home_dir</code> [1564]	
Variable Name	<code>innodb_log_group_home_dir</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The directory path to the InnoDB redo log files. If you do not specify any InnoDB log variables, the default is to create two 5MB files named `ib_logfile0` and `ib_logfile1` in the MySQL data directory.

- `innodb_max_dirty_pages_pct`

Command-Line Format	<code>--innodb_max_dirty_pages_pct=#</code>	
Option-File Format	<code>innodb_max_dirty_pages_pct</code>	
Option Sets Variable	Yes, <code>innodb_max_dirty_pages_pct</code> [1565]	
Variable Name	<code>innodb_max_dirty_pages_pct</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>75</code>
	Range	<code>0 .. 99</code>

This is an integer in the range from 0 to 99. The default value is 75. The main thread in InnoDB tries to write pages from the buffer pool so that the percentage of dirty (not yet written) pages will not exceed this value.

- `innodb_max_purge_lag`

Command-Line Format	<code>--innodb_max_purge_lag=#</code>	
Option-File Format	<code>innodb_max_purge_lag</code>	
Option Sets Variable	Yes, <code>innodb_max_purge_lag</code> [1565]	
Variable Name	<code>innodb_max_purge_lag</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 4294967295</code>

This variable controls how to delay `INSERT`, `UPDATE`, and `DELETE` operations when purge operations are lagging (see Section 14.3.10, “InnoDB Multi-Versioning”). The default value 0 (no delays).

The InnoDB transaction system maintains a list of transactions that have index records delete-marked by UPDATE or DELETE operations. Let the length of this list be *purge_lag*. When *purge_lag* exceeds *innodb_max_purge_lag* [1565], each INSERT, UPDATE, and DELETE operation is delayed by $((\text{purge_lag}/\text{innodb_max_purge_lag} [1565]) \times 10) - 5$ milliseconds. The delay is computed in the beginning of a purge batch, every ten seconds. The operations are not delayed if purge cannot run because of an old [consistent read](#) view that could see the rows to be purged.

A typical setting for a problematic workload might be 1 million, assuming that transactions are small, only 100 bytes in size, and it is permissible to have 100MB of unpurged InnoDB table rows.

The lag value is displayed as the history list length in the [TRANSACTIONS](#) section of InnoDB Monitor output. For example, if the output includes the following lines, the lag value is 20:

```
-----
TRANSACTIONS
-----
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
```

- [innodb_mirrored_log_groups](#)

Has no effect.

- [innodb_old_blocks_pct](#)

Command-Line Format	<code>--innodb_old_blocks_pct=#</code>	
Option-File Format	<code>innodb_old_blocks_pct</code>	
Variable Name	<code>innodb_old_blocks_pct</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>37</code>
	Range	<code>5 .. 95</code>

Specifies the approximate percentage of the InnoDB [buffer pool](#) used for the old block [sublist](#). The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool). See [Section 8.9.1, “The InnoDB Buffer Pool”](#) for information about buffer pool management, such as the [LRU](#) algorithm and [eviction](#) policies.

- [innodb_old_blocks_time](#)

Command-Line Format	<code>--innodb_old_blocks_time=#</code>	
Option-File Format	<code>innodb_old_blocks_time</code>	
Variable Name	<code>innodb_old_blocks_time</code>	
Variable Scope	Global	
Dynamic Variable	Yes	

Non-zero values protect against the [buffer pool](#) being filled up by data that is referenced only for a brief period, such as during a [full table scan](#). Increasing this value offers more protection against full table scans interfering with data cached in the buffer pool.

Specifies how long in milliseconds (ms) a block inserted into the old [sublist](#) must stay there after its first access before it can be moved to the new sublist. If the value is 0, a block inserted into the old sublist moves immediately to the new sublist the first time it is accessed, no matter how soon after insertion the access occurs. If the value is greater than 0, blocks remain in the old sublist until an access occurs at least that many ms after the first access. For example, a value of 1000 causes blocks to stay in the old sublist for 1 second after the first access before they become eligible to move to the new sublist.

This variable is often used in combination with [innodb_old_blocks_pct](#) [1566]. See [Section 8.9.1, “The InnoDB Buffer Pool”](#) for information about buffer pool management, such as the [LRU](#) algorithm and [eviction](#) policies.

- [innodb_open_files](#)

Command-Line Format	<code>--innodb_open_files=#</code>
Option-File Format	<code>innodb_open_files</code>
Variable Name	<code>innodb_open_files</code>
Variable Scope	Global
Dynamic Variable	No

This variable is relevant only if you use multiple [InnoDB tablespaces](#). It specifies the maximum number of [.ibd files](#) that MySQL can keep open at one time. The minimum value is 10. The default value is 300.

The file descriptors used for [.ibd](#) files are for [InnoDB](#) tables only. They are independent of those specified by the `--open-files-limit` [456] server option, and do not affect the operation of the table cache.

- [innodb_purge_batch_size](#)

Version Introduced	5.5.4	
Command-Line Format	<code>--innodb_purge_batch_size=#</code>	
Option-File Format	<code>innodb_purge_batch_size</code>	
Variable Name	<code>innodb_purge_batch_size</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values (>= 5.5.4)	
	Type	numeric
	Default	20
	Range	1 .. 5000

The granularity of changes, expressed in units of [redo log](#) records, that trigger a [purge](#) operation, flushing the changed [buffer pool](#) blocks to disk. The default value is 20, and the range is 1-5000. This option is intended for tuning performance in combination with the setting `innodb_purge_threads=1`, and typical users do not need to modify it.

- [innodb_purge_threads](#)

Version Introduced	5.5.4	
Command-Line Format	<code>--innodb_purge_threads=#</code>	
Option-File Format	<code>innodb_purge_threads</code>	
Variable Name	<code>innodb_purge_threads</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values (>= 5.5.4)	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 1</code>

The number of background threads devoted to the InnoDB [purge](#) operation. Currently, can only be 0 (the default) or 1. The default value of 0 signifies that the purge operation is performed as part of the [master thread](#). Running the purge operation in its own thread can reduce internal contention within InnoDB, improving scalability. Currently, the performance gain might be minimal because the background thread might encounter different kinds of contention than before. This feature primarily lays the groundwork for future performance work.

- `innodb_read_ahead_threshold`

Command-Line Format	<code>--innodb_read_ahead_threshold=#</code>	
Option-File Format	<code>innodb_read_ahead_threshold</code>	
Option Sets Variable	Yes, <code>innodb_read_ahead_threshold</code> [1568]	
Variable Name	<code>innodb_read_ahead_threshold</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>56</code>
	Range	<code>0 .. 64</code>

Controls the sensitivity of linear [read-ahead](#) that InnoDB uses to prefetch pages into the [buffer pool](#). If InnoDB reads at least `innodb_read_ahead_threshold` [\[1568\]](#) pages sequentially from an [extent](#) (64 pages), it initiates an asynchronous read for the entire following extent. The permissible range of values is 0 to 64. The default is 56: InnoDB must read at least 56 pages sequentially from an extent to initiate an asynchronous read for the following extent.

- `innodb_read_io_threads`

Command-Line Format	<code>--innodb_read_io_threads=#</code>	
Option-File Format	<code>innodb_read_io_threads</code>	
Option Sets Variable	Yes, <code>innodb_read_io_threads</code> [1568]	
Variable Name	<code>innodb_read_io_threads</code>	
Variable Scope	Global	

Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	4
	Range	1 .. 64

The number of I/O threads for read operations in InnoDB. The default value is 4.



Note

On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for `innodb_read_io_threads`, `innodb_write_io_threads` [1575], and the Linux `aio-max-nr` setting can exceed system limits. Ideally, increase the `aio-max-nr` setting; as a workaround, you might reduce the settings for one or both of the MySQL configuration options.

- `innodb_replication_delay`

Command-Line Format	<code>--innodb_replication_delay=#</code>	
Option-File Format	<code>innodb_replication_delay</code>	
Option Sets Variable	Yes, <code>innodb_replication_delay</code> [1569]	
Variable Name	<code>innodb_replication_delay</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 4294967295

The replication thread delay (in ms) on a slave server if `innodb_thread_concurrency` [1573] is reached.

- `innodb_rollback_on_timeout`

Command-Line Format	<code>--innodb_rollback_on_timeout</code>	
Option-File Format	<code>innodb_rollback_on_timeout</code>	
Option Sets Variable	Yes, <code>innodb_rollback_on_timeout</code> [1569]	
Variable Name	<code>innodb_rollback_on_timeout</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

In MySQL 5.5, InnoDB **rolls back** only the last statement on a transaction timeout by default. If `--innodb_rollback_on_timeout [1569]` is specified, a transaction timeout causes InnoDB to abort and roll back the entire transaction (the same behavior as in MySQL 4.1).

- `innodb_rollback_segments`

Version Introduced	5.5.11	
Command-Line Format	<code>--innodb_rollback_segments=#</code>	
Option-File Format	<code>innodb_rollback_segments</code>	
Option Sets Variable	Yes, <code>innodb_rollback_segments</code> [1570]	
Variable Name	<code>innodb_rollback_segments</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>128</code>
	Range	<code>1 .. 128</code>

Defines how many of the **rollback segments** in the system tablespace that InnoDB uses within a **transaction**. You might reduce this value from its default of 128 if a smaller number of rollback segments performs better for your workload.

- `innodb_spin_wait_delay`

Command-Line Format	<code>--innodb_spin_wait_delay=#</code>	
Option-File Format	<code>innodb_spin_wait_delay</code>	
Option Sets Variable	Yes, <code>innodb_spin_wait_delay</code> [1570]	
Variable Name	<code>innodb_spin_wait_delay</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>6</code>
	Range	<code>0 .. 4294967295</code>

The maximum delay between polls for a **spin** lock. The default value is 6.

- `innodb_stats_method`

Version Introduced	5.5.10	
Command-Line Format	<code>--innodb_stats_method=name</code>	
Option-File Format	<code>innodb_stats_method</code>	
Option Sets Variable	Yes, <code>innodb_stats_method</code> [1570]	
Variable Name	<code>innodb_stats_method</code>	

Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	nulls_equal
	Valid Values	nulls_equal nulls_unequal nulls_ignored

How the server treats `NULL` values when collecting [statistics](#) about the distribution of index values for InnoDB tables. This variable has three possible values, `nulls_equal`, `nulls_unequal`, and `nulls_ignored`. For `nulls_equal`, all `NULL` index values are considered equal and form a single value group that has a size equal to the number of `NULL` values. For `nulls_unequal`, `NULL` values are considered unequal, and each `NULL` forms a distinct value group of size 1. For `nulls_ignored`, `NULL` values are ignored.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in [Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”](#).

- `innodb_stats_on_metadata`

Version Introduced	5.5.4
Command-Line Format	<code>--innodb_stats_on_metadata</code>
Option-File Format	<code>innodb_stats_on_metadata</code>
Option Sets Variable	Yes, <code>innodb_stats_on_metadata</code> [1571]
Variable Name	<code>innodb_stats_on_metadata</code>
Variable Scope	Global
Dynamic Variable	Yes

When this variable is enabled (which is the default, as before the variable was created), InnoDB updates [statistics](#) during metadata statements such as `SHOW TABLE STATUS` or `SHOW INDEX`, or when accessing the `INFORMATION_SCHEMA` tables `TABLES` or `STATISTICS`. (These updates are similar to what happens for `ANALYZE TABLE`.) When disabled, InnoDB does not update statistics during these operations. Disabling this variable can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of [execution plans](#) for queries that involve InnoDB tables.

- `innodb_stats_sample_pages`

Command-Line Format	<code>--innodb_stats_sample_pages=#</code>
Option-File Format	<code>innodb_stats_sample_pages</code>
Option Sets Variable	Yes, <code>innodb_stats_sample_pages</code> [1571]
Variable Name	<code>innodb_stats_sample_pages</code>
Variable Scope	Global
Dynamic Variable	Yes
Deprecated	5.6.3

	Permitted Values	
Type	numeric	
Default	8	
Range	1 .. 2**64-1	

The number of index pages to sample for index distribution [statistics](#) such as are calculated by [ANALYZE TABLE](#). The default value is 8. For more information, see [Section 14.4.8, “Changes for Flexibility, Ease of Use and Reliability”](#).

- `innodb_strict_mode`

Command-Line Format	<code>--innodb_strict_mode=#</code>	
Option-File Format	<code>innodb_strict_mode</code>	
Option Sets Variable	Yes, <code>innodb_strict_mode</code> [1572]	
Variable Name	<code>innodb_strict_mode</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
Type	boolean	
Default	OFF	

Whether [InnoDB](#) returns errors rather than warnings for certain conditions. This is analogous to strict SQL mode. The default value is `OFF`. See [Section 14.4.8.4, “InnoDB Strict Mode”](#) for a list of the conditions that are affected.

- `innodb_support_xa`

Command-Line Format	<code>--innodb_support_xa</code>	
Option-File Format	<code>innodb_support_xa</code>	
Option Sets Variable	Yes, <code>innodb_support_xa</code> [1572]	
Variable Name	<code>innodb_support_xa</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
Type	boolean	
Default	TRUE	

Enables [InnoDB](#) support for two-phase commit in [XA](#) transactions, causing an extra disk flush for transaction preparation. This setting is the default. The XA mechanism is used internally and is essential for any server that has its binary log turned on and is accepting changes to its data from more than one thread. If you turn it off, transactions can be written to the binary log in a different order from the one in which the live database is committing them. This can produce different data when the binary log is replayed in disaster recovery or on a replication slave. Do not turn it off on a replication master server unless you have an unusual setup where only one thread is able to change data.

For a server that is accepting data changes from only one thread, it is safe and recommended to turn off this option to improve performance for InnoDB tables. For example, you can turn it off on replication slaves where only the replication SQL thread is changing data.

You can also turn off this option if you do not need it for safe binary logging or replication, and you also do not use an external XA transaction manager.

- `innodb_sync_spin_loops`

Command-Line Format	<code>--innodb_sync_spin_loops=#</code>	
Option-File Format	<code>innodb_sync_spin_loops</code>	
Option Sets Variable	Yes, <code>innodb_sync_spin_loops</code> [1573]	
Variable Name	<code>innodb_sync_spin_loops</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>30</code>
	Range	<code>0 .. 4294967295</code>

The number of times a thread waits for an InnoDB mutex to be freed before the thread is suspended. The default value is 30.

- `innodb_table_locks`

Command-Line Format	<code>--innodb_table_locks</code>	
Option-File Format	<code>innodb_table_locks</code>	
Option Sets Variable	Yes, <code>innodb_table_locks</code> [1573]	
Variable Name	<code>innodb_table_locks</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>TRUE</code>

If `autocommit = 0` [484], InnoDB honors `LOCK TABLES`; MySQL does not return from `LOCK TABLES ... WRITE` until all other threads have released all their locks to the table. The default value of `innodb_table_locks` [1573] is 1, which means that `LOCK TABLES` causes InnoDB to lock a table internally if `autocommit = 0` [484].

As of MySQL 5.5.3, `innodb_table_locks = 0` [1573] has no effect for tables locked explicitly with `LOCK TABLES ... WRITE`. It still has an effect for tables locked for read or write by `LOCK TABLES ... WRITE` implicitly (for example, through triggers) or by `LOCK TABLES ... READ`.

- `innodb_thread_concurrency`

Command-Line Format	<code>--innodb_thread_concurrency=#</code>
----------------------------	--

Option-File Format	<code>innodb_thread_concurrency</code>	
Option Sets Variable	Yes, <code>innodb_thread_concurrency</code> [1573]	
Variable Name	<code>innodb_thread_concurrency</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 1000</code>

InnoDB tries to keep the number of operating system threads concurrently inside InnoDB less than or equal to the limit given by this variable. Once the number of threads reaches this limit, additional threads are placed into a wait state within a FIFO queue for execution. Threads waiting for locks are not counted in the number of concurrently executing threads.

The correct value for this variable is dependent on environment and workload. Try a range of different values to determine what value works for your applications. A recommended value is 2 times the number of CPUs plus the number of disks.

The range of this variable is 0 to 1000. A value of 0 (the default) is interpreted as infinite concurrency (no concurrency checking). Disabling thread concurrency checking enables InnoDB to create as many threads as it needs. A value of 0 also disables the `queries inside InnoDB` and `queries in queue counters` in the `ROW OPERATIONS` section of `SHOW ENGINE INNODB STATUS` output.

- `innodb_thread_sleep_delay`

Command-Line Format	<code>--innodb_thread_sleep_delay=#</code>	
Option-File Format	<code>innodb_thread_sleep_delay</code>	
Option Sets Variable	Yes, <code>innodb_thread_sleep_delay</code> [1574]	
Variable Name	<code>innodb_thread_sleep_delay</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>10000</code>

How long InnoDB threads sleep before joining the InnoDB queue, in microseconds. The default value is 10,000. A value of 0 disables sleep.

- `innodb_use_native_aio`

Version Introduced	<code>5.5.4</code>
Command-Line Format	<code>--innodb_use_native_aio=#</code>
Option-File Format	<code>innodb_use_native_aio</code>
Option Sets Variable	Yes, <code>innodb_use_native_aio</code> [1574]
Variable Name	<code>innodb_use_native_aio</code>

Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	ON

Specifies whether to use the Linux asynchronous I/O subsystem. This variable applies to Linux systems only, and cannot be changed while the server is running.

Normally, you do not need to touch this option, because it is enabled by default. If a problem with the asynchronous I/O subsystem in the OS prevents InnoDB from starting, start the server with this variable disabled (use `innodb_use_native_aio=0` [1574] in the option file). This option could also be turned off automatically during startup, if InnoDB detects a potential problem such as a combination of `tmpdir` location, `tmpfs` filesystem, and Linux kernel that does not support AIO on `tmpfs`.

This variable was added in MySQL 5.5.4.

- `innodb_use_sys_malloc`

Command-Line Format	<code>--innodb_use_sys_malloc=#</code>	
Option-File Format	<code>innodb_use_sys_malloc</code>	
Option Sets Variable	Yes, <code>innodb_use_sys_malloc</code> [1575]	
Variable Name	<code>innodb_use_sys_malloc</code>	
Variable Scope	Global	
Dynamic Variable	No	
Deprecated	5.6.3	
	Permitted Values	
	Type	boolean
	Default	ON

Whether InnoDB uses the operating system memory allocator (ON) or its own (OFF). The default value is ON.

- `innodb_version`

The InnoDB version number. Starting in 5.5.30, the separate numbering for InnoDB is discontinued and this value is the same as for the `version` [572] variable.

- `innodb_write_io_threads`

Command-Line Format	<code>--innodb_write_io_threads=#</code>	
Option-File Format	<code>innodb_write_io_threads</code>	
Option Sets Variable	Yes, <code>innodb_write_io_threads</code> [1575]	
Variable Name	<code>innodb_write_io_threads</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	

	Type	numeric
	Default	4
	Range	1 .. 64

The number of I/O threads for write operations in InnoDB. The default value is 4.



Note

On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for `innodb_read_io_threads`, `innodb_write_io_threads` [1575], and the Linux `aio-max-nr` setting can exceed system limits. Ideally, increase the `aio-max-nr` setting; as a workaround, you might reduce the settings for one or both of the MySQL configuration options.

- `sync_binlog` [1904]

Command-Line Format	<code>--sync-binlog=#</code>
Option-File Format	<code>sync_binlog</code>
Option Sets Variable	Yes, <code>sync_binlog</code> [1904]
Variable Name	<code>sync_binlog</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Platform Bit Size 32
	Type numeric
	Default 0
	Range 0 .. 4294967295
	Permitted Values
	Platform Bit Size 64
	Type numeric
	Default 0
	Range 0 .. 18446744073709547520

If the value of this variable is greater than 0, the MySQL server synchronizes its binary log to disk (using `fdatasync()`) after every `sync_binlog` [1904] writes to the binary log. There is one write to the binary log per statement if autocommit is enabled, and one write per transaction otherwise. The default value of `sync_binlog` [1904] is 0, which does no synchronizing to disk. A value of 1 is the safest choice, because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

14.3.5. Creating and Using InnoDB Tables

To create an InnoDB table, specify an `ENGINE=InnoDB` option in the `CREATE TABLE` statement:

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;
```

The statement creates a table and an index on column `a` in the `InnoDB` tablespace that consists of the data files that you specified in `my.cnf`. In addition, MySQL creates a file `customers.frm` in the `test` directory under the MySQL database directory. Internally, `InnoDB` adds an entry for the table to its own data dictionary. The entry includes the database name. For example, if `test` is the database in which the `customers` table is created, the entry is for `'test/customers'`. This means you can create a table of the same name `customers` in some other database, and the table names do not collide inside `InnoDB`.

You can query the amount of free space in the `InnoDB` tablespace by issuing a `SHOW TABLE STATUS` statement for any `InnoDB` table. The amount of free space in the tablespace appears in the `Data_free` section in the output of `SHOW TABLE STATUS`. For example:

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

The `statistics SHOW` displays for `InnoDB` tables are only approximate. They are used in SQL optimization. Table and index reserved sizes in bytes are accurate, though.

14.3.5.1. Using InnoDB Transactions

Transactions in SQL

By default, each client that connects to the MySQL server begins with `autocommit` mode enabled, which automatically commits every SQL statement as you execute it. To use multiple-statement transactions, you can switch `autocommit` off with the SQL statement `SET autocommit = 0` and end each transaction with either `COMMIT` or `ROLLBACK`. If you want to leave `autocommit` on, you can begin your transactions within `START TRANSACTION` and end them with `COMMIT` or `ROLLBACK`. The following example shows two transactions. The first is committed; the second is rolled back.

```
shell> mysql test

mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (a))
  -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM customer;
+-----+-----+
| a     | b       |
+-----+-----+
|    10 | Heikki  |
+-----+-----+
1 row in set (0.00 sec)
mysql>
```

Transactions in Client-Side Languages

In APIs such as PHP, Perl DBI, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as `COMMIT` to the MySQL server as strings just like any other SQL statements such as `SELECT` or `INSERT`. Some APIs also offer separate special transaction commit and rollback functions or methods.

14.3.5.2. Converting Tables from MyISAM to InnoDB

To convert a non-InnoDB table to use InnoDB use `ALTER TABLE`:

```
ALTER TABLE table_name ENGINE=InnoDB;
```



Important

Do not convert MySQL system tables in the `mysql` database (such as `user` or `host`) to the InnoDB type. This is an unsupported operation. The system tables must always be of the MyISAM type.

To make an InnoDB table that is a clone of a MyISAM table:

- Create an empty InnoDB table with identical definitions.
- Create the appropriate indexes.
- Insert the rows with `INSERT INTO innodb_table SELECT * FROM myisam_table`.

You can also create the indexes after inserting the data. Historically, creating new secondary indexes was a slow operation for InnoDB, but this is no longer the case.

If you have `UNIQUE` constraints on secondary keys, you can speed up a table import by turning off the uniqueness checks temporarily during the import operation:

```
SET unique_checks=0;... import operation ...
SET unique_checks=1;
```

For big tables, this saves disk I/O because InnoDB can use its `insert buffer` to write secondary index records as a batch. Be certain that the data contains no duplicate keys. `unique_checks` [571] permits but does not require storage engines to ignore duplicate keys.

To get better control over the insertion process, you might insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records have been inserted, you can rename the tables.

During the conversion of big tables, increase the size of the InnoDB buffer pool to reduce disk I/O, to a maximum of 80% of physical memory. You can also increase the sizes of the InnoDB log files.

Make sure that you do not fill up the tablespace: InnoDB tables require a lot more disk space than MyISAM tables. If an `ALTER TABLE` operation runs out of space, it starts a rollback, and that can take hours if it is disk-bound. For inserts, InnoDB uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. For rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for millions of disk I/O operations to complete. For the complete procedure, see [Section 14.3.7.2, “Forcing InnoDB Recovery”](#).

If you want all new user-created tables to use the InnoDB storage engine, add the line `default-storage-engine=innodb` to the `[mysqld]` section of your server option file.

14.3.5.3. AUTO_INCREMENT Handling in InnoDB

InnoDB provides an optimization that significantly improves scalability and performance of SQL statements that insert rows into tables with `AUTO_INCREMENT` columns. To use the `AUTO_INCREMENT` mechanism with an InnoDB table, an `AUTO_INCREMENT` column `ai_col` must be defined as part of an index such that it is possible to perform the equivalent of an indexed `SELECT MAX(ai_col)` lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.

This section provides background information on the original (“traditional”) implementation of auto-increment locking in InnoDB, explains the configurable locking mechanism, documents the parameter for configuring the mechanism, and describes its behavior and interaction with replication.

14.3.5.3.1. Traditional InnoDB Auto-Increment Locking

The original implementation of auto-increment handling in InnoDB uses the following strategy to prevent problems when using the binary log for statement-based replication or for certain recovery scenarios.

If you specify an `AUTO_INCREMENT` column for an InnoDB table, the table handle in the InnoDB data dictionary contains a special counter called the auto-increment counter that is used in assigning new values for the column. This counter is stored only in main memory, not on disk.

InnoDB uses the following algorithm to initialize the auto-increment counter for a table `t` that contains an `AUTO_INCREMENT` column named `ai_col`: After a server startup, for the first insert into a table `t`, InnoDB executes the equivalent of this statement:

```
SELECT MAX(ai_col) FROM t FOR UPDATE;
```

InnoDB increments the value retrieved by the statement and assigns it to the column and to the auto-increment counter for the table. By default, the value is incremented by one. This default can be overridden by the `auto_increment_increment` [1865] configuration setting.

If the table is empty, InnoDB uses the value 1. This default can be overridden by the `auto_increment_offset` [1868] configuration setting.

If a `SHOW TABLE STATUS` statement examines the table `t` before the auto-increment counter is initialized, InnoDB initializes but does not increment the value and stores it for use by later inserts. This initialization uses a normal exclusive-locking read on the table and the lock lasts to the end of the transaction.

InnoDB follows the same procedure for initializing the auto-increment counter for a freshly created table.

After the auto-increment counter has been initialized, if you do not explicitly specify a value for an `AUTO_INCREMENT` column, InnoDB increments the counter and assigns the new value to the column. If you insert a row that explicitly specifies the column value, and the value is bigger than the current counter value, the counter is set to the specified column value.

If a user specifies `NULL` or 0 for the `AUTO_INCREMENT` column in an `INSERT`, InnoDB treats the row as if the value was not specified and generates a new value for it.

The behavior of the auto-increment mechanism is not defined if you assign a negative value to the column, or if the value becomes bigger than the maximum integer that can be stored in the specified integer type.

When accessing the auto-increment counter, InnoDB uses a special table-level `AUTO-INC` lock that it keeps to the end of the current SQL statement, not to the end of the transaction. The special lock release strategy was introduced to improve concurrency for inserts into a table containing an `AUTO_INCREMENT` column. Nevertheless, two transactions cannot have the `AUTO-INC` lock on the same table simultaneously, which can have a performance impact if the `AUTO-INC` lock is held for a long time. That might be the case

for a statement such as `INSERT INTO t1 ... SELECT ... FROM t2` that inserts all rows from one table into another.

InnoDB uses the in-memory auto-increment counter as long as the server runs. When the server is stopped and restarted, InnoDB reinitializes the counter for each table for the first `INSERT` to the table, as described earlier.

A server restart also cancels the effect of the `AUTO_INCREMENT = N` table option in `CREATE TABLE` and `ALTER TABLE` statements, which you can use with InnoDB tables to set the initial counter value or alter the current counter value.

You may see gaps in the sequence of values assigned to the `AUTO_INCREMENT` column if you roll back transactions that have generated numbers using the counter.

14.3.5.3.2. Configurable InnoDB Auto-Increment Locking

As described in the previous section, InnoDB uses a special lock called the table-level `AUTO-INC` lock for inserts into tables with `AUTO_INCREMENT` columns. This lock is normally held to the end of the statement (not to the end of the transaction), to ensure that auto-increment numbers are assigned in a predictable and repeatable order for a given sequence of `INSERT` statements.

In the case of statement-based replication, this means that when an SQL statement is replicated on a slave server, the same values are used for the auto-increment column as on the master server. The result of execution of multiple `INSERT` statements is deterministic, and the slave reproduces the same data as on the master. If auto-increment values generated by multiple `INSERT` statements were interleaved, the result of two concurrent `INSERT` statements would be nondeterministic, and could not reliably be propagated to a slave server using statement-based replication.

To make this clear, consider an example that uses this table:

```
CREATE TABLE t1 (  
  c1 INT(11) NOT NULL AUTO_INCREMENT,  
  c2 VARCHAR(10) DEFAULT NULL,  
  PRIMARY KEY (c1)  
) ENGINE=InnoDB;
```

Suppose that there are two transactions running, each inserting rows into a table with an `AUTO_INCREMENT` column. One transaction is using an `INSERT ... SELECT` statement that inserts 1000 rows, and another is using a simple `INSERT` statement that inserts one row:

```
Tx1: INSERT INTO t1 (c2) SELECT 1000 rows from another table ...  
Tx2: INSERT INTO t1 (c2) VALUES ('xxx');
```

InnoDB cannot tell in advance how many rows will be retrieved from the `SELECT` in the `INSERT` statement in Tx1, and it assigns the auto-increment values one at a time as the statement proceeds. With a table-level lock, held to the end of the statement, only one `INSERT` statement referring to table `t1` can execute at a time, and the generation of auto-increment numbers by different statements is not interleaved. The auto-increment value generated by the Tx1 `INSERT ... SELECT` statement will be consecutive, and the (single) auto-increment value used by the `INSERT` statement in Tx2 will either be smaller or larger than all those used for Tx1, depending on which statement executes first.

As long as the SQL statements execute in the same order when replayed from the binary log (when using statement-based replication, or in recovery scenarios), the results will be the same as they were when Tx1 and Tx2 first ran. Thus, table-level locks held until the end of a statement make `INSERT` statements using auto-increment safe for use with statement-based replication. However, those locks limit concurrency and scalability when multiple transactions are executing insert statements at the same time.

In the preceding example, if there were no table-level lock, the value of the auto-increment column used for the `INSERT` in Tx2 depends on precisely when the statement executes. If the `INSERT` of Tx2 executes while the `INSERT` of Tx1 is running (rather than before it starts or after it completes), the specific auto-increment values assigned by the two `INSERT` statements are nondeterministic, and may vary from run to run.

InnoDB can avoid using the table-level `AUTO-INC` lock for a class of `INSERT` statements where the number of rows is known in advance, and still preserve deterministic execution and safety for statement-based replication. Further, if you are not using the binary log to replay SQL statements as part of recovery or replication, you can entirely eliminate use of the table-level `AUTO-INC` lock for even greater concurrency and performance, at the cost of permitting gaps in auto-increment numbers assigned by a statement and potentially having the numbers assigned by concurrently executing statements interleaved.

For `INSERT` statements where the number of rows to be inserted is known at the beginning of processing the statement, InnoDB quickly allocates the required number of auto-increment values without taking any lock, but only if there is no concurrent session already holding the table-level `AUTO-INC` lock (because that other statement will be allocating auto-increment values one-by-one as it proceeds). More precisely, such an `INSERT` statement obtains auto-increment values under the control of a mutex (a light-weight lock) that is *not* held until the statement completes, but only for the duration of the allocation process.

This new locking scheme enables much greater scalability, but it does introduce some subtle differences in how auto-increment values are assigned compared to the original mechanism. To describe the way auto-increment works in InnoDB, the following discussion defines some terms, and explains how InnoDB behaves using different settings of the `innodb_autoinc_lock_mode` [1548] configuration parameter, which you can set at server startup. Additional considerations are described following the explanation of auto-increment locking behavior.

First, some definitions:

- “`INSERT`-like” statements

All statements that generate new rows in a table, including `INSERT`, `INSERT ... SELECT`, `REPLACE`, `REPLACE ... SELECT`, and `LOAD DATA`.

- “Simple inserts”

Statements for which the number of rows to be inserted can be determined in advance (when the statement is initially processed). This includes single-row and multiple-row `INSERT` and `REPLACE` statements that do not have a nested subquery, but not `INSERT ... ON DUPLICATE KEY UPDATE`.

- “Bulk inserts”

Statements for which the number of rows to be inserted (and the number of required auto-increment values) is not known in advance. This includes `INSERT ... SELECT`, `REPLACE ... SELECT`, and `LOAD DATA` statements, but not plain `INSERT`. InnoDB will assign new values for the `AUTO_INCREMENT` column one at a time as each row is processed.

- “Mixed-mode inserts”

These are “simple insert” statements that specify the auto-increment value for some (but not all) of the new rows. An example follows, where `c1` is an `AUTO_INCREMENT` column of table `t1`:

```
INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
```

Another type of “mixed-mode insert” is `INSERT ... ON DUPLICATE KEY UPDATE`, which in the worst case is in effect an `INSERT` followed by a `UPDATE`, where the allocated value for the `AUTO_INCREMENT` column may or may not be used during the update phase.

There are three possible settings for the `innodb_autoinc_lock_mode` [1548] parameter:

- `innodb_autoinc_lock_mode = 0` (“traditional” lock mode)

This lock mode provides the same behavior as before `innodb_autoinc_lock_mode` [1548] existed. For all “INSERT-like” statements, a special table-level `AUTO-INC` lock is obtained and held to the end of the statement. This assures that the auto-increment values assigned by any given statement are consecutive.

This lock mode is provided for:

- Backward compatibility.
 - Performance testing.
 - Working around issues with “mixed-mode inserts”, due to the possible differences in semantics described later.
- `innodb_autoinc_lock_mode = 1` (“consecutive” lock mode)

This is the default lock mode. In this mode, “bulk inserts” use the special `AUTO-INC` table-level lock and hold it until the end of the statement. This applies to all `INSERT ... SELECT`, `REPLACE ... SELECT`, and `LOAD DATA` statements. Only one statement holding the `AUTO-INC` lock can execute at a time.

With this lock mode, “simple inserts” (only) use a new locking model where a light-weight mutex is used during the allocation of auto-increment values, and no table-level `AUTO-INC` lock is used, unless an `AUTO-INC` lock is held by another transaction. If another transaction does hold an `AUTO-INC` lock, a “simple insert” waits for the `AUTO-INC` lock, as if it too were a “bulk insert”.

This lock mode ensures that, in the presence of `INSERT` statements where the number of rows is not known in advance (and where auto-increment numbers are assigned as the statement progresses), all auto-increment values assigned by any “INSERT-like” statement are consecutive, and operations are safe for statement-based replication.

Simply put, the important impact of this lock mode is significantly better scalability. This mode is safe for use with statement-based replication. Further, as with “traditional” lock mode, auto-increment numbers assigned by any given statement are *consecutive*. In this mode, there is *no change* in semantics compared to “traditional” mode for any statement that uses auto-increment, with one important exception.

The exception is for “mixed-mode inserts”, where the user provides explicit values for an `AUTO_INCREMENT` column for some, but not all, rows in a multiple-row “simple insert”. For such inserts, InnoDB will allocate more auto-increment values than the number of rows to be inserted. However, all values automatically assigned are consecutively generated (and thus higher than) the auto-increment value generated by the most recently executed previous statement. “Excess” numbers are lost.

- `innodb_autoinc_lock_mode = 2` (“interleaved” lock mode)

In this lock mode, no “INSERT-like” statements use the table-level `AUTO-INC` lock, and multiple statements can execute at the same time. This is the fastest and most scalable lock mode, but it is *not safe* when using statement-based replication or recovery scenarios when SQL statements are replayed from the binary log.

In this lock mode, auto-increment values are guaranteed to be unique and monotonically increasing across all concurrently executing “INSERT-like” statements. However, because multiple statements can be generating numbers at the same time (that is, allocation of numbers is *interleaved* across statements), the values generated for the rows inserted by any given statement may not be consecutive.

If the only statements executing are “simple inserts” where the number of rows to be inserted is known ahead of time, there will be no gaps in the numbers generated for a single statement, except for “mixed-mode inserts”. However, when “bulk inserts” are executed, there may be gaps in the auto-increment values assigned by any given statement.

The auto-increment locking modes provided by `innodb_autoinc_lock_mode` [1548] have several usage implications:

- Using auto-increment with replication

If you are using statement-based replication, set `innodb_autoinc_lock_mode` [1548] to 0 or 1 and use the same value on the master and its slaves. Auto-increment values are not ensured to be the same on the slaves as on the master if you use `innodb_autoinc_lock_mode` [1548] = 2 (“interleaved”) or configurations where the master and slaves do not use the same lock mode.

If you are using row-based replication, all of the auto-increment lock modes are safe. Row-based replication is not sensitive to the order of execution of the SQL statements.

- “Lost” auto-increment values and sequence gaps

In all lock modes (0, 1, and 2), if a transaction that generated auto-increment values rolls back, those auto-increment values are “lost”. Once a value is generated for an auto-increment column, it cannot be rolled back, whether or not the “INSERT-like” statement is completed, and whether or not the containing transaction is rolled back. Such lost values are not reused. Thus, there may be gaps in the values stored in an `AUTO_INCREMENT` column of a table.

- Gaps in auto-increment values for “bulk inserts”

With `innodb_autoinc_lock_mode` [1548] set to 0 (“traditional”) or 1 (“consecutive”), the auto-increment values generated by any given statement will be consecutive, without gaps, because the table-level `AUTO-INC` lock is held until the end of the statement, and only one such statement can execute at a time.

With `innodb_autoinc_lock_mode` [1548] set to 2 (“interleaved”), there may be gaps in the auto-increment values generated by “bulk inserts,” but only if there are concurrently executing “INSERT-like” statements.

For lock modes 1 or 2, gaps may occur between successive statements because for bulk inserts the exact number of auto-increment values required by each statement may not be known and overestimation is possible.

- Auto-increment values assigned by “mixed-mode inserts”

Consider a “mixed-mode insert,” where a “simple insert” specifies the auto-increment value for some (but not all) resulting rows. Such a statement will behave differently in lock modes 0, 1, and 2. For example, assume `c1` is an `AUTO_INCREMENT` column of table `t1`, and that the most recent automatically generated sequence number is 100. Consider the following “mixed-mode insert” statement:

```
INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
```

With `innodb_autoinc_lock_mode` [1548] set to 0 (“traditional”), the four new rows will be:

```
+-----+-----+
| c1   | c2   |
+-----+-----+
| 1    | a    |
```

101	b
5	c
102	d

The next available auto-increment value will be 103 because the auto-increment values are allocated one at a time, not all at once at the beginning of statement execution. This result is true whether or not there are concurrently executing “INSERT-like” statements (of any type).

With `innodb_autoinc_lock_mode` [1548] set to 1 (“consecutive”), the four new rows will also be:

c1	c2
1	a
101	b
5	c
102	d

However, in this case, the next available auto-increment value will be 105, not 103 because four auto-increment values are allocated at the time the statement is processed, but only two are used. This result is true whether or not there are concurrently executing “INSERT-like” statements (of any type).

With `innodb_autoinc_lock_mode` [1548] set to mode 2 (“interleaved”), the four new rows will be:

c1	c2
1	a
x	b
5	c
y	d

The values of `x` and `y` will be unique and larger than any previously generated rows. However, the specific values of `x` and `y` will depend on the number of auto-increment values generated by concurrently executing statements.

Finally, consider the following statement, issued when the most-recently generated sequence number was the value 4:

```
INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
```

With any `innodb_autoinc_lock_mode` [1548] setting, this statement will generate a duplicate-key error 23000 (Can't write; duplicate key in table) because 5 will be allocated for the row (NULL, 'b') and insertion of the row (5, 'c') will fail.

14.3.5.4. FOREIGN KEY Constraints

InnoDB supports foreign keys, which let you cross-reference related data across tables, and **foreign key constraints**, which help keep this spread-out data consistent. The syntax for an InnoDB foreign key constraint definition in the `CREATE TABLE` or `ALTER TABLE` statement looks like this:

```
[CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (index_col_name, ...)
  REFERENCES tbl_name (index_col_name,...)
  [ON DELETE reference_option]
  [ON UPDATE reference_option]

reference_option:
  RESTRICT | CASCADE | SET NULL | NO ACTION
```

index_name represents a foreign key ID. If given, this is ignored if an index for the foreign key is defined explicitly. Otherwise, if InnoDB creates an index for the foreign key, it uses *index_name* for the index name.

Foreign keys definitions are subject to the following conditions:

- Foreign key relationships involve a **parent table** that holds the central data values, and a **child table** with identical values pointing back to its parent. The **FOREIGN KEY** clause is specified in the child table. The parent and child tables must both be InnoDB tables. They must not be **TEMPORARY** tables.
- Corresponding columns in the foreign key and the referenced key must have similar internal data types inside InnoDB so that they can be compared without a type conversion. *The size and sign of integer types must be the same.* The length of string types need not be the same. For nonbinary (character) string columns, the character set and collation must be the same.
- InnoDB requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist. This index might be silently dropped later, if you create another index that can be used to enforce the foreign key constraint. *index_name*, if given, is used as described previously.
- InnoDB permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.
- Index prefixes on foreign key columns are not supported. One consequence of this is that **BLOB** and **TEXT** columns cannot be included in a foreign key because indexes on those columns must always include a prefix length.
- If the **CONSTRAINT symbol** clause is given, the *symbol* value must be unique in the database. If the clause is not given, InnoDB creates the name automatically.

Referential Actions

This section describes how foreign keys play an important part in safeguarding **referential integrity**.

InnoDB rejects any **INSERT** or **UPDATE** operation that attempts to create a foreign key value in a child table if there is no a matching candidate key value in the parent table.

When an **UPDATE** or **DELETE** operation affects a key value in the parent table that has matching rows in the child table, the result depends on the *referential action* specified using **ON UPDATE** and **ON DELETE** subclauses of the **FOREIGN KEY** clause. InnoDB supports five options regarding the action to be taken. If **ON DELETE** or **ON UPDATE** are not specified, the default action is **RESTRICT**.

- **CASCADE**: Delete or update the row from the parent table, and automatically delete or update the matching rows in the child table. Both **ON DELETE CASCADE** and **ON UPDATE CASCADE** are supported. Between two tables, do not define several **ON UPDATE CASCADE** clauses that act on the same column in the parent table or in the child table.



Note

Currently, cascaded foreign key actions do not activate triggers.

- **SET NULL**: Delete or update the row from the parent table, and set the foreign key column or columns in the child table to **NULL**. Both **ON DELETE SET NULL** and **ON UPDATE SET NULL** clauses are supported.

If you specify a `SET NULL` action, *make sure that you have not declared the columns in the child table as `NOT NULL`.*

- **RESTRICT:** Rejects the delete or update operation for the parent table. Specifying `RESTRICT` (or `NO ACTION`) is the same as omitting the `ON DELETE` or `ON UPDATE` clause.
- **NO ACTION:** A keyword from standard SQL. In MySQL, equivalent to `RESTRICT`. InnoDB rejects the delete or update operation for the parent table if there is a related foreign key value in the referenced table. Some database systems have deferred checks, and `NO ACTION` is a deferred check. In MySQL, foreign key constraints are checked immediately, so `NO ACTION` is the same as `RESTRICT`.
- **SET DEFAULT:** This action is recognized by the parser, but InnoDB rejects table definitions containing `ON DELETE SET DEFAULT` or `ON UPDATE SET DEFAULT` clauses.

InnoDB supports foreign key references between one column and another within a table. (A column cannot have a foreign key reference to itself.) In these cases, “child table records” really refers to dependent records within the same table.

Examples of Foreign Key Clauses

Here is a simple example that relates `parent` and `child` tables through a single-column foreign key:

```
CREATE TABLE parent (id INT NOT NULL,
                     PRIMARY KEY (id)
) ENGINE=INNODB;
CREATE TABLE child (id INT, parent_id INT,
                    INDEX par_ind (parent_id),
                    FOREIGN KEY (parent_id) REFERENCES parent(id)
                    ON DELETE CASCADE
) ENGINE=INNODB;
```

A more complex example in which a `product_order` table has foreign keys for two other tables. One foreign key references a two-column index in the `product` table. The other references a single-column index in the `customer` table:

```
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
                      price DECIMAL,
                      PRIMARY KEY(category, id)) ENGINE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
                       PRIMARY KEY (id)) ENGINE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                             product_category INT NOT NULL,
                             product_id INT NOT NULL,
                             customer_id INT NOT NULL,
                             PRIMARY KEY(no),
                             INDEX (product_category, product_id),
                             FOREIGN KEY (product_category, product_id)
                             REFERENCES product(category, id)
                             ON UPDATE CASCADE ON DELETE RESTRICT,
                             INDEX (customer_id),
                             FOREIGN KEY (customer_id)
                             REFERENCES customer(id)) ENGINE=INNODB;
```

InnoDB enables you to add a new foreign key constraint to a table by using `ALTER TABLE`:

```
ALTER TABLE tbl_name
  ADD [CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (index_col_name, ...)
  REFERENCES tbl_name (index_col_name,...)
  [ON DELETE reference_option]
  [ON UPDATE reference_option]
```

The foreign key can be self referential (referring to the same table). When you add a foreign key constraint to a table using `ALTER TABLE`, *remember to create the required indexes first*.

Foreign Keys and ALTER TABLE

InnoDB supports the use of `ALTER TABLE` to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

If the `FOREIGN KEY` clause included a `CONSTRAINT` name when you created the foreign key, you can refer to that name to drop the foreign key. Otherwise, the `fk_symbol` value is internally generated by InnoDB when the foreign key is created. To find out the symbol value when you want to drop a foreign key, use the `SHOW CREATE TABLE` statement. For example:

```
mysql> SHOW CREATE TABLE ibtest11c\G
***** 1. row *****
      Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`, `D`)
REFERENCES `ibtest11a` (`A`, `D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`, `C`)
REFERENCES `ibtest11a` (`B`, `C`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY `0_38775`;
```

You cannot add a foreign key and drop a foreign key in separate clauses of a single `ALTER TABLE` statement. Separate statements are required.

If `ALTER TABLE` for an InnoDB table results in changes to column values (for example, because a column is truncated), InnoDB's `FOREIGN KEY` constraint checks do not notice possible violations caused by changing the values.

How Foreign Keys Work with Other MySQL Commands

The InnoDB parser permits table and column identifiers in a `FOREIGN KEY ... REFERENCES ...` clause to be quoted within backticks. (Alternatively, double quotation marks can be used if the `ANSI_QUOTES` [612] SQL mode is enabled.) The InnoDB parser also takes into account the setting of the `lower_case_table_names` [518] system variable.

InnoDB returns a child table's foreign key definitions as part of the output of the `SHOW CREATE TABLE` statement:

```
SHOW CREATE TABLE tbl_name;
```

`mysqldump` also produces correct definitions of tables in the dump file, including the foreign keys for child tables.

To make it easier to reload dump files for tables that have foreign key relationships, `mysqldump` automatically includes a statement in the dump output to set `foreign_key_checks` [500] to 0. This

avoids problems with tables having to be reloaded in a particular order when the dump is reloaded. It is also possible to set this variable manually:

```
mysql> SET foreign_key_checks = 0;
mysql> SOURCE dump_file_name;
mysql> SET foreign_key_checks = 1;
```

This enables you to import the tables in any order if the dump file contains tables that are not correctly ordered for foreign keys. It also speeds up the import operation. Setting `foreign_key_checks` [500] to 0 can also be useful for ignoring foreign key constraints during `LOAD DATA` and `ALTER TABLE` operations. However, even if `foreign_key_checks = 0` [500], InnoDB does not permit the creation of a foreign key constraint where a column references a nonmatching column type. Also, if an InnoDB table has foreign key constraints, `ALTER TABLE` cannot be used to change the table to use another storage engine. To alter the storage engine, drop any foreign key constraints first.

You cannot issue `DROP TABLE` for an InnoDB table that is referenced by a `FOREIGN KEY` constraint, unless you do `SET foreign_key_checks = 0`. When you drop a table, the constraints that were defined in its create statement are also dropped.

If you re-create a table that was dropped, it must have a definition that conforms to the foreign key constraints referencing it. It must have the right column names and types, and it must have indexes on the referenced keys, as stated earlier. If these are not satisfied, MySQL returns error number 1005 and refers to error 150 in the error message.

If MySQL reports an error number 1005 from a `CREATE TABLE` statement, and the error message refers to error 150, table creation failed because a foreign key constraint was not correctly formed. Similarly, if an `ALTER TABLE` fails and it refers to error 150, that means a foreign key definition would be incorrectly formed for the altered table. To display a detailed explanation of the most recent InnoDB foreign key error in the server, issue `SHOW ENGINE INNODB STATUS`.



Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including InnoDB, recognizes or enforces the `MATCH` clause used in referential-integrity constraint definitions. Use of an explicit `MATCH` clause will not have the specified effect, and also causes `ON DELETE` and `ON UPDATE` clauses to be ignored. For these reasons, specifying `MATCH` should be avoided.

The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key. InnoDB essentially implements the semantics defined by `MATCH SIMPLE`, which permit a foreign key to be all or partially `NULL`. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.

Additionally, MySQL and InnoDB require that the referenced columns be indexed for performance. However, the system does not enforce a requirement that the referenced columns be `UNIQUE` or be declared `NOT NULL`. The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only `UNIQUE` and `NOT NULL` keys.

Furthermore, InnoDB does not recognize or support “inline `REFERENCES` specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. InnoDB accepts `REFERENCES` clauses only

when specified as part of a separate `FOREIGN KEY` specification. For other storage engines, MySQL Server parses and ignores foreign key specifications.

Deviation from SQL standards: If there are several rows in the parent table that have the same referenced key value, InnoDB acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you have defined a `RESTRICT` type constraint, and there is a child row with several parent rows, InnoDB does not permit the deletion of any of those parent rows.

InnoDB performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

Deviation from SQL standards: A `FOREIGN KEY` constraint that references a non-`UNIQUE` key is not standard SQL. It is an InnoDB extension to standard SQL.

Deviation from SQL standards: If `ON UPDATE CASCADE` or `ON UPDATE SET NULL` recurses to update the *same table* it has previously updated during the cascade, it acts like `RESTRICT`. This means that you cannot use self-referential `ON UPDATE CASCADE` or `ON UPDATE SET NULL` operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential `ON DELETE SET NULL`, on the other hand, is possible, as is a self-referential `ON DELETE CASCADE`. Cascading operations may not be nested more than 15 levels deep.

Deviation from SQL standards: Like MySQL in general, in an SQL statement that inserts, deletes, or updates many rows, InnoDB checks `UNIQUE` and `FOREIGN KEY` constraints row-by-row. When performing foreign key checks, InnoDB sets shared row-level locks on child or parent records it has to look at. InnoDB checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. Until InnoDB implements deferred constraint checking, some things will be impossible, such as deleting a record that refers to itself using a foreign key.

14.3.5.5. InnoDB and MySQL Replication

MySQL replication works for InnoDB tables as it does for MyISAM tables. It is also possible to use replication in a way where the storage engine on the slave is not the same as the original storage engine on the master. For example, you can replicate modifications to an InnoDB table on the master to a MyISAM table on the slave.

To set up a new slave for a master, you have to make a copy of the InnoDB tablespace and the log files, as well as the `.frm` files of the InnoDB tables, and move the copies to the slave. If the `innodb_file_per_table` [1555] variable is enabled, copy the `.ibd` files as well. For the proper procedure to do this, see [Section 14.3.7, “Backing Up and Recovering an InnoDB Database”](#).

If you can shut down the master or an existing slave, you can take a cold backup of the InnoDB tablespace and log files and use that to set up a slave. To make a new slave without taking down any server you can also use the MySQL Enterprise Backup product.

Transactions that fail on the master do not affect replication at all. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A transaction that fails (for example, because of a foreign key violation, or because it is rolled back) is not written to the binary log, so it is not sent to slaves. See [Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

Replication and CASCADE. Cascading actions for InnoDB tables on the master are replicated on the slave *only* if the tables sharing the foreign key relation use InnoDB on both the master and slave. This is true whether you are using statement-based or row-based replication. Suppose that you have started replication, and then create two tables on the master using the following `CREATE TABLE` statements:

```
CREATE TABLE fc1 (
  i INT PRIMARY KEY,
  j INT
) ENGINE = InnoDB;

CREATE TABLE fc2 (
  m INT PRIMARY KEY,
  n INT,
  FOREIGN KEY ni (n) REFERENCES fc1 (i)
    ON DELETE CASCADE
) ENGINE = InnoDB;
```

Suppose that the slave does not have [InnoDB](#) support enabled. If this is the case, then the tables on the slave are created, but they use the [MyISAM](#) storage engine, and the [FOREIGN KEY](#) option is ignored. Now we insert some rows into the tables on the master:

```
master> INSERT INTO fc1 VALUES (1, 1), (2, 2);
Query OK, 2 rows affected (0.09 sec)
Records: 2 Duplicates: 0 Warnings: 0

master> INSERT INTO fc2 VALUES (1, 1), (2, 2), (3, 1);
Query OK, 3 rows affected (0.19 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

At this point, on both the master and the slave, table `fc1` contains 2 rows, and table `fc2` contains 3 rows, as shown here:

```
master> SELECT * FROM fc1;
+-----+
| i | j |
+-----+
| 1 | 1 |
| 2 | 2 |
+-----+
2 rows in set (0.00 sec)

master> SELECT * FROM fc2;
+-----+
| m | n |
+-----+
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |
+-----+
3 rows in set (0.00 sec)

slave> SELECT * FROM fc1;
+-----+
| i | j |
+-----+
| 1 | 1 |
| 2 | 2 |
+-----+
2 rows in set (0.00 sec)

slave> SELECT * FROM fc2;
+-----+
| m | n |
+-----+
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |
+-----+
3 rows in set (0.00 sec)
```

Now suppose that you perform the following [DELETE](#) statement on the master:

```
master> DELETE FROM fc1 WHERE i=1;
Query OK, 1 row affected (0.09 sec)
```

Due to the cascade, table `fc2` on the master now contains only 1 row:

```
master> SELECT * FROM fc2;
+-----+
| m | n |
+-----+
| 2 | 2 |
+-----+
1 row in set (0.00 sec)
```

However, the cascade does not propagate on the slave because on the slave the `DELETE` for `fc1` deletes no rows from `fc2`. The slave's copy of `fc2` still contains all of the rows that were originally inserted:

```
slave> SELECT * FROM fc2;
+-----+
| m | n |
+-----+
| 1 | 1 |
| 3 | 1 |
| 2 | 2 |
+-----+
3 rows in set (0.00 sec)
```

This difference is due to the fact that the cascading deletes are handled internally by the InnoDB storage engine, which means that none of the changes are logged.

14.3.6. Adding, Removing, or Resizing InnoDB Data and Log Files

This section describes what you can do when your InnoDB tablespace runs out of room or when you want to change the size of the log files.

The easiest way to increase the size of the InnoDB tablespace is to configure it from the beginning to be auto-extending. Specify the `autoextend` attribute for the last data file in the tablespace definition. Then InnoDB increases the size of that file automatically in 8MB increments when it runs out of space. The increment size can be changed by setting the value of the `innodb_autoextend_increment` [1548] system variable, which is measured in MB.

Alternatively, you can increase the size of your tablespace by adding another data file. To do this, you have to shut down the MySQL server, change the tablespace configuration to add a new data file to the end of `innodb_data_file_path` [1552], and start the server again.

If your last data file was defined with the keyword `autoextend`, the procedure for reconfiguring the tablespace must take into account the size to which the last data file has grown. Obtain the size of the data file, round it down to the closest multiple of 1024×1024 bytes (= 1MB), and specify the rounded size explicitly in `innodb_data_file_path` [1552]. Then you can add another data file. Remember that only the last data file in the `innodb_data_file_path` [1552] can be specified as auto-extending.

As an example, assume that the tablespace has just one auto-extending data file `ibdata1`:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that this data file, over time, has grown to 988MB. Here is the configuration line after modifying the original data file to not be auto-extending and adding another auto-extending data file:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

When you add a new file to the tablespace configuration, make sure that it does not exist. InnoDB will create and initialize the file when you restart the server.

Currently, you cannot remove a data file from the tablespace. To decrease the size of your tablespace, use this procedure:

1. Use `mysqldump` to dump all your InnoDB tables.
2. Stop the server.
3. Remove all the existing tablespace files, including the `ibdata` and `ib_log` files. If you want to keep a backup copy of the information, then copy all the `ib*` files to another location before the removing the files in your MySQL installation.
4. Remove any `.frm` files for InnoDB tables.
5. Configure a new tablespace.
6. Restart the server.
7. Import the dump files.

If you want to change the number or the size of your InnoDB log files, use the following instructions. The procedure to use depends on the value of `innodb_fast_shutdown` [1553]:

- If `innodb_fast_shutdown` [1553] is not set to 2: Stop the MySQL server and make sure that it shuts down without errors (to ensure that there is no information for outstanding transactions in the log). Copy the old log files into a safe place in case something went wrong during the shutdown and you need them to recover the tablespace. Delete the old log files from the log file directory, edit `my.cnf` to change the log file configuration, and start the MySQL server again. `mysqld` sees that no InnoDB log files exist at startup and creates new ones.
- If `innodb_fast_shutdown` [1553] is set to 2: Set `innodb_fast_shutdown` [1553] to 1:

```
mysql> SET GLOBAL innodb_fast_shutdown = 1;
```

Then follow the instructions in the previous item.

14.3.7. Backing Up and Recovering an InnoDB Database

The key to safe database management is making regular backups. Depending on your data volume, number of MySQL servers, and database workload, you can use these techniques, alone or in combination: **hot backup** with MySQL Enterprise Backup; **cold backup** by copying files while the MySQL server is shut down; **physical backup** for fast operation (especially for restore); **logical backup** with `mysqldump` for smaller data volumes or to record the structure of schema objects.

Hot Backups

The `mysqlbackup` command, part of the MySQL Enterprise Backup component, lets you back up a running MySQL instance, including InnoDB and MyISAM tables, with minimal disruption to operations while producing a consistent snapshot of the database. When `mysqlbackup` is copying InnoDB tables, reads and writes to both InnoDB and MyISAM tables can continue. During the copying of MyISAM tables, reads (but not writes) to those tables are permitted. MySQL Enterprise Backup can also create compressed backup files, and back up subsets of tables and databases. In conjunction with MySQL's binary log, users can perform point-in-time recovery. MySQL Enterprise Backup is part of the MySQL Enterprise subscription. For more details, see [Section 24.2, "MySQL Enterprise Backup"](#).

Cold Backups

If you can shut down your MySQL server, you can make a binary backup that consists of all files used by InnoDB to manage its tables. Use the following procedure:

1. Do a [slow shutdown](#) of the MySQL server and make sure that it stops without errors.
2. Copy all InnoDB data files (`ibdata` files and `.ibd` files) into a safe place.
3. Copy all the `.frm` files for InnoDB tables to a safe place.
4. Copy all InnoDB log files (`ib_logfile` files) to a safe place.
5. Copy your `my.cnf` configuration file or files to a safe place.

Alternative Backup Types

In addition to making binary backups as just described, regularly make dumps of your tables with `mysqldump`. A binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, because the format is simpler, the chance for serious data corruption is smaller. `mysqldump` also has a `--single-transaction` [321] option for making a consistent snapshot without locking out other clients. See [Section 7.3.1, “Establishing a Backup Policy”](#).

Replication works with InnoDB tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability.

Performing Recovery

To recover your InnoDB database to the present from the time at which the binary backup was made, you must run your MySQL server with binary logging turned on, even before taking the backup. To achieve point-in-time recovery after restoring a backup, you can apply changes from the binary log that occurred after the backup was made. See [Section 7.5, “Point-in-Time \(Incremental\) Recovery Using the Binary Log”](#).

To recover from a crash of your MySQL server, the only requirement is to restart it. InnoDB automatically checks the logs and performs a roll-forward of the database to the present. InnoDB automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, `mysqld` displays output something like this:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

If your database becomes corrupted or disk failure occurs, you must perform the recovery using a backup. In the case of corruption, first find a backup that is not corrupted. After restoring the base backup, do a point-in-time recovery from the binary log files using `mysqlbinlog` and `mysql` to restore the changes that occurred after the backup was made.

In some cases of database corruption it is enough just to dump, drop, and re-create one or a few corrupt tables. You can use the `CHECK TABLE` SQL statement to check whether a table is corrupt, although `CHECK TABLE` naturally cannot detect every possible kind of corruption. You can use the Tablespace Monitor to check the integrity of the file space management inside the tablespace files.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best first to try restarting your computer. Doing so may eliminate errors that appeared to be database page corruption.

14.3.7.1. The InnoDB Recovery Process

InnoDB crash recovery consists of several steps. The first step, redo log application, is performed during the initialization, before accepting any connections. If all changes were flushed from the buffer pool to the tablespaces (`ibdata*` and `*.ibd` files) at the time of the shutdown or crash, the redo log application can be skipped. If the redo log files are missing at startup, InnoDB skips the redo log application.

The remaining steps after redo log application do not depend on the redo log (other than for logging the writes) and are performed in parallel with normal processing. These include:

- Rolling back incomplete transactions: Any transactions that were active at the time of crash or fast shutdown.
- Insert buffer merge: Applying changes from the insert buffer tree (from the shared tablespace) to leaf pages of secondary indexes as the index pages are read to the buffer pool.
- Purge: Deleting delete-marked records that are no longer visible for any active transaction.

Of these, only rollback of incomplete transactions is special to crash recovery. The insert buffer merge and the purge are performed during normal processing.

14.3.7.2. Forcing InnoDB Recovery

If there is database page corruption, you may want to dump your tables from the database with `SELECT ... INTO OUTFILE`. Usually, most of the data obtained in this way is intact. However, it is possible that the corruption might cause `SELECT * FROM tbl_name` statements or InnoDB background operations to crash or assert, or even cause InnoDB roll-forward recovery to crash. In such cases, you can use the `innodb_force_recovery` [1558] option to force the InnoDB storage engine to start up while preventing background operations from running, so that you can dump your tables. For example, you can add the following line to the `[mysqld]` section of your option file before restarting the server:

```
[mysqld]
innodb_force_recovery = 4
```

`innodb_force_recovery` [1558] is 0 by default (normal startup without forced recovery). The permissible nonzero values for `innodb_force_recovery` [1558] follow. A larger number includes all precautions of smaller numbers. If you can dump your tables with an option value of at most 4, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 6 is more drastic because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

- 1 (`SRV_FORCE_IGNORE_CORRUPT`)

Let the server run even if it detects a corrupt [page](#). Try to make `SELECT * FROM tbl_name` jump over corrupt index records and pages, which helps in dumping tables.

- 2 (`SRV_FORCE_NO_BACKGROUND`)

Prevent the [master thread](#) from running. If a crash would occur during the [purge](#) operation, this recovery value prevents it.

- 3 (`SRV_FORCE_NO_TRX_UNDO`)

Do not run transaction [rollbacks](#) after [crash recovery](#).

- 4 (`SRV_FORCE_NO_IBUF_MERGE`)

Prevent [insert buffer](#) merge operations. If they would cause a crash, do not do them. Do not calculate table [statistics](#).

- 5 (`SRV_FORCE_NO_UNDO_LOG_SCAN`)

Do not look at [undo logs](#) when starting the database: InnoDB treats even incomplete transactions as committed.

- 6 (`SRV_FORCE_NO_LOG_REDO`)

Do not do the [redo log](#) roll-forward in connection with recovery.

With this value, you might not be able to do queries other than a basic `SELECT * FROM t`, with no `WHERE`, `ORDER BY`, or other clauses. More complex queries could encounter corrupted data structures and fail.

If corruption within the table data prevents you from dumping the entire table contents, a query with an `ORDER BY primary_key DESC` clause might be able to dump the portion of the table after the corrupted part.

The database must not otherwise be used with any nonzero value of `innodb_force_recovery` [1558]. As a safety measure, InnoDB prevents users from performing `INSERT`, `UPDATE`, or `DELETE` operations when `innodb_force_recovery` [1558] is greater than 0.

You can `SELECT` from tables to dump them, or `DROP` or `CREATE` tables even if forced recovery is used. If you know that a given table is causing a crash on rollback, you can drop it. You can also use this to stop a runaway rollback caused by a failing mass import or `ALTER TABLE`. You can kill the `mysqld` process and set `innodb_force_recovery` [1558] to 3 to bring the database up without the rollback, then `DROP` the table that is causing the runaway rollback.

14.3.7.3. InnoDB Checkpoints

Making your log files very large may reduce disk I/O during [checkpointing](#). It often makes sense to set the total size of the log files as large as the buffer pool or even larger. Although in the past large log files could make crash recovery take excessive time, starting with MySQL 5.5, performance enhancements to crash recovery make it possible to use large log files with fast startup after a crash. (Strictly speaking, this performance improvement is available for MySQL 5.1 with the InnoDB Plugin 1.0.7 and higher. It is with MySQL 5.5 and InnoDB 1.1 that this improvement is available in the default InnoDB storage engine.)

How Checkpoint Processing Works

InnoDB implements a checkpoint mechanism known as “fuzzy” checkpointing. InnoDB flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one

single batch, which would in practice stop processing of user SQL statements during the checkpointing process.

During crash recovery, [InnoDB](#) looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then [InnoDB](#) scans the log files forward from the checkpoint, applying the logged modifications to the database.

[InnoDB](#) writes to its log files on a rotating basis. It also writes checkpoint information to the first log file at each checkpoint. All committed modifications that make the database pages in the buffer pool different from the images on disk must be available in the log files in case [InnoDB](#) has to do a recovery. This means that when [InnoDB](#) starts to reuse a log file, it has to make sure that the database page images on disk contain the modifications logged in the log file that [InnoDB](#) is going to reuse. In other words, [InnoDB](#) must create a checkpoint and this often involves flushing of modified database pages to disk.

14.3.8. Moving or Copying [InnoDB](#) Tables to Another Machine

This section explains various techniques for moving or copying some or all [InnoDB](#) tables to a different server. For example, you might move an entire MySQL instance to a larger, faster server; you might clone an entire MySQL instance to a new replication slave server; you might copy individual tables to another server to development and test an application, or to a data warehouse server to produce reports.

On Windows, [InnoDB](#) always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names. A convenient way to accomplish this is to add the following line to the `[mysqld]` section of your `my.cnf` or `my.ini` file before creating any databases or tables:

```
[mysqld]
lower_case_table_names=1
```

Like [MyISAM](#) data files, [InnoDB](#) data and log files are binary-compatible on all platforms having the same floating-point number format. You can move an [InnoDB](#) database simply by copying all the relevant files listed in [Section 14.3.7, “Backing Up and Recovering an \[InnoDB\]\(#\) Database”](#). If the floating-point formats differ but you have not used `FLOAT` or `DOUBLE` data types in your tables, then the procedure is the same: simply copy the relevant files. If you use `mysqldump` to dump your tables on one machine and then import the dump files on the other machine, it does not matter whether the formats differ or your tables contain floating-point data.

One way to increase performance is to switch off autocommit mode when importing data, assuming that the tablespace has enough space for the big rollback segment that the import transactions generate. Do the commit only after importing a whole table or a segment of a table.

14.3.9. The [InnoDB](#) Transaction Model and Locking

To implement a large-scale, busy, or highly reliable database application, to port substantial code from a different database system, or to push MySQL performance to the limits of the laws of physics, you must understand the notions of [transactions](#) and [locking](#) as they relate to the [InnoDB](#) storage engine.

In the [InnoDB](#) transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking. [InnoDB](#) does locking on the row level and runs queries as nonlocking consistent reads by default, in the style of Oracle. The lock information in [InnoDB](#) is stored so space-efficiently that lock escalation is not needed: Typically, several users are permitted to lock every row in [InnoDB](#) tables, or any random subset of the rows, without causing [InnoDB](#) memory exhaustion.

In [InnoDB](#), all user activity occurs inside a transaction. If autocommit mode is enabled, each SQL statement forms a single transaction on its own. By default, MySQL starts the session for each new connection with autocommit enabled, so MySQL does a commit after each SQL statement if that statement

did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See [Section 14.3.13, “InnoDB Error Handling”](#).

A session that has autocommit enabled can perform a multiple-statement transaction by starting it with an explicit `START TRANSACTION` or `BEGIN` statement and ending it with a `COMMIT` or `ROLLBACK` statement. See [Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

If autocommit mode is disabled within a session with `SET autocommit = 0`, the session always has a transaction open. A `COMMIT` or `ROLLBACK` statement ends the current transaction and a new one starts.

A `COMMIT` means that the changes made in the current transaction are made permanent and become visible to other sessions. A `ROLLBACK` statement, on the other hand, cancels all modifications made by the current transaction. Both `COMMIT` and `ROLLBACK` release all InnoDB locks that were set during the current transaction.

In terms of the SQL:1992 transaction isolation levels, the default InnoDB level is `REPEATABLE READ [1401]`. InnoDB offers all four transaction isolation levels described by the SQL standard: `READ UNCOMMITTED [1402]`, `READ COMMITTED [1402]`, `REPEATABLE READ [1401]`, and `SERIALIZABLE [1402]`.

A user can change the isolation level for a single session or for all subsequent connections with the `SET TRANSACTION` statement. To set the server's default isolation level for all connections, use the `--transaction-isolation [465]` option on the command line or in an option file. For detailed information about isolation levels and level-setting syntax, see [Section 13.3.6, “SET TRANSACTION Syntax”](#).

In row-level locking, InnoDB normally uses next-key locking. That means that besides index records, InnoDB can also lock the “gap” preceding an index record to block insertions by other sessions in the gap immediately before the index record. A next-key lock refers to a lock that locks an index record and the gap before it. A gap lock refers to a lock that locks only the gap before some index record.

For more information about row-level locking, and the circumstances under which gap locking is disabled, see [Section 14.3.9.4, “InnoDB Record, Gap, and Next-Key Locks”](#).

14.3.9.1. InnoDB Lock Modes

InnoDB implements standard row-level locking where there are two types of locks:

- A shared (*S*) lock permits a transaction to read a row.
- An exclusive (*X*) lock permits a transaction to update or delete a row.

If transaction *T1* holds a shared (*S*) lock on row *r*, then requests from some distinct transaction *T2* for a lock on row *r* are handled as follows:

- A request by *T2* for an *S* lock can be granted immediately. As a result, both *T1* and *T2* hold an *S* lock on *r*.
- A request by *T2* for an *X* lock cannot be granted immediately.

If a transaction *T1* holds an exclusive (*X*) lock on row *r*, a request from some distinct transaction *T2* for a lock of either type on *r* cannot be granted immediately. Instead, transaction *T2* has to wait for transaction *T1* to release its lock on row *r*.

Additionally, InnoDB supports *multiple granularity locking* which permits coexistence of record locks and locks on entire tables. To make locking at multiple granularity levels practical, additional types of locks called *intention locks* are used. Intention locks are table locks in InnoDB. The idea behind intention locks is for a transaction to indicate which type of lock (shared or exclusive) it will require later for a row in that table. There are two types of intention locks used in InnoDB (assume that transaction *T* has requested a lock of the indicated type on table *t*):

- Intention shared (*IS*): Transaction *T* intends to set *S* locks on individual rows in table *t*.
- Intention exclusive (*IX*): Transaction *T* intends to set *X* locks on those rows.

For example, `SELECT ... LOCK IN SHARE MODE` sets an *IS* lock and `SELECT ... FOR UPDATE` sets an *IX* lock.

The intention locking protocol is as follows:

- Before a transaction can acquire an *S* lock on a row in table *t*, it must first acquire an *IS* or stronger lock on *t*.
- Before a transaction can acquire an *X* lock on a row, it must first acquire an *IX* lock on *t*.

These rules can be conveniently summarized by means of the following *lock type compatibility matrix*.

	<i>X</i>	<i>IX</i>	<i>S</i>	<i>IS</i>
<i>X</i>	Conflict	Conflict	Conflict	Conflict
<i>IX</i>	Conflict	Compatible	Conflict	Compatible
<i>S</i>	Conflict	Conflict	Compatible	Compatible
<i>IS</i>	Conflict	Compatible	Compatible	Compatible

A lock is granted to a requesting transaction if it is compatible with existing locks, but not if it conflicts with existing locks. A transaction waits until the conflicting existing lock is released. If a lock request conflicts with an existing lock and cannot be granted because it would cause deadlock, an error occurs.

Thus, intention locks do not block anything except full table requests (for example, `LOCK TABLES ... WRITE`). The main purpose of *IX* and *IS* locks is to show that someone is locking a row, or going to lock a row in the table.

The following example illustrates how an error can occur when a lock request would cause a deadlock. The example involves two clients, A and B.

First, client A creates a table containing one row, and then begins a transaction. Within the transaction, A obtains an *S* lock on the row by selecting it in share mode:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;
+-----+
| i     |
+-----+
| 1    |
+-----+
1 row in set (0.10 sec)
```

Next, client B begins a transaction and attempts to delete the row from the table:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

The delete operation requires an *X* lock. The lock cannot be granted because it is incompatible with the *S* lock that client A holds, so the request goes on the queue of lock requests for the row and client B blocks.

Finally, client A also attempts to delete the row from the table:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

Deadlock occurs here because client A needs an *X* lock to delete the row. However, that lock request cannot be granted because client B already has a request for an *X* lock and is waiting for client A to release its *S* lock. Nor can the *S* lock held by A be upgraded to an *X* lock because of the prior request by B for an *X* lock. As a result, InnoDB generates an error for one of the clients and releases its locks. The client returns this error:

```
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

At that point, the lock request for the other client can be granted and it deletes the row from the table.

14.3.9.2. Consistent Nonlocking Reads

A consistent read means that InnoDB uses multi-versioning to present to a query a snapshot of the database at a point in time. The query sees the changes made by transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by earlier statements within the same transaction. This exception causes the following anomaly: If you update some rows in a table, a *SELECT* sees the latest version of the updated rows, but it might also see older versions of any rows. If other sessions simultaneously update the same table, the anomaly means that you might see the table in a state that never existed in the database.

If the transaction isolation level is *REPEATABLE READ* [1401] (the default level), all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

With *READ COMMITTED* [1402] isolation level, each consistent read within a transaction sets and reads its own fresh snapshot.

Consistent read is the default mode in which InnoDB processes *SELECT* statements in *READ COMMITTED* [1402] and *REPEATABLE READ* [1401] isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other sessions are free to modify those tables at the same time a consistent read is being performed on the table.

Suppose that you are running in the default *REPEATABLE READ* [1401] isolation level. When you issue a consistent read (that is, an ordinary *SELECT* statement), InnoDB gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.



Note

The snapshot of the database state applies to *SELECT* statements within a transaction, not necessarily to *DML* statements. If you insert or modify some rows and then commit that transaction, a *DELETE* or *UPDATE* statement issued from another concurrent *REPEATABLE READ* transaction could affect those just-committed rows, even though the session could not query them. If a transaction does update or delete rows committed by a different transaction, those changes do become visible to the current transaction. For example, you might encounter a situation like the following:

```

SELECT COUNT(c1) FROM t1 WHERE c1 = 'xyz'; -- Returns 0: no rows match.
DELETE FROM t1 WHERE c1 = 'xyz'; -- Deletes several rows recently committed by other transact

SELECT COUNT(c2) FROM t1 WHERE c2 = 'abc'; -- Returns 0: no rows match.
UPDATE t1 SET c2 = 'cba' WHERE c2 = 'abc'; -- Affects 10 rows: another txn just committed 10
SELECT COUNT(c2) FROM t1 WHERE c2 = 'cba'; -- Returns 10: this txn can now see the rows it ju
    
```

You can advance your timepoint by committing your transaction and then doing another [SELECT](#) or [START TRANSACTION WITH CONSISTENT SNAPSHOT](#).

This is called *multi-versioned concurrency control*.

In the following example, session A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

	Session A	Session B
time	SET autocommit=0;	SET autocommit=0;
	SELECT * FROM t;	
	empty set	
		INSERT INTO t VALUES (1, 2);
v	SELECT * FROM t;	
	empty set	COMMIT;
	SELECT * FROM t;	
	empty set	
	COMMIT;	
	SELECT * FROM t;	

	1 2	

	1 row in set	

If you want to see the “freshest” state of the database, use either the [READ COMMITTED \[1402\]](#) isolation level or a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

With [READ COMMITTED \[1402\]](#) isolation level, each consistent read within a transaction sets and reads its own fresh snapshot. With [LOCK IN SHARE MODE](#), a locking read occurs instead: A [SELECT](#) blocks until the transaction containing the freshest rows ends (see [Section 14.3.9.3, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”](#)).

Consistent read does not work over certain DDL statements:

- Consistent read does not work over [DROP TABLE](#), because MySQL cannot use a table that has been dropped and [InnoDB](#) destroys the table.
- Consistent read does not work over [ALTER TABLE](#), because that statement makes a temporary copy of the original table and deletes the original table when the temporary copy is built. When you reissue a consistent read within a transaction, rows in the new table are not visible because those rows did not exist when the transaction's snapshot was taken.

The type of read varies for selects in clauses like [INSERT INTO ... SELECT](#), [UPDATE ... \(SELECT\)](#), and [CREATE TABLE ... SELECT](#) that do not specify [FOR UPDATE](#) or [LOCK IN SHARE MODE](#):

- By default, InnoDB uses stronger locks and the `SELECT` part acts like `READ COMMITTED` [1402], where each consistent read, even within the same transaction, sets and reads its own fresh snapshot.
- To use a consistent read in such cases, enable the `innodb_locks_unsafe_for_binlog` [1561] option and set the isolation level of the transaction to `READ UNCOMMITTED` [1402], `READ COMMITTED` [1402], or `REPEATABLE READ` [1401] (that is, anything other than `SERIALIZABLE` [1402]). In this case, no locks are set on rows read from the selected table.

14.3.9.3. `SELECT ... FOR UPDATE` and `SELECT ... LOCK IN SHARE MODE` Locking Reads

If you query data and then insert or update related data within the same transaction, the regular `SELECT` statement does not give enough protection. Other transactions can update or delete the same rows you just queried. InnoDB supports two types of locking reads that offer extra safety:

- `SELECT ... LOCK IN SHARE MODE` sets a shared mode lock on any rows that are read. Other sessions can read the rows, but cannot modify them until your transaction commits. If any of these rows were changed by another transaction that has not yet committed, your query waits until that transaction ends and then uses the latest values.
- `SELECT ... FOR UPDATE` locks the rows and any associated index entries, the same as if you issued an `UPDATE` statement for those rows. Other transactions are blocked from updating those rows, from doing `SELECT ... LOCK IN SHARE MODE`, or from reading the data in certain transaction isolation levels. Consistent reads ignore any locks set on the records that exist in the read view. (Old versions of a record cannot be locked; they are reconstructed by applying undo logs on an in-memory copy of the record.)

These clauses are primarily useful when dealing with tree-structured or graph-structured data, either in a single table or split across multiple tables.

All locks set by `LOCK IN SHARE MODE` and `FOR UPDATE` queries are released when the transaction is committed or rolled back.



Note

Locking of rows for update using `SELECT FOR UPDATE` only applies when autocommit is disabled (either by beginning transaction with `START TRANSACTION` or by setting `autocommit` [484] to 0. If autocommit is enabled, the rows matching the specification are not locked.

Usage Examples

Suppose that you want to insert a new row into a table `child`, and make sure that the child row has a parent row in table `parent`. Your application code can ensure referential integrity throughout this sequence of operations.

First, use a consistent read to query the table `PARENT` and verify that the parent row exists. Can you safely insert the child row to table `CHILD`? No, because some other session could delete the parent row in the moment between your `SELECT` and your `INSERT`, without you being aware of it.

To avoid this potential issue, perform the `SELECT` using `LOCK IN SHARE MODE`:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

After the `LOCK IN SHARE MODE` query returns the parent 'Jones', you can safely add the child record to the `CHILD` table and commit the transaction. Any transaction that tries to read or write to the applicable row in the `PARENT` table waits until you are finished, that is, the data in all tables is in a consistent state.

For another example, consider an integer counter field in a table `CHILD_CODES`, used to assign a unique identifier to each child added to table `CHILD`. Do not use either consistent read or a shared mode read to read the present value of the counter, because two users of the database could see the same value for the counter, and a duplicate-key error occurs if two transactions attempt to add rows with the same identifier to the `CHILD` table.

Here, `LOCK IN SHARE MODE` is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when it attempts to update the counter.

Here are two ways to implement reading and incrementing the counter without interference from another transaction:

- First update the counter by incrementing it by 1, then read it and use the new value in the `CHILD` table. Any other transaction that tries to read the counter waits until your transaction commits. If another transaction is in the middle of this same sequence, your transaction waits until the other one commits.
- First perform a locking read of the counter using `FOR UPDATE`, and then increment the counter:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A `SELECT ... FOR UPDATE` reads the latest available data, setting exclusive locks on each row it reads. Thus, it sets the same locks a searched SQL `UPDATE` would set on the rows.

The preceding description is merely an example of how `SELECT ... FOR UPDATE` works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

The `SELECT` statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

14.3.9.4. InnoDB Record, Gap, and Next-Key Locks

InnoDB has several types of record-level locks:

- Record lock: This is a lock on an index record.
- Gap lock: This is a lock on a gap between index records, or a lock on the gap before the first or after the last index record.
- Next-key lock: This is a combination of a record lock on the index record and a gap lock on the gap before the index record.

Record locks always lock index records, even if a table is defined with no indexes. For such cases, InnoDB creates a hidden clustered index and uses this index for record locking. See [Section 14.3.11.1, “Clustered and Secondary Indexes”](#).

By default, InnoDB operates in `REPEATABLE READ [1401]` transaction isolation level and with the `innodb_locks_unsafe_for_binlog [1561]` system variable disabled. In this case, InnoDB uses next-key locks for searches and index scans, which prevents phantom rows (see [Section 14.3.9.5, “Avoiding the Phantom Problem Using Next-Key Locking”](#)).

Next-key locking combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock

plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record *R* in an index, another session cannot insert a new index record in the gap immediately before *R* in the index order.

Suppose that an index contains the values 10, 11, 13, and 20. The possible next-key locks for this index cover the following intervals, where (or) denote exclusion of the interval endpoint and [or] denote inclusion of the endpoint:

```
(negative infinity, 10]
(10, 11]
(11, 13]
(13, 20]
(20, positive infinity)
```

For the last interval, the next-key lock locks the gap above the largest value in the index and the “supremum” pseudo-record having a value higher than any value actually in the index. The supremum is not a real index record, so, in effect, this next-key lock locks only the gap following the largest index value.

The preceding example shows that a gap might span a single index value, multiple index values, or even be empty.

Gap locking is not needed for statements that lock rows using a unique index to search for a unique row. (This does not include the case that the search condition includes only some columns of a multiple-column unique index; in that case, gap locking does occur.) For example, if the *id* column has a unique index, the following statement uses only an index-record lock for the row having *id* value 100 and it does not matter whether other sessions insert rows in the preceding gap:

```
SELECT * FROM child WHERE id = 100;
```

If *id* is not indexed or has a nonunique index, the statement does lock the preceding gap.

A type of gap lock called an insertion intention gap lock is set by *INSERT* operations prior to row insertion. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

Gap locking can be disabled explicitly. This occurs if you change the transaction isolation level to *READ COMMITTED* [1402] or enable the *innodb_locks_unsafe_for_binlog* [1561] system variable. Under these circumstances, gap locking is disabled for searches and index scans and is used only for foreign-key constraint checking and duplicate-key checking.

There are also other effects of using the *READ COMMITTED* [1402] isolation level or enabling *innodb_locks_unsafe_for_binlog* [1561]: Record locks for nonmatching rows are released after MySQL has evaluated the *WHERE* condition. For *UPDATE* statements, InnoDB does a “semi-consistent” read, such that it returns the latest committed version to MySQL so that MySQL can determine whether the row matches the *WHERE* condition of the *UPDATE*.

14.3.9.5. Avoiding the Phantom Problem Using Next-Key Locking

The so-called *phantom* problem occurs within a transaction when the same query produces different sets of rows at different times. For example, if a *SELECT* is executed twice, but returns a row the second time that was not returned the first time, the row is a “phantom” row.

Suppose that there is an index on the *id* column of the *child* table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where `id` is bigger than 100. Let the table contain rows having `id` values of 90 and 102. If the locks set on the index records in the scanned range do not lock out inserts made in the gaps (in this case, the gap between 90 and 102), another session can insert a new row into the table with an `id` of 101. If you were to execute the same `SELECT` within the same transaction, you would see a new row with an `id` of 101 (a “phantom”) in the result set returned by the query. If we regard a set of rows as a data item, the new phantom child would violate the isolation principle of transactions that a transaction should be able to run so that the data it has read does not change during the transaction.

To prevent phantoms, InnoDB uses an algorithm called *next-key locking* that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record `R` in an index, another session cannot insert a new index record in the gap immediately before `R` in the index order.

When InnoDB scans an index, it can also lock the gap after the last record in the index. Just that happens in the preceding example: To prevent any insert into the table where `id` would be bigger than 100, the locks set by InnoDB include a lock on the gap following `id` value 102.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus, the next-key locking enables you to “lock” the nonexistence of something in your table.

Gap locking can be disabled as discussed in [Section 14.3.9.4, “InnoDB Record, Gap, and Next-Key Locks”](#). This may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled.

14.3.9.6. Locks Set by Different SQL Statements in InnoDB

A locking read, an `UPDATE`, or a `DELETE` generally set record locks on every index record that is scanned in the processing of the SQL statement. It does not matter whether there are `WHERE` conditions in the statement that would exclude the row. InnoDB does not remember the exact `WHERE` condition, but only knows which index ranges were scanned. The locks are normally next-key locks that also block inserts into the “gap” immediately before the record. However, gap locking can be disabled explicitly, which causes next-key locking not to be used. For more information, see [Section 14.3.9.4, “InnoDB Record, Gap, and Next-Key Locks”](#). The transaction isolation level also can affect which locks are set; see [Section 13.3.6, “SET TRANSACTION Syntax”](#).

If a secondary index is used in a search and index record locks to be set are exclusive, InnoDB also retrieves the corresponding clustered index records and sets locks on them.

Differences between shared and exclusive locks are described in [Section 14.3.9.1, “InnoDB Lock Modes”](#).

If you have no indexes suitable for your statement and MySQL must scan the entire table to process the statement, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily scan many rows.

For `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE`, locks are acquired for scanned rows, and expected to be released for rows that do not qualify for inclusion in the result set (for example,

if they do not meet the criteria given in the `WHERE` clause). However, in some cases, rows might not be unlocked immediately because the relationship between a result row and its original source is lost during query execution. For example, in a `UNION`, scanned (and locked) rows from a table might be inserted into a temporary table before evaluation whether they qualify for the result set. In this circumstance, the relationship of the rows in the temporary table to the rows in the original table is lost and the latter rows are not unlocked until the end of query execution.

InnoDB sets specific types of locks as follows.

- `SELECT ... FROM` is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to `SERIALIZABLE [1402]`. For `SERIALIZABLE [1402]` level, the search sets shared next-key locks on the index records it encounters.
- `SELECT ... FROM ... LOCK IN SHARE MODE` sets shared next-key locks on all index records the search encounters.
- For index records the search encounters, `SELECT ... FROM ... FOR UPDATE` blocks other sessions from doing `SELECT ... FROM ... LOCK IN SHARE MODE` or from reading in certain transaction isolation levels. Consistent reads will ignore any locks set on the records that exist in the read view.
- `UPDATE ... WHERE ...` sets an exclusive next-key lock on every record the search encounters.
- `DELETE FROM ... WHERE ...` sets an exclusive next-key lock on every record the search encounters.
- `INSERT` sets an exclusive lock on the inserted row. This lock is an index-record lock, not a next-key lock (that is, there is no gap lock) and does not prevent other sessions from inserting into the gap before the inserted row.

Prior to inserting the row, a type of gap lock called an insertion intention gap lock is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

If a duplicate-key error occurs, a shared lock on the duplicate index record is set. This use of a shared lock can result in deadlock should there be multiple sessions trying to insert the same row if another session already has an exclusive lock. This can occur if another session deletes the row. Suppose that an InnoDB table `t1` has the following structure:

```
CREATE TABLE t1 (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
```

Now suppose that three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;  
INSERT INTO t1 VALUES(1);
```

Session 2:

```
START TRANSACTION;  
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;  
INSERT INTO t1 VALUES(1);
```

Session 1:

```
ROLLBACK;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 rolls back, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

A similar situation occurs if the table already contains a row with key value 1 and three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
DELETE FROM t1 WHERE i = 1;
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
COMMIT;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 commits, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

- `INSERT ... ON DUPLICATE KEY UPDATE` differs from a simple `INSERT` in that an exclusive next-key lock rather than a shared lock is placed on the row to be updated when a duplicate-key error occurs.
- `REPLACE` is done like an `INSERT` if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row to be replaced.
- `INSERT INTO T SELECT ... FROM S WHERE ...` sets an exclusive index record without a gap lock on each row inserted into `T`. If the transaction isolation level is `READ COMMITTED` [1402] or `innodb_locks_unsafe_for_binlog` [1561] is enabled, and the transaction isolation level is not `SERIALIZABLE` [1402], InnoDB does the search on `S` as a consistent read (no locks). Otherwise, InnoDB sets shared next-key locks on rows from `S`. InnoDB has to set locks in the latter case: In roll-forward recovery from a backup, every SQL statement must be executed in exactly the same way it was done originally.

`CREATE TABLE ... SELECT ...` performs the `SELECT` with shared next-key locks or as a consistent read, as for `INSERT ... SELECT`.

When a `SELECT` is used in the constructs `REPLACE INTO t SELECT ... FROM s WHERE ...` or `UPDATE t ... WHERE col IN (SELECT ... FROM s ...)`, InnoDB sets shared next-key locks on rows from table `s`.

- While initializing a previously specified `AUTO_INCREMENT` column on a table, InnoDB sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column. In accessing the auto-increment counter, InnoDB uses a specific `AUTO-INC` table lock mode where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other sessions cannot insert into the table while the `AUTO-INC` table lock is held; see [Section 14.3.9, “The InnoDB Transaction Model and Locking”](#).

InnoDB fetches the value of a previously initialized `AUTO_INCREMENT` column without setting any locks.

- If a `FOREIGN KEY` constraint is defined on a table, any insert, update, or delete that requires the constraint condition to be checked sets shared record-level locks on the records that it looks at to check the constraint. InnoDB also sets these locks in the case where the constraint fails.
- `LOCK TABLES` sets table locks, but it is the higher MySQL layer above the InnoDB layer that sets these locks. InnoDB is aware of table locks if `innodb_table_locks = 1` (the default) and `autocommit = 0` [484], and the MySQL layer above InnoDB knows about row-level locks.

Otherwise, InnoDB's automatic deadlock detection cannot detect deadlocks where such table locks are involved. Also, because in this case the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another session currently has row-level locks. However, this does not endanger transaction integrity, as discussed in [Section 14.3.9.8, “Deadlock Detection and Rollback”](#). See also [Section 14.3.15, “Limits on InnoDB Tables”](#).

14.3.9.7. Implicit Transaction Commit and Rollback

By default, MySQL starts the session for each new connection with autocommit mode enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See [Section 14.3.13, “InnoDB Error Handling”](#).

If a session that has autocommit disabled ends without explicitly committing the final transaction, MySQL rolls back that transaction.

Some statements implicitly end a transaction, as if you had done a `COMMIT` before executing the statement. For details, see [Section 13.3.3, “Statements That Cause an Implicit Commit”](#).

14.3.9.8. Deadlock Detection and Rollback

InnoDB automatically detects transaction [deadlocks](#) and rolls back a transaction or transactions to break the deadlock. InnoDB tries to pick small transactions to roll back, where the size of a transaction is determined by the number of rows inserted, updated, or deleted.

InnoDB is aware of table locks if `innodb_table_locks = 1` (the default) and `autocommit = 0` [484], and the MySQL layer above it knows about row-level locks. Otherwise, InnoDB cannot detect deadlocks where a table lock set by a MySQL `LOCK TABLES` statement or a lock set by a storage engine other than InnoDB is involved. Resolve these situations by setting the value of the `innodb_lock_wait_timeout` [1560] system variable.

When InnoDB performs a complete rollback of a transaction, all locks set by the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the statement may be preserved. This happens because InnoDB stores row locks in a format such that it cannot know afterward which lock was set by which statement.

If a `SELECT` calls a stored function in a transaction, and a statement within the function fails, that statement rolls back. Furthermore, if `ROLLBACK` is executed after that, the entire transaction rolls back.

For techniques to organize database operations to avoid deadlocks, see [Section 14.3.9.9, “How to Cope with Deadlocks”](#).

14.3.9.9. How to Cope with Deadlocks

This section builds on the conceptual information about deadlocks in [Section 14.3.9.8, “Deadlock Detection and Rollback”](#). It explains how to organize database operations to minimize deadlocks and the subsequent error handling required in applications.

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really “atomic”; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- Use `SHOW ENGINE INNODB STATUS` to determine the cause of the latest deadlock. That can help you to tune your application to avoid deadlocks.
- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.
- Commit your transactions immediately after making a set of related changes. Small transactions are less prone to collision. In particular, do not leave an interactive `mysql` session open for a long time with an uncommitted transaction.
- If you are using locking reads (`SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE`), try using a lower isolation level such as `READ COMMITTED [1402]`.
- When modifying multiple tables within a transaction, or different sets of rows in the same table, do those operations in a consistent order each time. Then transactions form well-defined queues and do not deadlock. For example, organize database operations into functions within your application, or call stored routines, rather than coding multiple similar sequences of `INSERT`, `UPDATE`, and `DELETE` statements in different places.
- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use `EXPLAIN SELECT` to determine which indexes the MySQL server regards as the most appropriate for your queries.
- Use less locking. If you can afford to permit a `SELECT` to return data from an old snapshot, do not add the clause `FOR UPDATE` or `LOCK IN SHARE MODE` to it. Using the `READ COMMITTED [1402]` isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.
- If nothing else helps, serialize your transactions with table-level locks. The correct way to use `LOCK TABLES` with transactional tables, such as InnoDB tables, is to begin a transaction with `SET autocommit = 0` (not `START TRANSACTION`) followed by `LOCK TABLES`, and to not call `UNLOCK TABLES` until you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

Table-level locks prevent concurrent updates to the table, avoiding deadlocks at the expense of less responsiveness for a busy system.

- Another way to serialize transactions is to create an auxiliary “semaphore” table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the [InnoDB](#) instant deadlock detection algorithm also works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.

14.3.10. InnoDB Multi-Versioning

[InnoDB](#) is a multi-versioned storage engine: it keeps information about old versions of changed rows, to support transactional features such as concurrency and rollback. This information is stored in the tablespace in a data structure called a [rollback segment](#) (after an analogous data structure in Oracle). [InnoDB](#) uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Internal Details of Multi-Versioning

Internally, [InnoDB](#) adds three fields to each row stored in the database. A 6-byte `DB_TRX_ID` field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte `DB_ROLL_PTR` field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated. A 6-byte `DB_ROW_ID` field contains a row ID that increases monotonically as new rows are inserted. If [InnoDB](#) generates a clustered index automatically, the index contains row ID values. Otherwise, the `DB_ROW_ID` column does not appear in any index.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, but they can be discarded only after there is no transaction present for which [InnoDB](#) has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

Guidelines for Managing Rollback Segments

Commit your transactions regularly, including those transactions that issue only consistent reads. Otherwise, [InnoDB](#) cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space needed for your rollback segment.

In the [InnoDB](#) multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. [InnoDB](#) only physically removes the corresponding row and its index records when it discards the update undo log record written for the deletion. This removal operation is called a [purge](#), and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

If you insert and delete rows in smallish batches at about the same rate in the table, the purge thread can start to lag behind and the table can grow bigger and bigger because of all the “dead” rows, making everything disk-bound and very slow. In such a case, throttle new row operations, and allocate more

resources to the purge thread by tuning the `innodb_max_purge_lag` [1565] system variable. See Section 14.3.4, “InnoDB Startup Options and System Variables” for more information.

14.3.11. InnoDB Table and Index Structures

Role of the .frm File

MySQL stores its data dictionary information for tables in `.frm` files in database directories. This is true for all MySQL storage engines, but every InnoDB table also has its own entry in the InnoDB internal data dictionary inside the tablespace. When MySQL drops a table or a database, it has to delete one or more `.frm` files as well as the corresponding entries inside the InnoDB data dictionary. Consequently, you cannot move InnoDB tables between databases simply by moving the `.frm` files.

14.3.11.1. Clustered and Secondary Indexes

Every InnoDB table has a special index called the **clustered index** where the data for the rows is stored. Typically, the clustered index is synonymous with the **primary key**. To get the best performance from queries, inserts, and other database operations, you must understand how InnoDB uses the clustered index to optimize the most common lookup and DML operations for each table.

- If you define a **PRIMARY KEY** on your table, InnoDB uses it as the clustered index. Define a primary key for each table that you create. If there is no logical unique and non-null column or set of columns, add a new **auto-increment** column, whose values are filled in automatically.
- If you do not define a **PRIMARY KEY** for your table, MySQL locates the first **UNIQUE** index where all the key columns are **NOT NULL** and InnoDB uses it as the clustered index.
- If the table has no **PRIMARY KEY** or suitable **UNIQUE** index, InnoDB internally generates a hidden clustered index on a synthetic column containing row ID values. The rows are ordered by the ID that InnoDB assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in insertion order.

How the Clustered Index Speeds Up Queries

Accessing a row through the clustered index is fast because the row data is on the same page where the index search leads. If a table is large, the clustered index architecture often saves a disk I/O operation when compared to storage organizations that store row data using a different page from the index record. (For example, MyISAM uses one file for data rows and another for index records.)

How Secondary Indexes Relate to the Clustered Index

All indexes other than the clustered index are known as **secondary indexes**. In InnoDB, each record in a secondary index contains the primary key columns for the row, as well as the columns specified for the secondary index. InnoDB uses this primary key value to search for the row in the clustered index.

If the primary key is long, the secondary indexes use more space, so it is advantageous to have a short primary key.

14.3.11.2. Physical Structure of an InnoDB Index

All InnoDB indexes are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB. When new records are inserted, InnoDB tries to leave 1/16 of the page free for future insertions and updates of the index records.

If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full. If the **fill factor** of an index page drops below 1/2, InnoDB tries to contract the index tree to free the page.

**Note**

Changing the page size is not a supported operation and there is no guarantee that InnoDB will function normally with a page size other than 16KB. Problems compiling or running InnoDB may occur. In particular, `ROW_FORMAT=COMPRESSED` in the Barracuda file format assumes that the page size is at most 16KB and uses 14-bit pointers.

A version of InnoDB built for one page size cannot use data files or log files from a version built for a different page size.

14.3.11.3. Insert Buffering

Database applications often insert new rows in the ascending order of the primary key. In this case, due to the layout of the clustered index in the same order as the primary key, insertions into an InnoDB table do not require random reads from a disk.

On the other hand, secondary indexes are usually nonunique, and insertions into secondary indexes happen in a relatively random order. In the same way, deletes and updates can affect data pages that are not adjacent in secondary indexes. This would cause a lot of random disk I/O operations without a special mechanism used in InnoDB.

When an index record is inserted, marked for deletion, or deleted from a nonunique secondary index, InnoDB checks whether the secondary index page is in the buffer pool. If that is the case, InnoDB applies the change directly to the index page. If the index page is not found in the buffer pool, InnoDB records the change in a special structure known as the **insert buffer**. The insert buffer is kept small so that it fits entirely in the buffer pool, and changes can be applied very quickly. This process is known as **change buffering**. (Formerly, it applied only to inserts and was called insert buffering. The data structure is still called the insert buffer.)

Disk I/O for Flushing the Insert Buffer

Periodically, the insert buffer is merged into the secondary index trees in the database. Often, it is possible to merge several changes into the same page of the index tree, saving disk I/O operations. It has been measured that the insert buffer can speed up insertions into a table up to 15 times.

The insert buffer merging may continue to happen *after* the transaction has been committed. In fact, it may continue to happen after a server shutdown and restart (see [Section 14.3.7.2, “Forcing InnoDB Recovery”](#)).

Insert buffer merging may take many hours when many secondary indexes must be updated and many rows have been inserted. During this time, disk I/O will be increased, which can cause significant slowdown on disk-bound queries. Another significant background I/O operation is the purge thread (see [Section 14.3.10, “InnoDB Multi-Versioning”](#)).

14.3.11.4. Adaptive Hash Indexes

The feature known as the **adaptive hash index** (AHI) lets InnoDB perform more like an in-memory database on systems with appropriate combinations of workload and ample memory for the **buffer pool**, without sacrificing any transactional features or reliability.

Based on the observed pattern of searches, MySQL builds a hash index using a prefix of the index key. The prefix of the key can be any length, and it may be that only some of the values in the B-tree appear in the hash index. Hash indexes are built on demand for those pages of the index that are often accessed.

If a table fits almost entirely in main memory, a hash index can speed up queries by enabling direct lookup of any element, turning the index value into a sort of pointer. InnoDB has a mechanism that monitors index

searches. If InnoDB notices that queries could benefit from building a hash index, it does so automatically. This feature is enabled by the `innodb_adaptive_hash_index` [1547] option, or turned off by the `--skip-innodb_adaptive_hash_index` at server startup.

With some workloads, the speedup from hash index lookups greatly outweighs the extra work to monitor index lookups and maintain the hash index structure. Sometimes, the read/write lock that guards access to the adaptive hash index can become a source of contention under heavy workloads, such as multiple concurrent joins. Queries with `LIKE` operators and `%` wildcards also tend not to benefit from the AHI. For workloads where the adaptive hash index is not needed, turning it off reduces unnecessary performance overhead. Because it is difficult to predict in advance whether this feature is appropriate for a particular system, consider running benchmarks with it both enabled and disabled, using a realistic workload.

The hash index is always built based on an existing B-tree index on the table. InnoDB can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches that InnoDB observes for the B-tree index. A hash index can be partial, covering only those pages of the index that are often accessed.

You can monitor the use of the adaptive hash index and the contention for its use in the `SEMAPHORES` section of the output of the `SHOW ENGINE INNODB STATUS` command. If you see many threads waiting on an RW-latch created in `btr0sea.c`, then it might be useful to disable adaptive hash indexing.

For more information about the performance characteristics of hash indexes, see [Section 8.3.8, “Comparison of B-Tree and Hash Indexes”](#).

14.3.11.5. Physical Row Structure

The physical row structure for an InnoDB table depends on the row format specified when the table was created. InnoDB uses the `COMPACT` format by default, but the `REDUNDANT` format is available to retain compatibility with older versions of MySQL. To check the row format of an InnoDB table, use `SHOW TABLE STATUS`.

The compact row format decreases row storage space by about 20% at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed, compact format is likely to be faster. If the workload is a rare case that is limited by CPU speed, compact format might be slower.

Rows in InnoDB tables that use `REDUNDANT` row format have the following characteristics:

- Each index record contains a 6-byte header. The header is used to link together consecutive records, and also in row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key was defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record also contains all the primary key fields defined for the clustered index key that are not in the secondary index.
- A record contains a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of these pointers is called the record directory. The area where these pointers point is called the data part of the record.
- Internally, InnoDB stores fixed-length character columns such as `CHAR(10)` in a fixed-length format. InnoDB does not truncate trailing spaces from `VARCHAR` columns.
- An SQL `NULL` value reserves one or two bytes in the record directory. Besides that, an SQL `NULL` value reserves zero bytes in the data part of the record if stored in a variable length column. In a fixed-length

column, it reserves the fixed length of the column in the data part of the record. Reserving the fixed space for `NULL` values enables an update of the column from `NULL` to a non-`NULL` value to be done in place without causing fragmentation of the index page.

Rows in InnoDB tables that use `COMPACT` row format have the following characteristics:

- Each index record contains a 5-byte header that may be preceded by a variable-length header. The header is used to link together consecutive records, and also in row-level locking.
- The variable-length part of the record header contains a bit vector for indicating `NULL` columns. If the number of columns in the index that can be `NULL` is N , the bit vector occupies $\text{CEILING}(N/8)$ bytes. (For example, if there are anywhere from 9 to 15 columns that can be `NULL`, the bit vector uses two bytes.) Columns that are `NULL` do not occupy space other than the bit in this vector. The variable-length part of the header also contains the lengths of variable-length columns. Each length takes one or two bytes, depending on the maximum length of the column. If all columns in the index are `NOT NULL` and have a fixed length, the record header has no variable-length part.
- For each non-`NULL` variable-length field, the record header contains the length of the column in one or two bytes. Two bytes will only be needed if part of the column is stored externally in overflow pages or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes. For an externally stored column, the 2-byte length indicates the length of the internally stored part plus the 20-byte pointer to the externally stored part. The internal part is 768 bytes, so the length is 768+20. The 20-byte pointer stores the true length of the column.
- The record header is followed by the data contents of the non-`NULL` columns.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key was defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record also contains all the primary key fields defined for the clustered index key that are not in the secondary index. If any of these primary key fields are variable length, the record header for each secondary index will have a variable-length part to record their lengths, even if the secondary index is defined on fixed-length columns.
- Internally, InnoDB stores fixed-length, fixed-width character columns such as `CHAR(10)` in a fixed-length format. InnoDB does not truncate trailing spaces from `VARCHAR` columns.
- Internally, InnoDB attempts to store UTF-8 `CHAR(N)` columns in N bytes by trimming trailing spaces. (With `REDUNDANT` row format, such columns occupy $3 \times N$ bytes.) Reserving the minimum space N in many cases enables column updates to be done in place without causing fragmentation of the index page.

14.3.12. InnoDB Disk I/O and File Space Management

The laws of physics dictate that it takes time and effort to read and write data. The ACID design model requires a certain amount of I/O that might seem redundant, but helps to ensure data reliability. Within these constraints, InnoDB tries to optimize the database work and the organization of disk files to minimize the amount of disk I/O. Sometimes, I/O is postponed until the database is not busy, or until everything needs to be brought to a consistent state, such as during a database restart after a `fast shutdown`.

14.3.12.1. InnoDB Disk I/O

InnoDB uses asynchronous disk I/O where possible, by creating a number of threads to handle I/O operations, while permitting other database operations to proceed while the I/O is still in progress. On Linux and Windows platforms, InnoDB uses the available OS and library functions to perform “native”

asynchronous I/O. On other platforms, InnoDB still uses I/O threads, but the threads may actually wait for I/O requests to complete; this technique is known as “simulated” asynchronous I/O.

Read-Ahead

If InnoDB can determine there is a high probability that data might be needed soon, it performs read-ahead operations to bring that data into the buffer pool so that it is available in memory. Making a few large read requests for contiguous data can be more efficient than making several small, spread-out requests. There are two read-ahead heuristics in InnoDB:

- In sequential read-ahead, if InnoDB notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.
- In random read-ahead, if InnoDB notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

Doublewrite Buffer

InnoDB uses a novel file flush technique involving a structure called the [doublewrite buffer](#). It adds safety to recovery following an operating system crash or a power outage, and improves performance on most varieties of Unix by reducing the need for `fsync()` operations.

Before writing pages to a data file, InnoDB first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does InnoDB write the pages to their proper positions in the data file. If the operating system crashes in the middle of a page write, InnoDB can later find a good copy of the page from the doublewrite buffer during recovery.

14.3.12.2. File Space Management

The data files that you define in the configuration file form the [InnoDB system tablespace](#). The files are logically concatenated to form the tablespace. There is no striping in use. Currently, you cannot define where within the tablespace your tables are allocated. However, in a newly created tablespace, InnoDB allocates space starting from the first data file.

To avoid the issues that come with storing all tables and indexes inside the system tablespace, you can turn on the `innodb_file_per_table` configuration option, which stores each newly created table in a separate tablespace file (with extension `.ibd`). For tables stored this way, there is less fragmentation within the disk file, and when the table is truncated, the space is returned to the operating system rather than still being reserved by InnoDB within the system tablespace.

Pages, Extents, Segments, and Tablespaces

Each tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of size 1MB (64 consecutive pages). The “files” inside a tablespace are called *segments* in InnoDB. (These segments are different from the “rollback segment”, which actually contains many tablespace segments.)

When a segment grows inside the tablespace, InnoDB allocates the first 32 pages to it individually. After that, InnoDB starts to allocate whole extents to the segment. InnoDB can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

Two segments are allocated for each index in InnoDB. One is for nonleaf nodes of the B-tree, the other is for the leaf nodes. Keeping the leaf nodes contiguous on disk enables better sequential I/O operations, because these leaf nodes contain the actual table data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an InnoDB tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a `SHOW TABLE STATUS` statement, InnoDB reports the extents that are definitely free in the tablespace. InnoDB always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, InnoDB contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only in an (automatic) purge operation after they are no longer needed for transaction rollbacks or consistent reads. (See [Section 14.3.10, “InnoDB Multi-Versioning”](#).)

To see information about the tablespace, use the Tablespace Monitor. See [Section 14.3.14.2, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”](#).

How Pages Relate to Table Rows

The maximum row length, except for variable-length columns (`VARBINARY`, `VARCHAR`, `BLOB` and `TEXT`), is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. `LONGBLOB` and `LONGTEXT` columns must be less than 4GB, and the total row length, including `BLOB` and `TEXT` columns, must be less than 4GB.

If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page. For a column chosen for off-page storage, InnoDB stores the first 768 bytes locally in the row, and the rest externally into overflow pages. Each such column has its own list of overflow pages. The 768-byte prefix is accompanied by a 20-byte value that stores the true length of the column and points into the overflow list where the rest of the value is stored.

14.3.12.3. Defragmenting a Table

Random insertions into or deletions from a secondary index may cause the index to become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

One symptom of fragmentation is that a table takes more space than it “should” take. How much that is exactly, is difficult to determine. All InnoDB data and indexes are stored in B-trees, and their `fill factor` may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it “should” take:

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

The preceding query requires MySQL to scan the clustered index rather than a secondary index. Most disks can read 10MB/s to 50MB/s, which can be used to estimate how fast a table scan should be.

To speed up index scans, you can periodically perform a “null” `ALTER TABLE` operation, which causes MySQL to rebuild the table:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

Another way to perform a defragmentation operation is to use `mysqldump` to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions into an index are always ascending and records are deleted only from the end, the InnoDB filespace management algorithm guarantees that fragmentation in the index does not occur.

14.3.13. InnoDB Error Handling

Error handling in InnoDB is not always the same as specified in the SQL standard. According to the standard, any error during an SQL statement should cause rollback of that statement. InnoDB sometimes rolls back only part of the statement, or the whole transaction. The following items describe how InnoDB performs error handling:

- If you run out of file space in the tablespace, a MySQL `Table is full` error occurs and InnoDB rolls back the SQL statement.
- A transaction deadlock causes InnoDB to roll back the entire transaction. Retry the whole transaction when this happens.

A lock wait timeout causes InnoDB to roll back only the single statement that was waiting for the lock and encountered the timeout. (To have the entire transaction roll back, start the server with the `--innodb_rollback_on_timeout [1569]` option.) Retry the statement if using the current behavior, or the entire transaction if using `--innodb_rollback_on_timeout [1569]`.

Both deadlocks and lock wait timeouts are normal on busy servers and it is necessary for applications to be aware that they may happen and handle them by retrying. You can make them less likely by doing as little work as possible between the first change to data during a transaction and the commit, so the locks are held for the shortest possible time and for the smallest possible number of rows. Sometimes splitting work between different transactions may be practical and helpful.

When a transaction rollback occurs due to a deadlock or lock wait timeout, it cancels the effect of the statements within the transaction. But if the start-transaction statement was `START TRANSACTION` or `BEGIN` statement, rollback does not cancel that statement. Further SQL statements become part of the transaction until the occurrence of `COMMIT`, `ROLLBACK`, or some SQL statement that causes an implicit commit.

- A duplicate-key error rolls back the SQL statement, if you have not specified the `IGNORE` option in your statement.
- A `row too long error` rolls back the SQL statement.
- Other errors are mostly detected by the MySQL layer of code (above the InnoDB storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During implicit rollbacks, as well as during the execution of an explicit `ROLLBACK` SQL statement, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the relevant connection.

14.3.13.1. InnoDB Error Codes

The following is a nonexhaustive list of common InnoDB-specific errors that you may encounter, with information about why each occurs and how to resolve the problem.

- `1005 (ER_CANT_CREATE_TABLE)`

Cannot create table. If the error message refers to error 150, table creation failed because a foreign key constraint was not correctly formed. If the error message refers to error -1, table creation probably failed because the table includes a column name that matched the name of an internal InnoDB table.

- `1016 (ER_CANT_OPEN_FILE)`

Cannot find the InnoDB table from the InnoDB data files, although the `.frm` file for the table exists. See [Section 14.3.14.4, “Troubleshooting InnoDB Data Dictionary Operations”](#).

- `1114 (ER_RECORD_FILE_FULL)`

InnoDB has run out of free space in the tablespace. Reconfigure the tablespace to add a new data file.

- [1205 \(ER_LOCK_WAIT_TIMEOUT\)](#)

Lock wait timeout expired. The statement that waited too long was **rolled back** (not the entire **transaction**). You can increase the value of the `innodb_lock_wait_timeout` [1560] configuration option if SQL statements should wait longer for other transactions to complete, or decrease it if too many long-running transactions are causing **locking** problems and reducing **concurrency** on a busy system.

- [1206 \(ER_LOCK_TABLE_FULL\)](#)

The total number of locks exceeds the lock table size. To avoid this error, increase the value of `innodb_buffer_pool_size` [1549]. Within an individual application, a workaround may be to break a large operation into smaller pieces. For example, if the error occurs for a large `INSERT`, perform several smaller `INSERT` operations.

- [1213 \(ER_LOCK_DEADLOCK\)](#)

The **transaction** encountered a **deadlock** and was automatically **rolled back** so that your application could take corrective action. To recover from this error, run all the operations in this transaction again. A deadlock occurs when requests for locks arrive in inconsistent order between transactions. The transaction that was rolled back released all its locks, and the other transaction can now get all the locks it requested. Thus when you re-run the transaction that was rolled back, it might have to wait for other transactions to complete, but typically the deadlock does not recur. If you encounter frequent deadlocks, make the sequence of locking operations (`LOCK TABLES`, `SELECT ... FOR UPDATE`, and so on) consistent between the different transactions or applications that experience the issue. See [Section 14.3.9.9, “How to Cope with Deadlocks”](#) for details.

- [1216 \(ER_NO_REFERENCED_ROW\)](#)

You are trying to add a row but there is no parent row, and a foreign key constraint fails. Add the parent row first.

- [1217 \(ER_ROW_IS_REFERENCED\)](#)

You are trying to delete a parent row that has children, and a foreign key constraint fails. Delete the children first.

14.3.13.2. Operating System Error Codes

To print the meaning of an operating system error number, use the `perror` program that comes with the MySQL distribution.

- **Linux System Error Codes**

The following table provides a list of some common Linux system error codes. For a more complete list, see [Linux source code](#).

Number	Macro	Description
1	<code> EPERM </code>	Operation not permitted
2	<code> ENOENT </code>	No such file or directory
3	<code> ESRCH </code>	No such process
4	<code> EINTR </code>	Interrupted system call
5	<code> EIO </code>	I/O error

Number	Macro	Description
6	ENXIO	No such device or address
7	E2BIG	Arg list too long
8	ENOEXEC	Exec format error
9	EBADF	Bad file number
10	ECHILD	No child processes
11	EAGAIN	Try again
12	ENOMEM	Out of memory
13	EACCES	Permission denied
14	EFAULT	Bad address
15	ENOTBLK	Block device required
16	EBUSY	Device or resource busy
17	EEXIST	File exists
18	EXDEV	Cross-device link
19	ENODEV	No such device
20	ENOTDIR	Not a directory
21	EISDIR	Is a directory
22	EINVAL	Invalid argument
23	ENFILE	File table overflow
24	EMFILE	Too many open files
25	ENOTTY	Inappropriate ioctl for device
26	ETXTBSY	Text file busy
27	EFBIG	File too large
28	ENOSPC	No space left on device
29	ESPIPE	File descriptor does not allow seeking
30	EROFS	Read-only file system
31	EMLINK	Too many links

- **Windows System Error Codes**

The following table provides a list of some common Windows system error codes. For a complete list, see the [Microsoft Web site](#).

Number	Macro	Description
1	ERROR_INVALID_FUNCTION	Incorrect function.
2	ERROR_FILE_NOT_FOUND	The system cannot find the file specified.
3	ERROR_PATH_NOT_FOUND	The system cannot find the path specified.
4	ERROR_TOO_MANY_OPEN_FILES	The system cannot open the file.
5	ERROR_ACCESS_DENIED	Access is denied.
6	ERROR_INVALID_HANDLE	The handle is invalid.
7	ERROR_ARENA_TRASHED	The storage control blocks were destroyed.

Number	Macro	Description
8	ERROR_NOT_ENOUGH_DISK	Not enough storage is available to process this command.
9	ERROR_INVALID_BLOCK	The storage control block address is invalid.
10	ERROR_BAD_ENVIRONMENT	The environment is incorrect.
11	ERROR_BAD_FORMAT	An attempt was made to load a program with an incorrect format.
12	ERROR_INVALID_ACCESS	The access code is invalid.
13	ERROR_INVALID_DATA	The data is invalid.
14	ERROR_OUTOFMEMORY	Not enough storage is available to complete this operation.
15	ERROR_INVALID_DRIVE	The system cannot find the drive specified.
16	ERROR_CURRENT_DIRECTORY	The directory cannot be removed.
17	ERROR_NOT_SAME_DRIVE	The system cannot move the file to a different disk drive.
18	ERROR_NO_MORE_FILES	There are no more files.
19	ERROR_WRITE_PROTECT	The media is write protected.
20	ERROR_BAD_UNIT	The system cannot find the device specified.
21	ERROR_NOT_READY	The device is not ready.
22	ERROR_BAD_COMMAND	The device does not recognize the command.
23	ERROR_CRC	Data error (cyclic redundancy check).
24	ERROR_BAD_LENGTH	The program issued a command but the command length is incorrect.
25	ERROR_SEEK	The drive cannot locate a specific area or track on the disk.
26	ERROR_NOT_DOS_DISK	The specified disk or diskette cannot be accessed.
27	ERROR_SECTOR_NOT_FOUND	The drive cannot find the sector requested.
28	ERROR_OUT_OF_PAPER	The printer is out of paper.
29	ERROR_WRITE_FAULT	The system cannot write to the specified device.
30	ERROR_READ_FAULT	The system cannot read from the specified device.
31	ERROR_GEN_FAILURE	A device attached to the system is not functioning.
32	ERROR_SHARING_VIOLATION	The process cannot access the file because it is being used by another process.
33	ERROR_LOCK_VIOLATION	The process cannot access the file because another process has locked a portion of the file.
34	ERROR_WRONG_DISK	The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.
36	ERROR_SHARING_BUFFER_EXCEEDED	Too many files opened for sharing.
38	ERROR_HANDLE_EOF	Reached the end of the file.
39	ERROR_HANDLE_DISK_FULL	The disk is full.
87	ERROR_INVALID_PARAMETER	The parameter is incorrect.
112	ERROR_DISK_FULL	The disk is full.
123	ERROR_INVALID_NAME	The file name, directory name, or volume label syntax is incorrect.
1450	ERROR_NO_SYSTEM_RESOURCES	Insufficient system resources exist to complete the requested service.

14.3.14. InnoDB Performance Tuning and Troubleshooting

14.3.14.1. InnoDB Performance Tuning Tips

With InnoDB becoming the default storage engine in MySQL 5.5 and higher, the tips and guidelines for InnoDB tables are now part of the main optimization chapter. See [Section 8.5, “Optimizing for InnoDB Tables”](#).

14.3.14.2. SHOW ENGINE INNODB STATUS and the InnoDB Monitors

InnoDB Monitors provide information about the InnoDB internal state. This information is useful for performance tuning. Each Monitor can be enabled by creating a table with a special name, which causes InnoDB to write Monitor output periodically. Also, output for the standard InnoDB Monitor is available on demand through the `SHOW ENGINE INNODB STATUS` SQL statement.

There are several types of InnoDB Monitors:

- The standard InnoDB Monitor displays the following types of information:
 - Table and record locks held by each active transaction.
 - Lock waits of a transaction.
 - Semaphore waits of threads.
 - Pending file I/O requests.
 - Buffer pool statistics.
 - Purge and insert buffer merge activity of the main InnoDB thread.

For a discussion of InnoDB lock modes, see [Section 14.3.9.1, “InnoDB Lock Modes”](#).

To enable the standard InnoDB Monitor for periodic output, create a table named `innodb_monitor`. To obtain Monitor output on demand, use the `SHOW ENGINE INNODB STATUS` SQL statement to fetch the output to your client program. If you are using the `mysql` interactive client, the output is more readable if you replace the usual semicolon statement terminator with `\G`:

```
mysql> SHOW ENGINE INNODB STATUS\G
```

- The InnoDB Lock Monitor is like the standard Monitor but also provides extensive lock information. To enable this Monitor for periodic output, create a table named `innodb_lock_monitor`.
- The InnoDB Tablespace Monitor prints a list of file segments in the shared tablespace and validates the tablespace allocation data structures. To enable this Monitor for periodic output, create a table named `innodb_tablespace_monitor`.
- The InnoDB Table Monitor prints the contents of the InnoDB internal data dictionary. To enable this Monitor for periodic output, create a table named `innodb_table_monitor`.

To enable an InnoDB Monitor for periodic output, use a `CREATE TABLE` statement to create the table associated with the Monitor. For example, to enable the standard InnoDB Monitor, create the `innodb_monitor` table:

```
CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
```

To stop the Monitor, drop the table:

```
DROP TABLE innodb_monitor;
```

The `CREATE TABLE` syntax is just a way to pass a command to the InnoDB engine through MySQL's SQL parser: The only things that matter are the table name `innodb_monitor` and that it be an InnoDB table. The structure of the table is not relevant at all for the InnoDB Monitor. If you shut down the server, the Monitor does not restart automatically when you restart the server. Drop the Monitor table and issue a new `CREATE TABLE` statement to start the Monitor. (This syntax may change in a future release.)

The `PROCESS` [696] privilege is required to start or stop the InnoDB Monitor tables.

When you enable InnoDB Monitors for periodic output, InnoDB writes their output to the `mysqld` server standard error output (`stderr`). In this case, no output is sent to clients. When switched on, InnoDB Monitors print data about every 15 seconds. Server output usually is directed to the error log (see [Section 5.2.2, "The Error Log"](#)). This data is useful in performance tuning. On Windows, start the server from a command prompt in a console window with the `--console` [438] option if you want to direct the output to the window rather than to the error log.

InnoDB sends diagnostic output to `stderr` or to files rather than to `stdout` or fixed-size memory buffers, to avoid potential buffer overflows. As a side effect, the output of `SHOW ENGINE INNODB STATUS` is written to a status file in the MySQL data directory every fifteen seconds. The name of the file is `innodb_status.pid`, where `pid` is the server process ID. InnoDB removes the file for a normal shutdown. If abnormal shutdowns have occurred, instances of these status files may be present and must be removed manually. Before removing them, you might want to examine them to see whether they contain useful information about the cause of abnormal shutdowns. The `innodb_status.pid` file is created only if the configuration option `innodb-status-file=1` [1546] is set.

InnoDB Monitors should be enabled only when you actually want to see Monitor information because output generation does result in some performance decrement. Also, if you enable monitor output by creating the associated table, your error log may become quite large if you forget to remove the table later.

For additional information about InnoDB monitors, see:

- Mark Leith: [InnoDB Table and Tablespace Monitors](#)

Each monitor begins with a header containing a timestamp and the monitor name. For example:

```
=====
090407 12:06:19 INNODB TABLESPACE MONITOR OUTPUT
=====
```

The header for the standard Monitor (`INNODB MONITOR OUTPUT`) is also used for the Lock Monitor because the latter produces the same output with the addition of extra lock information.

The following sections describe the output for each Monitor.

14.3.14.2.1. InnoDB Standard Monitor and Lock Monitor Output

The Lock Monitor is the same as the standard Monitor except that it includes additional lock information. Enabling either monitor for periodic output by creating the associated InnoDB table turns on the same output stream, but the stream includes the extra information if the Lock Monitor is enabled. For example, if you create the `innodb_monitor` and `innodb_lock_monitor` tables, that turns on a single output stream. The stream includes extra lock information until you disable the Lock Monitor by removing the `innodb_lock_monitor` table.

Example InnoDB Monitor output:

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
Status:
=====
030709 13:00:59 INNODB MONITOR OUTPUT
=====
```

```

Per second averages calculated from the last 18 seconds
-----
BACKGROUND THREAD
-----
srv_master_thread loops: 53 1_second, 44 sleeps, 5 10_second, 7 background,
  7 flush
srv_master_thread log flush and writes: 48
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds the
semaphore: X-lock on RW-latch at 41a28668 created in file btr0sea.c line 135
a writer (thread id 32782) has reserved it in mode wait exclusive
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
Last time write locked in file btr0sea.c line 1347
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 2, rounds 60, OS waits 2
RW-excl spins 0, rounds 0, OS waits 0
Spin rounds per wait: 0.00 mutex, 20.00 RW-shared, 0.00 RW-excl
-----
LATEST FOREIGN KEY ERROR
-----
030709 13:00:59 Transaction:
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195
inserting
15 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' , 'khDk')
Foreign key constraint fails for table test/ibtest11a:
'
  CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b` (`A`,
  `D`) ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index PRIMARY tuple:
  0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....;; 2:
  len 4; hex 6b68446b; asc khDk;; 3: len 6; hex 0000114e0edc; asc ..N..;; 4:
  len 7; hex 00000000c3e0a7; asc .....;; 5: len 4; hex 6b68446b; asc khDk;;
But in parent table test/ibtest11b, in index PRIMARY,
the closest match we can find is record:
RECORD: info bits 0 0: len 4; hex 8000015b; asc ...[;; 1: len 4; hex
80000005; asc ....;; 2: len 3; hex 6b6864; asc khD; 3: len 6; hex
0000111ef3eb; asc .....;; 4: len 7; hex 800001001e0084; asc .....;; 5:
len 3; hex 6b6864; asc khD;;
-----
LATEST DETECTED DEADLOCK
-----
030709 12:59:58
*** (1) TRANSACTION:
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185
inserting
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146
MySQL thread id 21, query id 4553379 localhost heikki update
INSERT INTO alex1 VALUES(86, 86, 794,'aa35818','bb','c79166','d4766t',
'e187358f','g84586','h794',date_format('2001-04-03 12:54:22','%Y-%m-%d
%H:%i'),7
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index
symbole trx id 0 290252780 lock mode S waiting
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138;
asc aa35818;; 1:
*** (2) TRANSACTION:
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190
inserting
130 lock struct(s), heap size 11584, undo log entries 437
MySQL thread id 23, query id 4554396 localhost heikki update
REPLACE INTO alex1 VALUES(NULL, 32, NULL,'aa3572','','c3572','d6012t','','

```



```

FILE I/O
-----
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
  ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
151671 OS file reads, 94747 OS file writes, 8750 OS fsyncs
25.44 reads/s, 18494 avg bytes/read, 17.55 writes/s, 2.33 fsyncs/s
-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
Ibuf for space 0: size 1, free list len 19, seg size 21,
85004 inserts, 85004 merged recs, 26669 merges
Hash table size 207619, used cells 14461, node heap has 16 buffer(s)
1877.67 hash searches/s, 5121.10 non-hash searches/s
---
LOG
---
Log sequence number 18 1212842764
Log flushed up to 18 1212665295
Last checkpoint at 18 1135877290
0 pending log writes, 0 pending chkp writes
4341 log i/o's done, 1.22 log i/o's/second
-----
BUFFER POOL AND MEMORY
-----
Total memory allocated 84966343; in additional pool allocated 1402624
Buffer pool size 3200
Free buffers 110
Database pages 3074
Modified db pages 2674
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 171380, created 51968, written 194688
28.72 reads/s, 20.72 creates/s, 47.55 writes/s
Buffer pool hit rate 999 / 1000
-----
ROW OPERATIONS
-----
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
-----
END OF INNODB MONITOR OUTPUT
=====

```

InnoDB Monitor output is limited to 1MB when produced using the `SHOW ENGINE INNODB STATUS` statement. This limit does not apply to output written to the server's error output.

Some notes on the output sections:

BACKGROUND THREAD

The `srv_master_thread` lines shows work done by the main background thread.

SEMAPHORES

This section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside InnoDB. Contention can be due to heavy parallelism of queries or problems in operating system thread scheduling. Setting the

`innodb_thread_concurrency` [1573] system variable smaller than the default value might help in such situations. The `Spin rounds per wait` line shows the number of spinlock rounds per OS wait for a mutex.

LATEST FOREIGN KEY ERROR

This section provides information about the most recent foreign key constraint error. It is not present if no such error has occurred. The contents include the statement that failed as well as information about the constraint that failed and the referenced and referencing tables.

LATEST DETECTED DEADLOCK

This section provides information about the most recent deadlock. It is not present if no deadlock has occurred. The contents show which transactions are involved, the statement each was attempting to execute, the locks they have and need, and which transaction InnoDB decided to roll back to break the deadlock. The lock modes reported in this section are explained in [Section 14.3.9.1, “InnoDB Lock Modes”](#).

TRANSACTIONS

If this section reports lock waits, your applications might have lock contention. The output can also help to trace the reasons for transaction deadlocks.

FILE I/O

This section provides information about threads that InnoDB uses to perform various types of I/O. The first few of these are dedicated to general InnoDB processing. The contents also display information for pending I/O operations and statistics for I/O performance.

The number of these threads are controlled by the `innodb_read_io_threads` [1568] and `innodb_write_io_threads` [1575] parameters. See [Section 14.3.4, “InnoDB Startup Options and System Variables”](#).

INSERT BUFFER AND ADAPTIVE HASH INDEX

This section shows the status of the InnoDB insert buffer and adaptive hash index. (See [Section 14.3.11.3, “Insert Buffering”](#), and [Section 14.3.11.4, “Adaptive Hash Indexes”](#).) The contents include the number of operations performed for each, plus statistics for hash index performance.

LOG

This section displays information about the InnoDB log. The contents include the current log sequence number, how far the log has been flushed to disk, and the position at which InnoDB last took a checkpoint. (See [Section 14.3.7.3, “InnoDB Checkpoints”](#).) The section also displays information about pending writes and write performance statistics.

BUFFER POOL AND MEMORY

This section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.

For additional information about the operation of the buffer pool, see [Section 8.9.1, “The InnoDB Buffer Pool”](#).

ROW OPERATIONS

This section shows what the main thread is doing, including the number and performance rate for each type of row operation.

In MySQL 5.5, output from the standard Monitor includes additional sections compared to the output for previous versions. For details, see [Section 1.4.3, “Diagnostic and Monitoring Capabilities”](#).

14.3.14.2.2. InnoDB Tablespace Monitor Output

The InnoDB Tablespace Monitor prints information about the file segments in the shared tablespace and validates the tablespace allocation data structures. If you use individual tablespaces by enabling `innodb_file_per_table` [1555], the Tablespace Monitor does not describe those tablespaces.

Example InnoDB Tablespace Monitor output:

```
=====
090408 21:28:09 INNODB TABLESPACE MONITOR OUTPUT
=====
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
SEGMENT id 0 2 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 3 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
SEGMENT id 0 488 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 17 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 171 space 0; page 2; res 592 used 481; full ext 7
fragm pages 16; free extents 0; not full extents 2: pages 17
SEGMENT id 0 172 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 173 space 0; page 2; res 96 used 44; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 12
...
SEGMENT id 0 601 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
NUMBER of file segments: 73
Validating tablespace
Validation ok
-----
END OF INNODB TABLESPACE MONITOR OUTPUT
=====
```

The Tablespace Monitor output includes information about the shared tablespace as a whole, followed by a list containing a breakdown for each segment within the tablespace.

The tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of size 1MB (64 consecutive pages).

The initial part of the output that displays overall tablespace information has this format:

```
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
```

Overall tablespace information includes these values:

- `id`: The tablespace ID. A value of 0 refers to the shared tablespace.

- `size`: The current tablespace size in pages.
- `free limit`: The minimum page number for which the free list has not been initialized. Pages at or above this limit are free.
- `free extents`: The number of free extents.
- `not full frag extents`, `used pages`: The number of fragment extents that are not completely filled, and the number of pages in those extents that have been allocated.
- `full frag extents`: The number of completely full fragment extents.
- `first seg id not used`: The first unused segment ID.

Individual segment information has this format:

```
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
```

Segment information includes these values:

`id`: The segment ID.

`space`, `page`: The tablespace number and page within the tablespace where the segment “inode” is located. A tablespace number of 0 indicates the shared tablespace. InnoDB uses inodes to keep track of segments in the tablespace. The other fields displayed for a segment (`id`, `res`, and so forth) are derived from information in the inode.

`res`: The number of pages allocated (reserved) for the segment.

`used`: The number of allocated pages in use by the segment.

`full ext`: The number of extents allocated for the segment that are completely used.

`fragm pages`: The number of initial pages that have been allocated to the segment.

`free extents`: The number of extents allocated for the segment that are completely unused.

`not full extents`: The number of extents allocated for the segment that are partially used.

`pages`: The number of pages used within the not-full extents.

When a segment grows, it starts as a single page, and InnoDB allocates the first pages for it individually, up to 32 pages (this is the `fragm pages` value). After that, InnoDB allocates complete 64-page extents. InnoDB can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

For the example segment shown earlier, it has 32 fragment pages, plus 2 full extents (64 pages each), for a total of 160 pages used out of 160 pages allocated. The following segment has 32 fragment pages and one partially full extent using 14 pages for a total of 46 pages used out of 96 pages allocated:

```
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
```

It is possible for a segment that has extents allocated to it to have a `fragm pages` value less than 32 if some of the individual pages have been deallocated subsequent to extent allocation.

14.3.14.2.3. InnoDB Table Monitor Output

The InnoDB Table Monitor prints the contents of the InnoDB internal data dictionary.

The output contains one section per table. The `SYS_FOREIGN` and `SYS_FOREIGN_COLS` sections are for internal data dictionary tables that maintain information about foreign keys. There are also sections for the

Table Monitor table and each user-created InnoDB table. Suppose that the following two tables have been created in the `test` database:

```
CREATE TABLE parent
(
  par_id    INT NOT NULL,
  fname    CHAR(20),
  lname    CHAR(20),
  PRIMARY KEY (par_id),
  UNIQUE INDEX (lname, fname)
) ENGINE = INNODB;

CREATE TABLE child
(
  par_id    INT NOT NULL,
  child_id  INT NOT NULL,
  name      VARCHAR(40),
  birth     DATE,
  weight    DECIMAL(10,2),
  misc_info VARCHAR(255),
  last_update TIMESTAMP,
  PRIMARY KEY (par_id, child_id),
  INDEX (name),
  FOREIGN KEY (par_id) REFERENCES parent (par_id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
) ENGINE = INNODB;
```

Then the Table Monitor output will look something like this (reformatted slightly):

```
=====
090420 12:09:32 INNODB TABLE MONITOR OUTPUT
=====
-----
TABLE: name SYS_FOREIGN, id 0 11, columns 7, indexes 3, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0;
            FOR_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
            REF_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
            N_COLS: DATA_INT len 4;
            DB_ROW_ID: DATA_SYS prtype 256 len 6;
            DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name ID_IND, id 0 11, fields 1/6, uniq 1, type 3
        root page 46, appr.key vals 1, leaf pages 1, size pages 1
        FIELDS:  ID DB_TRX_ID DB_ROLL_PTR FOR_NAME REF_NAME N_COLS
  INDEX: name FOR_IND, id 0 12, fields 1/2, uniq 2, type 0
        root page 47, appr.key vals 1, leaf pages 1, size pages 1
        FIELDS:  FOR_NAME ID
  INDEX: name REF_IND, id 0 13, fields 1/2, uniq 2, type 0
        root page 48, appr.key vals 1, leaf pages 1, size pages 1
        FIELDS:  REF_NAME ID
-----
TABLE: name SYS_FOREIGN_COLS, id 0 12, columns 7, indexes 1, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0;
            POS: DATA_INT len 4;
            FOR_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
            REF_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
            DB_ROW_ID: DATA_SYS prtype 256 len 6;
            DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name ID_IND, id 0 14, fields 2/6, uniq 2, type 3
        root page 49, appr.key vals 1, leaf pages 1, size pages 1
        FIELDS:  ID POS DB_TRX_ID DB_ROLL_PTR FOR_COL_NAME REF_COL_NAME
-----
TABLE: name test/child, id 0 14, columns 10, indexes 2, appr.rows 201
  COLUMNS: par_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
            child_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
            name: DATA_VARCHAR prtype 524303 len 40;
            birth: DATA_INT DATA_BINARY_TYPE len 3;
```

```

weight: DATA_FIXBINARY DATA_BINARY_TYPE len 5;
misc_info: DATA_VARCHAR prtype 524303 len 255;
last_update: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 4;
DB_ROW_ID: DATA_SYS prtype 256 len 6;
DB_TRX_ID: DATA_SYS prtype 257 len 6;
INDEX: name PRIMARY, id 0 17, fields 2/9, uniq 2, type 3
root page 52, appr.key vals 201, leaf pages 5, size pages 6
FIELDS: par_id child_id DB_TRX_ID DB_ROLL_PTR name birth weight misc_info last_update
INDEX: name name, id 0 18, fields 1/3, uniq 3, type 0
root page 53, appr.key vals 210, leaf pages 1, size pages 1
FIELDS: name par_id child_id
FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
REFERENCES test/parent ( par_id )
-----
TABLE: name test/innodb_table_monitor, id 0 15, columns 4, indexes 1, appr.rows 0
COLUMNS: i: DATA_INT DATA_BINARY_TYPE len 4;
DB_ROW_ID: DATA_SYS prtype 256 len 6;
DB_TRX_ID: DATA_SYS prtype 257 len 6;
INDEX: name GEN_CLUST_INDEX, id 0 19, fields 0/4, uniq 1, type 1
root page 193, appr.key vals 0, leaf pages 1, size pages 1
FIELDS: DB_ROW_ID DB_TRX_ID DB_ROLL_PTR i
-----
TABLE: name test/parent, id 0 13, columns 6, indexes 2, appr.rows 299
COLUMNS: par_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
fname: DATA_CHAR prtype 524542 len 20;
lname: DATA_CHAR prtype 524542 len 20;
DB_ROW_ID: DATA_SYS prtype 256 len 6;
DB_TRX_ID: DATA_SYS prtype 257 len 6;
INDEX: name PRIMARY, id 0 15, fields 1/5, uniq 1, type 3
root page 50, appr.key vals 299, leaf pages 2, size pages 3
FIELDS: par_id DB_TRX_ID DB_ROLL_PTR fname lname
INDEX: name lname, id 0 16, fields 2/3, uniq 2, type 2
root page 51, appr.key vals 300, leaf pages 1, size pages 1
FIELDS: lname fname par_id
FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
REFERENCES test/parent ( par_id )
-----
END OF INNODB TABLE MONITOR OUTPUT
=====

```

For each table, Table Monitor output contains a section that displays general information about the table and specific information about its columns, indexes, and foreign keys.

The general information for each table includes the table name (in *db_name/tbl_name* format except for internal tables), its ID, the number of columns and indexes, and an approximate row count.

The **COLUMNS** part of a table section lists each column in the table. Information for each column indicates its name and data type characteristics. Some internal columns are added by InnoDB, such as **DB_ROW_ID** (row ID), **DB_TRX_ID** (transaction ID), and **DB_ROLL_PTR** (a pointer to the rollback/undo data).

- **DATA_xxx**: These symbols indicate the data type. There may be multiple **DATA_xxx** symbols for a given column.
- **prtype**: The column's “precise” type. This field includes information such as the column data type, character set code, nullability, signedness, and whether it is a binary string. This field is described in the [innobase/include/data0type.h](#) source file.
- **len**: The column length in bytes.

Each **INDEX** part of the table section provides the name and characteristics of one table index:

- **name**: The index name. If the name is **PRIMARY**, the index is a primary key. If the name is **GEN_CLUST_INDEX**, the index is the clustered index that is created automatically if the table definition

doesn't include a primary key or non-NULL unique index. See [Section 14.3.11.1, “Clustered and Secondary Indexes”](#).

- `id`: The index ID.
- `fields`: The number of fields in the index, as a value in *m/n* format:
 - *m* is the number of user-defined columns; that is, the number of columns you would see in the index definition in a `CREATE TABLE` statement.
 - *n* is the total number of index columns, including those added internally. For the clustered index, the total includes the other columns in the table definition, plus any columns added internally. For a secondary index, the total includes the columns from the primary key that are not part of the secondary index.
- `uniq`: The number of leading fields that are enough to determine index values uniquely.
- `type`: The index type. This is a bit field. For example, 1 indicates a clustered index and 2 indicates a unique index, so a clustered index (which always contains unique values), will have a `type` value of 3. An index with a `type` value of 0 is neither clustered nor unique. The flag values are defined in the `innobase/include/dict0mem.h` source file.
- `root page`: The index root page number.
- `appr. key vals`: The approximate index cardinality.
- `leaf pages`: The approximate number of leaf pages in the index.
- `size pages`: The approximate total number of pages in the index.
- `FIELDS`: The names of the fields in the index. For a clustered index that was generated automatically, the field list begins with the internal `DB_ROW_ID` (row ID) field. `DB_TRX_ID` and `DB_ROLL_PTR` are always added internally to the clustered index, following the fields that comprise the primary key. For a secondary index, the final fields are those from the primary key that are not part of the secondary index.

The end of the table section lists the `FOREIGN KEY` definitions that apply to the table. This information appears whether the table is a referencing or referenced table.

14.3.14.3. InnoDB General Troubleshooting

The following general guidelines apply to troubleshooting InnoDB problems:

- When an operation fails or you suspect a bug, look at the MySQL server error log (see [Section 5.2.2, “The Error Log”](#)).
- Issues relating to the InnoDB data dictionary include failed `CREATE TABLE` statements (orphaned table files), inability to open `.InnoDB` files, and `system cannot find the path specified` errors. For information about these sorts of problems and errors, see [Section 14.3.14.4, “Troubleshooting InnoDB Data Dictionary Operations”](#).
- When troubleshooting, it is usually best to run the MySQL server from the command prompt, rather than through `mysqld_safe` or as a Windows service. You can then see what `mysqld` prints to the console, and so have a better grasp of what is going on. On Windows, start `mysqld` with the `--console` [\[438\]](#) option to direct the output to the console window.
- Use the InnoDB Monitors to obtain information about a problem (see [Section 14.3.14.2, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”](#)). If the problem is performance-related, or your

server appears to be hung, you should use the standard Monitor to print information about the internal state of [InnoDB](#). If the problem is with locks, use the Lock Monitor. If the problem is in creation of tables or other data dictionary operations, use the Table Monitor to print the contents of the [InnoDB](#) internal data dictionary. To see tablespace information use the Tablespace Monitor.

- If you suspect that a table is corrupt, run `CHECK TABLE` on that table.

14.3.14.4. Troubleshooting [InnoDB](#) Data Dictionary Operations

Information about table definitions is stored both in the `.frm` files, and in the [InnoDB data dictionary](#). If you move `.frm` files around, or if the server crashes in the middle of a data dictionary operation, these sources of information can become inconsistent.

Problem with CREATE TABLE

A symptom of an out-of-sync data dictionary is that a `CREATE TABLE` statement fails. If this occurs, look in the server's error log. If the log says that the table already exists inside the [InnoDB](#) internal data dictionary, you have an orphaned table inside the [InnoDB](#) tablespace files that has no corresponding `.frm` file. The error message looks like this:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

You can drop the orphaned table by following the instructions given in the error message. If you are still unable to use `DROP TABLE` successfully, the problem may be due to name completion in the `mysql` client. To work around this problem, start the `mysql` client with the `--skip-auto-rehash` [276] option and try `DROP TABLE` again. (With name completion on, `mysql` tries to construct a list of table names, which fails when a problem such as just described exists.)

Problem Opening Table

Another symptom of an out-of-sync data dictionary is that MySQL prints an error that it cannot open a `.InnoDB` file:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

In the error log you can find a message like this:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

This means that there is an orphaned `.frm` file without a corresponding table inside [InnoDB](#). You can drop the orphaned `.frm` file by deleting it manually.

Problem with Temporary Table

If MySQL crashes in the middle of an `ALTER TABLE` operation, you may end up with an orphaned temporary table inside the [InnoDB](#) tablespace. Using the Table Monitor, you can see listed a table with a name that begins with `#sql-`. You can perform SQL statements on tables whose name contains the character `#` if you enclose the name within backticks. Thus, you can drop such an orphaned table like any

other orphaned table using the method described earlier. To copy or rename a file in the Unix shell, you need to put the file name in double quotation marks if the file name contains “#”.

Problem with Missing Tablespace

With `innodb_file_per_table` [1555] enabled, the following message might occur if the `.frm` or `.ibd` files (or both) are missing:

```
InnoDB: in InnoDB data dictionary has tablespace id N,
InnoDB: but tablespace with that id or name does not exist. Have
InnoDB: you deleted or moved .ibd files?
InnoDB: This may also be a table created with CREATE TEMPORARY TABLE
InnoDB: whose .ibd and .frm files MySQL automatically removed, but the
InnoDB: table still exists in the InnoDB internal data dictionary.
```

If this occurs, try the following procedure to resolve the problem:

1. Create a matching `.frm` file in some other database directory and copy it to the database directory where the orphan table is located.
2. Issue `DROP TABLE` for the original table. That should successfully drop the table and InnoDB should print a warning to the error log that the `.ibd` file was missing.

14.3.15. Limits on InnoDB Tables



Warning

Do *not* convert MySQL system tables in the `mysql` database from MyISAM to InnoDB tables! This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or re-generate them with the `mysql_install_db` script.



Warning

It is not a good idea to configure InnoDB to use data files or log files on NFS volumes. Otherwise, the files might be locked by other processes and become unavailable for use by MySQL.

Maximums and Minimums

- A table can contain a maximum of 1000 columns.
- A table can contain a maximum of 64 [secondary indexes](#).
- By default, an index key for a single-column index can be up to 767 bytes. The same length limit applies to any index key prefix. See [Section 13.1.13, “CREATE INDEX Syntax”](#). For example, you might hit this limit with a [column prefix](#) index of more than 255 characters on a `TEXT` or `VARCHAR` column, assuming a UTF-8 character set and the maximum of 3 bytes for each character. When the [innodb_large_prefix](#) [1560] configuration option is enabled, this length limit is raised to 3072 bytes, for InnoDB tables that use the `DYNAMIC` and `COMPRESSED` row formats.

When you attempt to specify an index prefix length longer than allowed, the length is silently reduced to the maximum length. This configuration option changes the error handling for some combinations of row format and prefix length longer than the maximum allowed. See [innodb_large_prefix](#) [1560] for details.

- The InnoDB internal maximum key length is 3500 bytes, but MySQL itself restricts this to 3072 bytes. This limit applies to the length of the combined index key in a multi-column index.

- The maximum row length, except for variable-length columns (`VARBINARY`, `VARCHAR`, `BLOB` and `TEXT`), is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. `LONGBLOB` and `LONGTEXT` columns must be less than 4GB, and the total row length, including `BLOB` and `TEXT` columns, must be less than 4GB.

If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page, as described in [Section 14.3.12.2, “File Space Management”](#).

- Although InnoDB supports row sizes larger than 65,535 bytes internally, MySQL itself imposes a row-size limit of 65,535 for the combined size of all columns:

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),
  -> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
  -> f VARCHAR(10000), g VARCHAR(10000)) ENGINE=InnoDB;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

See [Section E.10.4, “Table Column-Count and Row-Size Limits”](#).

- On some older operating systems, files must be less than 2GB. This is not a limitation of InnoDB itself, but if you require a large tablespace, you will need to configure it using several smaller data files rather than one or a file large data files.
- The combined size of the InnoDB log files must be less than 4GB.
- The minimum tablespace size is 10MB. The maximum tablespace size is four billion database pages (64TB). This is also the maximum size for a table.
- The default database page size in InnoDB is 16KB.



Note

Changing the page size is not a supported operation and there is no guarantee that InnoDB will function normally with a page size other than 16KB. Problems compiling or running InnoDB may occur. In particular, `ROW_FORMAT=COMPRESSED` in the Barracuda file format assumes that the page size is at most 16KB and uses 14-bit pointers.

A version of InnoDB built for one page size cannot use data files or log files from a version built for a different page size. This limitation could affect restore or downgrade operations using data from MySQL 5.6, which does support page sizes other than 16KB.

Index Types

- InnoDB tables do not support `FULLTEXT` indexes.
- InnoDB tables support spatial data types, but not indexes on them.

Restrictions on InnoDB Tables

- `ANALYZE TABLE` determines index cardinality (as displayed in the `Cardinality` column of `SHOW INDEX` output) by doing eight random dives to each of the index trees and updating index cardinality estimates accordingly. Because these are only estimates, repeated runs of `ANALYZE TABLE` may produce different numbers. This makes `ANALYZE TABLE` fast on InnoDB tables but not 100% accurate because it does not take all rows into account.

You can change the number of random dives by modifying the `innodb_stats_sample_pages` [1571] system variable. For more information, see [Section 14.4.8, “Changes for Flexibility, Ease of Use and Reliability”](#).

MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you can try using `ANALYZE TABLE`. In the few cases that `ANALYZE TABLE` does not produce values good enough for your particular tables, you can use `FORCE INDEX` with your queries to force the use of a particular index, or set the `max_seeks_for_key` [524] system variable to ensure that MySQL prefers index lookups over table scans. See [Section 5.1.4, “Server System Variables”](#), and [Section C.5.6, “Optimizer-Related Issues”](#).

- `SHOW TABLE STATUS` does not give accurate [statistics](#) on InnoDB tables, except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.
- InnoDB does not keep an internal count of rows in a table because concurrent transactions might “see” different numbers of rows at the same time. To process a `SELECT COUNT(*) FROM t` statement, InnoDB scans an index of the table, which takes some time if the index is not entirely in the buffer pool. If your table does not change often, using the MySQL query cache is a good solution. To get a fast count, you have to use a counter table you create yourself and let your application update it according to the inserts and deletes it does. If an approximate row count is sufficient, `SHOW TABLE STATUS` can be used. See [Section 14.3.14.1, “InnoDB Performance Tuning Tips”](#).
- On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names.
- An `AUTO_INCREMENT` column `ai_col` must be defined as part of an index such that it is possible to perform the equivalent of an indexed `SELECT MAX(ai_col)` lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.
- While initializing a previously specified `AUTO_INCREMENT` column on a table, InnoDB sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column. While accessing the auto-increment counter, InnoDB uses a specific `AUTO-INC` table lock mode where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other clients cannot insert into the table while the `AUTO-INC` table lock is held. See [Section 14.3.5.3, “AUTO_INCREMENT Handling in InnoDB”](#).
- When you restart the MySQL server, InnoDB may reuse an old value that was generated for an `AUTO_INCREMENT` column but never stored (that is, a value that was generated during an old transaction that was rolled back).
- When an `AUTO_INCREMENT` integer column runs out of values, a subsequent `INSERT` operation returns a duplicate-key error. This is general MySQL behavior, similar to how MyISAM works.
- `DELETE FROM tbl_name` does not regenerate the table but instead deletes all rows, one by one.
- Under some conditions, `TRUNCATE tbl_name` for an InnoDB table is mapped to `DELETE FROM tbl_name`. See [Section 13.1.33, “TRUNCATE TABLE Syntax”](#).
- Currently, cascaded foreign key actions do not activate triggers.
- You cannot create a table with a column name that matches the name of an internal InnoDB column (including `DB_ROW_ID`, `DB_TRX_ID`, `DB_ROLL_PTR`, and `DB_MIX_ID`). The server reports error 1005 and refers to error –1 in the error message. This restriction applies only to use of the names in uppercase.

Locking and Transactions

- `LOCK TABLES` acquires two locks on each table if `innodb_table_locks=1` (the default). In addition to a table lock on the MySQL layer, it also acquires an InnoDB table lock. Versions of MySQL before 4.1.2 did not acquire InnoDB table locks; the old behavior can be selected by setting `innodb_table_locks=0`. If no InnoDB table lock is acquired, `LOCK TABLES` completes even if some records of the tables are being locked by other transactions.

As of MySQL 5.5.3, `innodb_table_locks=0` [1573] has no effect for tables locked explicitly with `LOCK TABLES ... WRITE`. It still has an effect for tables locked for read or write by `LOCK TABLES ... WRITE` implicitly (for example, through triggers) or by `LOCK TABLES ... READ`.

- All InnoDB locks held by a transaction are released when the transaction is committed or aborted. Thus, it does not make much sense to invoke `LOCK TABLES` on InnoDB tables in `autocommit=1` [484] mode because the acquired InnoDB table locks would be released immediately.
- You cannot lock additional tables in the middle of a transaction because `LOCK TABLES` performs an implicit `COMMIT` and `UNLOCK TABLES`.
- The limit of 1023 concurrent data-modifying transactions has been raised in MySQL 5.5 and above. The limit is now 128 * 1023 concurrent transactions that generate undo records. You can remove any workarounds that require changing the proper structure of your transactions, such as committing more frequently.

14.4. New Features of InnoDB 1.1

14.4.1. Introduction to InnoDB 1.1

InnoDB 1.1 combines the familiar reliability and performance of the InnoDB storage engine, with new performance and usability enhancements. InnoDB 1.1 includes all the features that were part of the InnoDB Plugin for MySQL 5.1, plus new features specific to MySQL 5.5 and higher.

Beginning with MySQL version 5.5, InnoDB is the default storage engine, rather than MyISAM, to promote greater data reliability and reducing the chance of corruption.

14.4.1.1. Features of the InnoDB Storage Engine

InnoDB in MySQL 5.5 contains several important new features:

- [Fast index creation: add or drop indexes without copying the data](#)
- [Data compression: shrink tables, to significantly reduce storage and I/O](#)
- [New row format: fully off-page storage of long BLOB, TEXT, and VARCHAR columns](#)
- [File format management: protects upward and downward compatibility](#)
- [INFORMATION_SCHEMA tables: information about compression and locking](#)
- [Performance and scalability enhancements:](#)
 - [Section 14.4.7.2, “Faster Locking for Improved Scalability”](#)
 - [Section 14.4.7.3, “Using Operating System Memory Allocators”](#)
 - [Section 14.4.7.4, “Controlling InnoDB Change Buffering”](#)
 - [Section 14.4.7.5, “Controlling Adaptive Hash Indexing”](#)

- [Section 14.4.7.6, “Changes Regarding Thread Concurrency”](#)
- [Section 14.4.7.7, “Changes in the Read-Ahead Algorithm”](#)
- [Section 14.4.7.8, “Multiple Background I/O Threads”](#)
- [Section 14.4.7.9, “Asynchronous I/O on Linux”](#)
- [Section 14.4.7.10, “Group Commit”](#)
- [Section 14.4.7.11, “Controlling the Master Thread I/O Rate”](#)
- [Section 14.4.7.12, “Controlling the Flushing Rate of Dirty Pages”](#)
- [Section 14.4.7.13, “Using the PAUSE Instruction in InnoDB Spin Loops”](#)
- [Section 14.4.7.14, “Control of Spin Lock Polling”](#)
- [Section 14.4.7.15, “Making Buffer Pool Scan Resistant”](#)
- [Section 14.4.7.16, “Improvements to Crash Recovery Performance”](#)
- [Section 14.4.7.17, “Integration with MySQL PERFORMANCE_SCHEMA”](#)
- [Section 14.4.7.18, “Improvements to Performance from Multiple Buffer Pools”](#)
- [Section 14.4.7.19, “Better Scalability with Multiple Rollback Segments”](#)
- [Section 14.4.7.20, “Better Scalability with Improved Purge Scheduling”](#)
- [Section 14.4.7.21, “Improved Log Sys Mutex”](#)
- [Section 14.4.7.22, “Separate Flush List Mutex”](#)
- [Section 14.4.14.3, “Parameters with New Defaults”](#)
- Other changes for flexibility, ease of use and reliability:
 - [Section 14.4.8.1, “The Barracuda File Format”](#)
 - [Section 14.4.8.2, “Dynamic Control of System Configuration Parameters”](#)
 - [Section 14.4.8.3, “TRUNCATE TABLE Reclaims Space”](#)
 - [Section 14.4.8.4, “InnoDB Strict Mode”](#)
 - [Section 14.4.8.5, “Controlling Optimizer Statistics Estimation”](#)
 - [Section 14.4.8.6, “Better Error Handling when Dropping Indexes”](#)
 - [Section 14.4.8.7, “More Compact Output of SHOW ENGINE INNODB MUTEX”](#)
 - [Section 14.4.8.8, “More Read-Ahead Statistics”](#)

Upward and Downward Compatibility

Note that the ability to use data compression and the new row format require the use of a new InnoDB file format called Barracuda. The previous file format, used by the built-in InnoDB in MySQL 5.1 and earlier, is

now called Antelope and does not support these features, but does support the other features introduced with the InnoDB storage engine.

The InnoDB storage engine is upward compatible from standard InnoDB as built in to, and distributed with, MySQL. Existing databases can be used with the InnoDB Storage Engine for MySQL. The new parameter `innodb_file_format` [1554] can help protect upward and downward compatibility between InnoDB versions and database files, allowing users to enable or disable use of new features that can only be used with certain versions of InnoDB.

InnoDB since version 5.0.21 has a safety feature that prevents it from opening tables that are in an unknown format. However, the system tablespace may contain references to new-format tables that confuse the built-in InnoDB in MySQL 5.1 and earlier. These references are cleared in a [slow shutdown](#).

With previous versions of InnoDB, no error would be returned until you try to access a table that is in a format “too new” for the software. To provide early feedback, InnoDB 1.1 checks the system tablespace before startup to ensure that the file format used in the database is supported by the storage engine. See [Section 14.4.4.2.1, “Compatibility Check When InnoDB Is Started”](#) for the details.

14.4.1.2. Obtaining and Installing the InnoDB Storage Engine

Starting with MySQL 5.4.2, you do not need to do anything special to get or install the most up-to-date InnoDB storage engine. From that version forward, the InnoDB storage engine in the server is what was formerly known as the InnoDB Plugin. Earlier versions of MySQL required some extra build and configuration steps to get the Plugin-specific features such as fast index creation and table compression.

Report any bugs in the InnoDB storage engine using the [My Oracle Support site](#). For general discussions about InnoDB Storage Engine for MySQL, see <http://forums.mysql.com/list.php?22>.

14.4.1.3. Viewing the InnoDB Storage Engine Version Number

InnoDB storage engine releases are numbered with version numbers independent of MySQL release numbers. The initial release of the InnoDB storage engine is version 1.0, and it is designed to work with MySQL 5.1. Version 1.1 of the InnoDB storage engine is for MySQL 5.5 and up.

- The first component of the InnoDB storage engine version number designates a major release level.
- The second component corresponds to the MySQL release. The digit 0 corresponds to MySQL 5.1. The digit 1 corresponds to MySQL 5.5.
- The third component indicates the specific release of the InnoDB storage engine (at a given major release level and for a specific MySQL release); only bug fixes and minor functional changes are introduced at this level.

Once you have installed the InnoDB storage engine, you can check its version number in three ways:

- In the error log, it is printed during startup.
- `SELECT * FROM information_schema.plugins;`
- `SELECT @@innodb_version;`

The InnoDB storage engine writes its version number to the error log, which can be helpful in diagnosis of errors:

```
091105 12:28:06 InnoDB Plugin 1.0.5 started; log sequence number 46509
```

Note that the `PLUGIN_VERSION` column in the table `INFORMATION_SCHEMA.PLUGINS` does not display the third component of the version number, only the first and second components, as in 1.0.

14.4.1.4. Compatibility Considerations for Downgrade and Backup

Because InnoDB 1.1 supports the “Barracuda” file format, with new on-disk data structures within both the database and log files, pay special attention to file format compatibility with respect to the following scenarios:

- Downgrading from MySQL 5.5 to the MySQL 5.1 or earlier (without the InnoDB Plugin enabled), or otherwise using earlier versions of MySQL with database files created by MySQL 5.5 and higher.
- Using `mysqldump`.
- Using MySQL replication.
- Using MySQL Enterprise Backup or InnoDB Hot Backup.

WARNING: Once you create any tables with the Barracuda file format, take care to avoid crashes and corruptions when using those files with an earlier version of MySQL. It is **strongly** recommended that you use a “slow shutdown” (`SET GLOBAL innodb_fast_shutdown=0`) when stopping the MySQL server before downgrading to MySQL 5.1 or earlier. This ensures that the log files and other system information do not cause consistency issues or startup problems when using a prior version of MySQL.

WARNING: If you dump a database containing compressed tables with `mysqldump`, the dump file may contain `CREATE TABLE` statements that attempt to create compressed tables, or those using `ROW_FORMAT=DYNAMIC` in the new database. Therefore, be sure the new database is running the InnoDB storage engine, with the proper settings for `innodb_file_format` [1554] and `innodb_file_per_table` [1555], if you want to have the tables re-created as they exist in the original database. Typically, when the `mysqldump` file is loaded, MySQL and InnoDB ignore `CREATE TABLE` options they do not recognize, and the table(s) are created in a format used by the running server.

WARNING: If you use MySQL replication, ensure all slaves are configured with the InnoDB storage engine, with the same settings for `innodb_file_format` [1554] and `innodb_file_per_table` [1555]. If you do not do so, and you create tables that require the new Barracuda file format, replication errors may occur. If a slave MySQL server is running an older version of MySQL, it ignores the `CREATE TABLE` options to create a compressed table or one with `ROW_FORMAT=DYNAMIC`, and creates the table uncompressed, with `ROW_FORMAT=COMPACT`.

WARNING: Version 3.0 of InnoDB Hot Backup does not support the new Barracuda file format. Using InnoDB Hot Backup Version 3 to backup databases in this format causes unpredictable behavior. MySQL Enterprise Backup, the successor product to InnoDB Hot Backup, does support tables with the Barracuda file format. You can also back up such databases with `mysqldump`.

14.4.2. Fast Index Creation in the InnoDB Storage Engine

In MySQL 5.5 and higher, or in MySQL 5.1 with the InnoDB Plugin, creating and dropping [secondary indexes](#) does not copy the contents of the entire table, making this operation much more efficient than with prior releases.

14.4.2.1. Overview of Fast Index Creation

With MySQL 5.5 and higher, or MySQL 5.1 with the InnoDB Plugin, creating and dropping [secondary indexes](#) for InnoDB tables is much faster than before. Historically, adding or dropping an index on a table with existing data could be very slow. The `CREATE INDEX` and `DROP INDEX` statements worked by creating a new, empty table defined with the requested set of indexes, then copying the existing rows to the new table one-by-one, updating the indexes as the rows are inserted. After all rows from the original table were copied, the old table was dropped and the copy was renamed with the name of the original table.

The performance speedup for fast index creation applies to secondary indexes, not to the primary key index. The rows of an InnoDB table are stored in a [clustered index](#) organized based on the [primary key](#), forming what some database systems call an “index-organized table”. Because the table structure is so closely tied to the primary key, redefining the primary key still requires copying the data.

This new mechanism also means that you can generally speed the overall process of creating and loading an indexed table by creating the table with only the clustered index, and adding the secondary indexes after the data is loaded.

Although no syntax changes are required in the [CREATE INDEX](#) or [DROP INDEX](#) commands, some factors affect the performance, space usage, and semantics of this operation (see [Section 14.4.2.6, “Limitations of Fast Index Creation”](#)).

14.4.2.2. Examples of Fast Index Creation

It is possible to create multiple indexes on a table with one [ALTER TABLE](#) statement. This is relatively efficient, because the clustered index of the table needs to be scanned only once (although the data is sorted separately for each new index). For example:

```
CREATE TABLE T1(A INT PRIMARY KEY, B INT, C CHAR(1)) ENGINE=InnoDB;
INSERT INTO T1 VALUES (1,2,'a'), (2,3,'b'), (3,2,'c'), (4,3,'d'), (5,2,'e');
COMMIT;
ALTER TABLE T1 ADD INDEX (B), ADD UNIQUE INDEX (C);
```

The above statements create table [T1](#) with the clustered index (primary key) on column [A](#), insert several rows, and then build two new indexes on columns [B](#) and [C](#). If there were many rows inserted into [T1](#) before the [ALTER TABLE](#) statement, this approach is much more efficient than creating all the secondary indexes before loading the data.

You can also create the indexes one at a time, but then the clustered index of the table is scanned (as well as sorted) once for each [CREATE INDEX](#) statement. Thus, the following statements are not as efficient as the [ALTER TABLE](#) statement above, even though neither requires recreating the clustered index for table [T1](#).

```
CREATE INDEX B ON T1 (B);
CREATE UNIQUE INDEX C ON T1 (C);
```

Dropping InnoDB secondary indexes also does not require any copying of table data. You can equally quickly drop multiple indexes with a single [ALTER TABLE](#) statement or multiple [DROP INDEX](#) statements:

```
ALTER TABLE T1 DROP INDEX B, DROP INDEX C;
```

or:

```
DROP INDEX B ON T1;
DROP INDEX C ON T1;
```

Restructuring the clustered index in InnoDB always requires copying the data in the table. For example, if you create a table without a primary key, InnoDB chooses one for you, which may be the first [UNIQUE](#) key defined on [NOT NULL](#) columns, or a system-generated key. Defining a [PRIMARY KEY](#) later causes the data to be copied, as in the following example:

```
CREATE TABLE T2 (A INT, B INT) ENGINE=InnoDB;
INSERT INTO T2 VALUES (NULL, 1);
ALTER TABLE T2 ADD PRIMARY KEY (B);
```

When you create a [UNIQUE](#) or [PRIMARY KEY](#) index, InnoDB must do some extra work. For [UNIQUE](#) indexes, InnoDB checks that the table contains no duplicate values for the key. For a [PRIMARY KEY](#) index,

InnoDB also checks that none of the `PRIMARY KEY` columns contains a `NULL`. It is best to define the primary key when you create a table, so you need not rebuild the table later.

14.4.2.3. Implementation Details of Fast Index Creation

InnoDB has two types of indexes: the clustered index and secondary indexes. Since the clustered index contains the data values in its B-tree nodes, adding or dropping a clustered index does involve copying the data, and creating a new copy of the table. A secondary index, however, contains only the index key and the value of the primary key. This type of index can be created or dropped without copying the data in the clustered index. Because each secondary index contains copies of the primary key values (used to access the clustered index when needed), when you change the definition of the primary key, all secondary indexes are recreated as well.

Dropping a secondary index is simple. Only the internal InnoDB system tables and the MySQL data dictionary tables are updated to reflect the fact that the index no longer exists. InnoDB returns the storage used for the index to the tablespace that contained it, so that new indexes or additional table rows can use the space.

To add a secondary index to an existing table, InnoDB scans the table, and sorts the rows using memory buffers and temporary files in order by the values of the secondary index key columns. The B-tree is then built in key-value order, which is more efficient than inserting rows into an index in random order. Because the B-tree nodes are split when they fill, building the index in this way results in a higher fill-factor for the index, making it more efficient for subsequent access.

14.4.2.4. Concurrency Considerations for Fast Index Creation

While an InnoDB secondary index is being created or dropped, the table is locked in shared mode. Any writes to the table are blocked, but the data in the table can be read. When you alter the clustered index of a table, the table is locked in exclusive mode, because the data must be copied. Thus, during the creation of a new clustered index, all operations on the table are blocked.

A `CREATE INDEX` or `ALTER TABLE` statement for an InnoDB table always waits for currently executing transactions that are accessing the table to commit or roll back. `ALTER TABLE` statements that redefine an InnoDB primary key wait for all `SELECT` statements that access the table to complete, or their containing transactions to commit. No transactions whose execution spans the creation of the index can be accessing the table, because the original table is dropped when the clustered index is restructured.

Once a `CREATE INDEX` or `ALTER TABLE` statement that creates an InnoDB secondary index begins executing, queries can access the table for read access, but cannot update the table. If an `ALTER TABLE` statement is changing the clustered index for an InnoDB table, all queries wait until the operation completes.

A newly-created InnoDB secondary index contains only the committed data in the table at the time the `CREATE INDEX` or `ALTER TABLE` statement begins to execute. It does not contain any uncommitted values, old versions of values, or values marked for deletion but not yet removed from the old index.

Because a newly-created index contains only information about data current at the time the index was created, queries that need to see data that was deleted or changed before the index was created cannot use the index. The only queries that could be affected by this limitation are those executing in transactions that began before the creation of the index was begun. For such queries, unpredictable results could occur. Newer queries can use the index.

14.4.2.5. How Crash Recovery Works with Fast Index Creation

Although no data is lost if the server crashes while an `ALTER TABLE` statement is executing, the `crash recovery` process is different for `clustered indexes` and `secondary indexes`.

If the server crashes while creating an InnoDB secondary index, upon recovery, MySQL drops any partially created indexes. You must re-run the `ALTER TABLE` or `CREATE INDEX` statement.

When a crash occurs during the creation of an InnoDB clustered index, recovery is more complicated, because the data in the table must be copied to an entirely new clustered index. Remember that all InnoDB tables are stored as clustered indexes. In the following discussion, we use the word table and clustered index interchangeably.

MySQL creates the new clustered index by copying the existing data from the original InnoDB table to a temporary table that has the desired index structure. Once the data is completely copied to this temporary table, the original table is renamed with a different temporary table name. The temporary table comprising the new clustered index is renamed with the name of the original table, and the original table is dropped from the database.

If a system crash occurs while creating a new clustered index, no data is lost, but you must complete the recovery process using the temporary tables that exist during the process. Since it is rare to re-create a clustered index or re-define primary keys on large tables, or to encounter a system crash during this operation, this manual does not provide information on recovering from this scenario. Instead, please see the InnoDB web site: <http://www.innodb.com/support/tips>.

14.4.2.6. Limitations of Fast Index Creation

Take the following considerations into account when creating or dropping InnoDB indexes:

- During index creation, files are written to the temporary directory (`$TMPDIR` on Unix, `%TEMP%` on Windows, or the value of the `--tmpdir` [568] configuration variable). Each temporary file is large enough to hold one column that makes up the new index, and each one is removed as soon as it is merged into the final index.
- The table is copied, rather than using Fast Index Creation when you create an index on a `TEMPORARY TABLE`. This has been reported as MySQL Bug #39833.
- To avoid consistency issues between the InnoDB data dictionary and the MySQL data dictionary, the table is copied, rather than using Fast Index Creation when you use the `ALTER TABLE ... RENAME COLUMN` syntax.
- The statement `ALTER IGNORE TABLE t ADD UNIQUE INDEX` does not delete duplicate rows. This has been reported as MySQL Bug #40344. The `IGNORE` keyword is ignored. If any duplicate rows exist, the operation fails with the following error message:

```
ERROR 23000: Duplicate entry '347' for key 'pl'
```

- As noted above, a newly-created index contains only information about data current at the time the index was created. Therefore, you should not run queries in a transaction that might use a secondary index that did not exist at the beginning of the transaction. There is no way for InnoDB to access “old” data that is consistent with the rest of the data read by the transaction. See the discussion of locking in [Section 14.4.2.4, “Concurrency Considerations for Fast Index Creation”](#).

Prior to InnoDB storage engine 1.0.4, unexpected results could occur if a query attempts to use an index created after the start of the transaction containing the query. If an old transaction attempts to access a “too new” index, InnoDB storage engine 1.0.4 and later reports an error:

```
ERROR HY000: Table definition has changed, please retry transaction
```

As the error message suggests, committing (or rolling back) the transaction, and restarting it, cures the problem.

- InnoDB storage engine 1.0.2 introduces some improvements in error handling when users attempt to drop indexes. See section [Section 14.4.8.6, “Better Error Handling when Dropping Indexes”](#) for details.
- MySQL 5.5 does not support efficient creation or dropping of `FOREIGN KEY` constraints. Therefore, if you use `ALTER TABLE` to add or remove a `REFERENCES` constraint, the child table is copied, rather than using Fast Index Creation.

14.4.3. InnoDB Data Compression

By setting InnoDB configuration options, you can create tables where the data is stored in compressed form. The compression means less data is transferred between disk and memory, and takes up less space in memory. The benefits are amplified for tables with secondary indexes, because index data is compressed also. Compression can be especially important for [SSD](#) storage devices, because they tend to have lower capacity than [HDD](#) devices.

14.4.3.1. Overview of Table Compression

Because processors and cache memories have increased in speed more than disk storage devices, many workloads are I/O-bound. Data [compression](#) enables smaller database size, reduced I/O, and improved throughput, at the small cost of increased CPU utilization. Compression is especially valuable for read-intensive applications, on systems with enough RAM to keep frequently-used data in memory.

An InnoDB table created with `ROW_FORMAT=COMPRESSED` can use a smaller page size on disk than the usual 16KB default. Smaller pages require less I/O to read from and write to disk, which is especially valuable for [SSD](#) devices.

The page size is specified through the `KEY_BLOCK_SIZE` parameter. The different page size means the table must be in its own `.ibd` file rather than the system tablespace, which requires enabling the `innodb_file_per_table` option. The level of compression is the same regardless of the `KEY_BLOCK_SIZE` value. As you specify smaller values for `KEY_BLOCK_SIZE`, you get the I/O benefits of increasingly smaller pages. But if you specify a value that is too small, there is additional overhead to reorganize the pages when data values cannot be compressed enough to fit multiple rows in each page. There is a hard limit on how small `KEY_BLOCK_SIZE` can be for a table, based on the lengths of the key columns for each of its indexes. Specify a value that is too small, and the `CREATE TABLE` or `ALTER TABLE` statement fails.

In the buffer pool, the compressed data is held in small pages, with a page size based on the `KEY_BLOCK_SIZE` value. For extracting or updating the column values, InnoDB also creates a 16KB page in the buffer pool with the uncompressed data. Within the buffer pool, any updates to the uncompressed page are also re-written back to the equivalent compressed page. You might need to size your buffer pool to accommodate the additional data of both compressed and uncompressed pages, although the uncompressed pages are [evicted](#) from the buffer pool when space is needed, and then uncompressed again on the next access.

14.4.3.2. Enabling Compression for a Table

The default uncompressed size of InnoDB data pages is 16KB. You can use the attributes `ROW_FORMAT=COMPRESSED`, `KEY_BLOCK_SIZE`, or both in the `CREATE TABLE` and `ALTER TABLE` statements to enable table compression. Depending on the combination of option values, InnoDB uses a page size of 1KB, 2KB, 4KB, 8KB, or 16KB for the `.ibd` file of the table. (The actual compression algorithm is not affected by the `KEY_BLOCK_SIZE` value.)



Note

Compression is applicable to tables, not to individual rows, despite the option name `ROW_FORMAT`.

To create a compressed table, you might use a statement like this:

```
CREATE TABLE name
  (column1 INT PRIMARY KEY)
  ENGINE=InnoDB
  ROW_FORMAT=COMPRESSED
  KEY_BLOCK_SIZE=4;
```

If you specify `ROW_FORMAT=COMPRESSED` but not `KEY_BLOCK_SIZE`, the default compressed page size of 8KB is used. If `KEY_BLOCK_SIZE` is specified, you can omit the attribute `ROW_FORMAT=COMPRESSED`.

Setting `KEY_BLOCK_SIZE=16` typically does not result in much compression, since the normal InnoDB page size is 16KB. This setting may still be useful for tables with many long `BLOB`, `VARCHAR` or `TEXT` columns, because such values often do compress well, and might therefore require fewer “overflow” pages as described in [Section 14.4.3.4, “Compressing BLOB, VARCHAR and TEXT Columns”](#).

All indexes of a table (including the clustered index) are compressed using the same page size, as specified in the `CREATE TABLE` or `ALTER TABLE` statement. Table attributes such as `ROW_FORMAT` and `KEY_BLOCK_SIZE` are not part of the `CREATE INDEX` syntax, and are ignored if they are specified (although you see them in the output of the `SHOW CREATE TABLE` statement).

14.4.3.2.1. Configuration Parameters for Compression

Compressed tables are stored in a format that previous versions of InnoDB cannot process. To preserve downward compatibility of database files, compression can be specified only when the [Barracuda](#) data file format is enabled using the configuration parameter `innodb_file_format` [\[1554\]](#).

Table compression is also not available for the InnoDB system tablespace. The system tablespace (space 0, the `ibdata*` files) may contain user data, but it also contains internal InnoDB system information, and therefore is never compressed. Thus, compression applies only to tables (and indexes) stored in their own tablespaces.

To use compression, enable the [file-per-table](#) mode using the configuration parameter `innodb_file_per_table` [\[1555\]](#) and enable the Barracuda disk file format using the parameter `innodb_file_format` [\[1554\]](#). If necessary, you can set these parameters in the MySQL option file `my.cnf` or `my.ini`, or with the `SET` statement without shutting down the MySQL server.

Specifying `ROW_FORMAT=COMPRESSED` or `KEY_BLOCK_SIZE` in `CREATE TABLE` or `ALTER TABLE` statements produces these warnings if the Barracuda file format is not enabled. You can view them with the `SHOW WARNINGS` statement.

Level	Code	Message
Warning	1478	InnoDB: KEY_BLOCK_SIZE requires innodb_file_per_table.
Warning	1478	InnoDB: KEY_BLOCK_SIZE requires innodb_file_format=1
Warning	1478	InnoDB: ignoring KEY_BLOCK_SIZE=4.
Warning	1478	InnoDB: ROW_FORMAT=COMPRESSED requires innodb_file_per_table.
Warning	1478	InnoDB: assuming ROW_FORMAT=COMPACT.



Note

These messages are only warnings, not errors, and the table is created as if the options were not specified. When InnoDB “strict mode” (see [Section 14.4.8.4, “InnoDB Strict Mode”](#)) is enabled, InnoDB generates an error, not a warning, for these cases. In strict mode, the table is not created if the current configuration does not permit using compressed tables.

The “non-strict” behavior is intended to permit you to import a `mysqldump` file into a database that does not support compressed tables, even if the source database contained compressed tables. In that case, MySQL creates the table in `ROW_FORMAT=COMPACT` instead of preventing the operation.

When you import the dump file into a new database, if you want to have the tables re-created as they exist in the original database, ensure the server is running the InnoDB storage engine with the proper settings for the configuration parameters `innodb_file_format` [1554] and `innodb_file_per_table` [1555],

14.4.3.2.2. SQL Compression Syntax Warnings and Errors

The attribute `KEY_BLOCK_SIZE` is permitted only when `ROW_FORMAT` is specified as `COMPRESSED` or is omitted. Specifying a `KEY_BLOCK_SIZE` with any other `ROW_FORMAT` generates a warning that you can view with `SHOW WARNINGS`. However, the table is non-compressed; the specified `KEY_BLOCK_SIZE` is ignored).

Level	Code	Message
Warning	1478	InnoDB: ignoring KEY_BLOCK_SIZE=n unless ROW_FORMAT=COMPRESSED.

If you are running in InnoDB strict mode, the combination of a `KEY_BLOCK_SIZE` with any `ROW_FORMAT` other than `COMPRESSED` generates an error, not a warning, and the table is not created.

Table 14.4, “Meaning of `CREATE TABLE` and `ALTER TABLE` options” summarizes how the various options on `CREATE TABLE` and `ALTER TABLE` are handled.

Table 14.4. Meaning of `CREATE TABLE` and `ALTER TABLE` options

Option	Usage	Description
<code>ROW_FORMAT=REDUNDANT</code>	Storage format used prior to MySQL 5.0.3	Less efficient than <code>ROW_FORMAT=COMPACT</code> ; for backward compatibility
<code>ROW_FORMAT=COMPACT</code>	Default storage format since MySQL 5.0.3	Stores a prefix of 768 bytes of long column values in the clustered index page, with the remaining bytes stored in an overflow page
<code>ROW_FORMAT=DYNAMIC</code>	Available only with <code>innodb_file_format=Barracuda</code>	Store values within the clustered index page if they fit; if not, stores only a 20-byte pointer to an overflow page (no prefix)
<code>ROW_FORMAT=COMPRESSED</code>	Available only with <code>innodb_file_format=Barracuda</code>	Compresses the table and indexes using zlib to default compressed page size of 8K bytes; implies <code>ROW_FORMAT=DYNAMIC</code>
<code>KEY_BLOCK_SIZE=n</code>	Available only with <code>innodb_file_format=Barracuda</code>	Specifies compressed page size of 1, 2, 4, 8 or 16K bytes; implies <code>ROW_FORMAT=DYNAMIC</code> and <code>ROW_FORMAT=COMPRESSED</code>

Table 14.5, “`CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF`” summarizes error conditions that occur with certain combinations of configuration parameters and options on the `CREATE TABLE` or `ALTER TABLE` statements, and how the options appear in the output of `SHOW TABLE STATUS`.

When InnoDB strict mode is `OFF`, InnoDB creates or alters the table, but may ignore certain settings, as shown below. You can see the warning messages in the MySQL error log. When InnoDB strict mode is `ON`, these specified combinations of options generate errors, and the table is not created or altered. You can see the full description of the error condition with `SHOW ERRORS`. For example:

```
mysql> CREATE TABLE x (id INT PRIMARY KEY, c INT)
```

```
-> ENGINE=INNODB KEY_BLOCK_SIZE=33333;

ERROR 1005 (HY000): Can't create table 'test.x' (errno: 1478)

mysql> SHOW ERRORS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Error | 1478 | InnoDB: invalid KEY_BLOCK_SIZE=33333.     |
| Error | 1005 | Can't create table 'test.x' (errno: 1478) |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Table 14.5. CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF

Syntax	Warning or Error Condition	Resulting ROW_FORMAT, as shown in SHOW TABLE STATUS
ROW_FORMAT=REDUNDANT	None	REDUNDANT
ROW_FORMAT=COMPACT	None	COMPACT
ROW_FORMAT=COMPRESSED or ROW_FORMAT=DYNAMIC or KEY_BLOCK_SIZE is specified	Ignored unless both innodb_file_format [1554]=Barracuda and innodb_file_per_table [1555] are enabled	COMPACT
Invalid KEY_BLOCK_SIZE is specified (not 1, 2, 4, 8 or 16)	KEY_BLOCK_SIZE is ignored	the requested one, or COMPACT by default
ROW_FORMAT=COMPRESSED and valid KEY_BLOCK_SIZE are specified	None; KEY_BLOCK_SIZE specified is used, not the 8K default	COMPRESSED
KEY_BLOCK_SIZE is specified with REDUNDANT, COMPACT or DYNAMIC row format	KEY_BLOCK_SIZE is ignored	REDUNDANT, COMPACT or DYNAMIC
ROW_FORMAT is not one of REDUNDANT, COMPACT, DYNAMIC or COMPRESSED	Ignored if recognized by the MySQL parser. Otherwise, an error is issued.	COMPACT or N/A

When InnoDB strict mode is ON (`innodb_strict_mode=1`), the InnoDB storage engine rejects invalid `ROW_FORMAT` or `KEY_BLOCK_SIZE` parameters. For compatibility with earlier versions of InnoDB, strict mode is not enabled by default; instead, InnoDB issues warnings (not errors) for ignored invalid parameters.

Note that it is not possible to see the chosen `KEY_BLOCK_SIZE` using `SHOW TABLE STATUS`. The statement `SHOW CREATE TABLE` displays the `KEY_BLOCK_SIZE` (even if it was ignored by InnoDB). The real compressed page size inside InnoDB cannot be displayed by MySQL.

14.4.3.3. Tuning InnoDB Compression

Most often, the internal optimizations in InnoDB described in [Section 14.4.3.4, “InnoDB Data Storage and Compression”](#) ensure that the system runs well with compressed data. However, because the efficiency of compression depends on the nature of your data, there are some factors you should consider to get best performance. You need to choose which tables to compress, and what compressed page size to use. You

might also adjust the size of the buffer pool based on run-time performance characteristics, such as the amount of time the system spends compressing and uncompressing data.

When to Use Compression

In general, compression works best on tables that include a reasonable number of character string columns and where the data is read far more often than it is written. Because there are no guaranteed ways to predict whether or not compression benefits a particular situation, always test with a specific [workload](#) and data set running on a representative configuration. Consider the following factors when deciding which tables to compress.

Data Characteristics and Compression

A key determinant of the efficiency of compression in reducing the size of data files is the nature of the data itself. Recall that compression works by identifying repeated strings of bytes in a block of data. Completely randomized data is the worst case. Typical data often has repeated values, and so compresses effectively. Character strings often compress well, whether defined in [CHAR](#), [VARCHAR](#), [TEXT](#) or [BLOB](#) columns. On the other hand, tables containing mostly binary data (integers or floating point numbers) or data that is previously compressed (for example JPEG or PNG images) may not generally compress well, significantly or at all.

You choose whether to turn on compression for each InnoDB tables. A table and all of its indexes use the same (compressed) page size. It might be that the primary key (clustered) index, which contains the data for all columns of a table, compresses more effectively than the secondary indexes. For those cases where there are long rows, the use of compression might result in long column values being stored “off-page”, as discussed in [Section 14.4.5.3, “Barracuda File Format: DYNAMIC and COMPRESSED Row Formats”](#). Those overflow pages may compress well. Given these considerations, for many applications, some tables compress more effectively than others, and you might find that your workload performs best only with a subset of tables compressed.

Experimenting is the only way to determine whether or not to compress a particular table. InnoDB compresses data in 16K chunks corresponding to the uncompressed page size, and in addition to user data, the page format includes some internal system data that is not compressed. Compression utilities compress an entire stream of data, and so may find more repeated strings across the entire input stream than InnoDB would find in a table compressed in 16K chunks. But you can get a sense of how compression efficiency by using a utility that implements LZ77 compression (such as [gzip](#) or WinZip) on your data file.

Another way to test compression on a specific table is to copy some data from your uncompressed table to a similar, compressed table (having all the same indexes) and look at the size of the resulting file. When you do so (if nothing else using compression is running), you can examine the ratio of successful compression operations to overall compression operations. (In the [INNODB_CMP](#) table, compare [COMPRESS_OPS](#) to [COMPRESS_OPS_OK](#). See [INNODB_CMP](#) for more information.) If a high percentage of compression operations complete successfully, the table might be a good candidate for compression.

Compression and Application and Schema Design

Decide whether to compress data in your application or in the InnoDB table. It is usually not sensible to store data that is compressed by an application in an InnoDB compressed table. Further compression is extremely unlikely, and the attempt to compress just wastes CPU cycles.

Compressing in the Database

The InnoDB table compression is automatic and applies to all columns and index values. The columns can still be tested with operators such as [LIKE](#), and sort operations can still use indexes even when the index values are compressed. Because indexes are often a significant fraction of the total size of a database,

compression could result in significant savings in storage, I/O or processor time. The compression and decompression operations happen on the database server, which likely is a powerful system that is sized to handle the expected load.

Compressing in the Application

If you compress data such as text in your application, before it is inserted into the database, You might save overhead for data that does not compress well by compressing some columns and not others. This approach uses CPU cycles for compression and uncompression on the client machine rather than the database server, which might be appropriate for a distributed application with many clients, or where the client machine has spare CPU cycles.

Hybrid Approach

Of course, it is possible to combine these approaches. For some applications, it may be appropriate to use some compressed tables and some uncompressed tables. It may be best to externally compress some data (and store it in uncompressed InnoDB tables) and allow InnoDB to compress (some of) the other tables in the application. As always, up-front design and real-life testing are valuable in reaching the right decision.

Workload Characteristics and Compression

In addition to choosing which tables to compress (and the page size), the workload is another key determinant of performance. If the application is dominated by reads, rather than updates, fewer pages need to be reorganized and recompressed after the index page runs out of room for the per-page “modification log” that InnoDB maintains for compressed data. If the updates predominantly change non-indexed columns or those containing `BLOBS` or large strings that happen to be stored “off-page”, the overhead of compression may be acceptable. If the only changes to a table are `INSERTs` that use a monotonically increasing primary key, and there are few secondary indexes, there is little need to reorganize and recompress index pages. Since InnoDB can “delete-mark” and delete rows on compressed pages “in place” by modifying uncompressed data, `DELETE` operations on a table are relatively efficient.

For some environments, the time it takes to load data can be as important as run-time retrieval. Especially in data warehouse environments, many tables may be read-only or read-mostly. In those cases, it might or might not be acceptable to pay the price of compression in terms of increased load time, unless the resulting savings in fewer disk reads or in storage cost is significant.

Fundamentally, compression works best when the CPU time is available for compressing and uncompressing data. Thus, if your workload is I/O bound, rather than CPU-bound, you might find that compression can improve overall performance. When you test your application performance with different compression configurations, test on a platform similar to the planned configuration of the production system.

Configuration Characteristics and Compression

Reading and writing database pages from and to disk is the slowest aspect of system performance. Compression attempts to reduce I/O by using CPU time to compress and uncompress data, and is most effective when I/O is a relatively scarce resource compared to processor cycles.

This is often especially the case when running in a multi-user environment with fast, multi-core CPUs. When a page of a compressed table is in memory, InnoDB often uses an additional 16K in the buffer pool for an uncompressed copy of the page. The adaptive LRU algorithm in the InnoDB storage engine attempts to balance the use of memory between compressed and uncompressed pages to take into account whether the workload is running in an I/O-bound or CPU-bound manner. Still, a configuration with more memory dedicated to the InnoDB buffer pool tends to run better when using compressed tables than a configuration where memory is highly constrained.

Choosing the Compressed Page Size

The optimal setting of the compressed page size depends on the type and distribution of data that the table and its indexes contain. The compressed page size should always be bigger than the maximum record size, or operations may fail as noted in [Section 14.4.3.4, “Compression of B-Tree Pages”](#).

Setting the compressed page size too large wastes some space, but the pages do not have to be compressed as often. If the compressed page size is set too small, inserts or updates may require time-consuming recompression, and the B-tree nodes may have to be split more frequently, leading to bigger data files and less efficient indexing.

Typically, you set the compressed page size to 8K or 4K bytes. Given that the maximum InnoDB record size is around 8K, `KEY_BLOCK_SIZE=8` is usually a safe choice.

Monitoring Compression at Runtime

Overall application performance, CPU and I/O utilization and the size of disk files are good indicators of how effective compression is for your application.

To dig deeper into performance considerations for compressed tables, you can monitor compression performance at run time, using the Information Schema tables described in [Example 14.1, “Using the Compression Information Schema Tables”](#). These tables reflect the internal use of memory and the rates of compression used overall.

The `INNODB_CMP` tables report information about compression activity for each compressed page size (`KEY_BLOCK_SIZE`) in use. The information in these tables is system-wide, and includes summary data across all compressed tables in your database. You can use this data to help decide whether or not to compress a table by examining these tables when no other compressed tables are being accessed.

The key statistics to consider are the number of, and amount of time spent performing, compression and uncompression operations. Since InnoDB must split B-tree nodes when they are too full to contain the compressed data following a modification, compare the number of “successful” compression operations with the number of such operations overall. Based on the information in the `INNODB_CMP` tables and overall application performance and hardware resource utilization, you might make changes in your hardware configuration, adjust the size of the InnoDB buffer pool, choose a different page size, or select a different set of tables to compress.

If the amount of CPU time required for compressing and uncompressing is high, changing to faster CPUs, or those with more cores, can help improve performance with the same data, application workload and set of compressed tables. Increasing the size of the InnoDB buffer pool might also help performance, so that more uncompressed pages can stay in memory, reducing the need to uncompress pages that exist in memory only in compressed form.

A large number of compression operations overall (compared to the number of `INSERT`, `UPDATE` and `DELETE` operations in your application and the size of the database) could indicate that some of your compressed tables are being updated too heavily for effective compression. If so, choose a larger page size, or be more selective about which tables you compress.

If the number of “successful” compression operations (`COMPRESS_OPS_OK`) is a high percentage of the total number of compression operations (`COMPRESS_OPS`), then the system is likely performing well. If the ratio is low, then InnoDB is reorganizing, recompressing, and splitting B-tree nodes more often than is desirable. In this case, avoid compressing some tables, or increase `KEY_BLOCK_SIZE` for some of the compressed tables. You might turn off compression for tables that cause the number of “compression failures” in your application to be more than 1% or 2% of the total. (Such a failure ratio might be acceptable during a temporary operation such as a data load).

14.4.3.4. How Compression Works in InnoDB

This section describes some internal implementation details about compression in InnoDB. The information presented here may be helpful in tuning for performance, but is not necessary to know for basic use of compression.

Compression Algorithms

Some operating systems implement compression at the file system level. Files are typically divided into fixed-size blocks that are compressed into variable-size blocks, which easily leads into fragmentation. Every time something inside a block is modified, the whole block is recompressed before it is written to disk. These properties make this compression technique unsuitable for use in an update-intensive database system.

InnoDB implements compression with the help of the well-known [zlib library](#), which implements the LZ77 compression algorithm. This compression algorithm is mature, robust, and efficient in both CPU utilization and in reduction of data size. The algorithm is “lossless”, so that the original uncompressed data can always be reconstructed from the compressed form. LZ77 compression works by finding sequences of data that are repeated within the data to be compressed. The patterns of values in your data determine how well it compresses, but typical user data often compresses by 50% or more.

Unlike compression performed by an application, or compression features of some other database management systems, InnoDB compression applies both to user data and to indexes. In many cases, indexes can constitute 40-50% or more of the total database size, so this difference is significant. When compression is working well for a data set, the size of the InnoDB data files (the `.idb` files) is 25% to 50% of the uncompressed size or possibly smaller. Depending on the workload, this smaller database can in turn lead to a reduction in I/O, and an increase in throughput, at a modest cost in terms of increased CPU utilization.

InnoDB Data Storage and Compression

All user data in InnoDB is stored in pages comprising a B-tree index (the [clustered index](#)). In some other database systems, this type of index is called an “index-organized table”. Each row in the index node contains the values of the (user-specified or system-generated) primary key and all the other columns of the table.

Secondary indexes in InnoDB are also B-trees, containing pairs of values: the index key and a pointer to a row in the clustered index. The pointer is in fact the value of the primary key of the table, which is used to access the clustered index if columns other than the index key and primary key are required. Secondary index records must always fit on a single B-tree page.

The compression of B-tree nodes (of both clustered and secondary indexes) is handled differently from compression of overflow pages used to store long [VARCHAR](#), [BLOB](#), or [TEXT](#) columns, as explained in the following sections.

Compression of B-Tree Pages

Because they are frequently updated, B-tree pages require special treatment. It is important to minimize the number of times B-tree nodes are split, as well as to minimize the need to uncompress and recompress their content.

One technique InnoDB uses is to maintain some system information in the B-tree node in uncompressed form, thus facilitating certain in-place updates. For example, this allows rows to be delete-marked and deleted without any compression operation.

In addition, InnoDB attempts to avoid unnecessary uncompression and recompression of index pages when they are changed. Within each B-tree page, the system keeps an uncompressed “modification log” to record changes made to the page. Updates and inserts of small records may be written to this modification log without requiring the entire page to be completely reconstructed.

When the space for the modification log runs out, InnoDB uncompresses the page, applies the changes and recompresses the page. If recompression fails, the B-tree nodes are split and the process is repeated until the update or insert succeeds.

Generally, InnoDB requires that each B-tree page can accommodate at least two records. For compressed tables, this requirement has been relaxed. Leaf pages of B-tree nodes (whether of the primary key or secondary indexes) only need to accommodate one record, but that record must fit in uncompressed form, in the per-page modification log. Starting with InnoDB storage engine version 1.0.2, and if InnoDB strict mode is `ON`, the InnoDB storage engine checks the maximum row size during `CREATE TABLE` or `CREATE INDEX`. If the row does not fit, the following error message is issued: `ERROR HY000: Too big row`.

If you create a table when InnoDB strict mode is `OFF`, and a subsequent `INSERT` or `UPDATE` statement attempts to create an index entry that does not fit in the size of the compressed page, the operation fails with `ERROR 42000: Row size too large`. (This error message does not name the index for which the record is too large, or mention the length of the index record or the maximum record size on that particular index page.) To solve this problem, rebuild the table with `ALTER TABLE` and select a larger compressed page size (`KEY_BLOCK_SIZE`), shorten any column prefix indexes, or disable compression entirely with `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPACT`.

Compressing BLOB, VARCHAR and TEXT Columns

In a clustered index, `BLOB`, `VARCHAR` and `TEXT` columns that are not part of the primary key may be stored on separately allocated (“overflow”) pages. We call these `off-page columns` whose values are stored on singly-linked lists of overflow pages.

For tables created in `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED`, the values of `BLOB`, `TEXT` or `VARCHAR` columns may be stored fully off-page, depending on their length and the length of the entire row. For columns that are stored off-page, the clustered index record only contains 20-byte pointers to the overflow pages, one per column. Whether any columns are stored off-page depends on the page size and the total size of the row. When the row is too long to fit entirely within the page of the clustered index, InnoDB chooses the longest columns for off-page storage until the row fits on the clustered index page. As noted above, if a row does not fit by itself on a compressed page, an error occurs.

Tables created in previous versions of InnoDB use the Antelope file format, which supports only `ROW_FORMAT=REDUNDANT` and `ROW_FORMAT=COMPACT`. In these formats, InnoDB stores the first 768 bytes of `BLOB`, `VARCHAR` and `TEXT` columns in the clustered index record along with the primary key. The 768-byte prefix is followed by a 20-byte pointer to the overflow pages that contain the rest of the column value.

When a table is in `COMPRESSED` format, all data written to overflow pages is compressed “as is”; that is, InnoDB applies the zlib compression algorithm to the entire data item. Other than the data, compressed overflow pages contain an uncompressed header and trailer comprising a page checksum and a link to the next overflow page, among other things. Therefore, very significant storage savings can be obtained for longer `BLOB`, `TEXT` or `VARCHAR` columns if the data is highly compressible, as is often the case with text data (but not previously compressed images).

The overflow pages are of the same size as other pages. A row containing ten columns stored off-page occupies ten overflow pages, even if the total length of the columns is only 8K bytes. In an uncompressed table, ten uncompressed overflow pages occupy 160K bytes. In a compressed table with an 8K page size, they occupy only 80K bytes. Thus, it is often more efficient to use compressed table format for tables with long column values.

Using a 16K compressed page size can reduce storage and I/O costs for `BLOB`, `VARCHAR` or `TEXT` columns, because such data often compress well, and might therefore require fewer “overflow” pages, even though the B-tree nodes themselves take as many pages as in the uncompressed form.

Compression and the InnoDB Buffer Pool

In a compressed InnoDB table, every compressed page (whether 1K, 2K, 4K or 8K) corresponds to an uncompressed page of 16K bytes. To access the data in a page, InnoDB reads the compressed page from disk if it is not already in the buffer pool, then uncompresses the page to its original 16K byte form. This section describes how InnoDB manages the buffer pool with respect to pages of compressed tables.

To minimize I/O and to reduce the need to uncompress a page, at times the buffer pool contains both the compressed and uncompressed form of a database page. To make room for other required database pages, InnoDB may “evict” from the buffer pool an uncompressed page, while leaving the compressed page in memory. Or, if a page has not been accessed in a while, the compressed form of the page may be written to disk, to free space for other data. Thus, at any given time, the buffer pool may contain both the compressed and uncompressed forms of the page, or only the compressed form of the page, or neither.

InnoDB keeps track of which pages to keep in memory and which to evict using a least-recently-used (LRU) list, so that “hot” or frequently accessed data tends to stay in memory. When compressed tables are accessed, InnoDB uses an adaptive LRU algorithm to achieve an appropriate balance of compressed and uncompressed pages in memory. This adaptive algorithm is sensitive to whether the system is running in an I/O-bound or CPU-bound manner. The goal is to avoid spending too much processing time uncompressing pages when the CPU is busy, and to avoid doing excess I/O when the CPU has spare cycles that can be used for uncompressing compressed pages (that may already be in memory). When the system is I/O-bound, the algorithm prefers to evict the uncompressed copy of a page rather than both copies, to make more room for other disk pages to become memory resident. When the system is CPU-bound, InnoDB prefers to evict both the compressed and uncompressed page, so that more memory can be used for “hot” pages and reducing the need to uncompress data in memory only in compressed form.

Compression and the InnoDB Log Files

Before a compressed page is written to a database file, InnoDB writes a copy of the page to the redo log (if it has been recompressed since the last time it was written to the database). This is done to ensure that redo logs will always be usable, even if a future version of InnoDB uses a slightly different compression algorithm. Therefore, some increase in the size of log files, or a need for more frequent checkpoints, can be expected when using compression. The amount of increase in the log file size or checkpoint frequency depends on the number of times compressed pages are modified in a way that requires reorganization and recompression.

Note that the redo log file format (and the database file format) are different from previous releases when using compression. The MySQL Enterprise Backup product does support this latest Barracuda file format for compressed InnoDB tables. The older InnoDB Hot Backup product can only back up tables using the file format Antelope, and thus does not support InnoDB tables that use compression.

14.4.4. InnoDB File-Format Management

As InnoDB evolves, new on-disk data structures are sometimes required to support new features. Features such as compressed tables (see [Section 14.4.3, “InnoDB Data Compression”](#)), and long variable-length columns stored off-page (see [Section 14.4.5, “How InnoDB Stores Variable-Length Columns”](#)) require data file formats that are not compatible with prior versions of InnoDB. These features both require use of the new [Barracuda](#) file format.



Note

All other new features are compatible with the original [Antelope](#) file format and do not require the Barracuda file format.

This section discusses how to specify the file format for new InnoDB tables, compatibility of different file formats between MySQL releases,

Named File Formats. InnoDB 1.1 has the idea of a named file format and a configuration parameter to enable the use of features that require use of that format. The new file format is the [Barracuda](#) format, and the original InnoDB file format is called [Antelope](#). Compressed tables and the new row format that stores long columns “off-page” require the use of the Barracuda file format or newer. Future versions of InnoDB may introduce a series of file formats, [identified with the names of animals](#), in ascending alphabetic order.

14.4.4.1. Enabling File Formats

The configuration parameter `innodb_file_format` [\[1554\]](#) controls whether such statements as `CREATE TABLE` and `ALTER TABLE` can be used to create tables that depend on support for the [Barracuda](#) file format.

Although Oracle recommends using the Barracuda format for new tables where practical, in MySQL 5.5 the default file format is still [Antelope](#), for maximum compatibility with replication configurations containing different MySQL releases.

The file format is a dynamic, global parameter that can be specified in the MySQL option file (`my.cnf` or `my.ini`) or changed with the `SET GLOBAL` command.

14.4.4.2. Verifying File Format Compatibility

InnoDB 1.1 incorporates several checks to guard against the possible crashes and data corruptions that might occur if you run an older release of the MySQL server on InnoDB data files using a newer file format. These checks take place when the server is started, and when you first access a table. This section describes these checks, how you can control them, and error and warning conditions that might arise.

Backward Compatibility

Considerations of backward compatibility only apply when using a recent version of InnoDB (the InnoDB Plugin, or MySQL 5.5 and higher with InnoDB 1.1) alongside an older one (MySQL 5.1 or earlier, with the built-in InnoDB rather than the InnoDB Plugin). To minimize the chance of compatibility issues, you can standardize on the InnoDB Plugin for all your MySQL 5.1 and earlier database servers.

In general, a newer version of InnoDB may create a table or index that cannot safely be read or written with a prior version of InnoDB without risk of crashes, hangs, wrong results or corruptions. InnoDB 1.1 includes a mechanism to guard against these conditions, and to help preserve compatibility among database files and versions of InnoDB. This mechanism lets you take advantage of some new features of an InnoDB release (such as performance improvements and bug fixes), and still preserve the option of using your database with a prior version of InnoDB, by preventing accidental use of new features that create downward-incompatible disk files.

If a version of InnoDB supports a particular file format (whether or not that format is the default), you can query and update any table that requires that format or an earlier format. Only the creation of new tables using new features is limited based on the particular file format enabled. Conversely, if a tablespace contains a table or index that uses a file format that is not supported by the currently running software, it cannot be accessed at all, even for read access.

The only way to “downgrade” an InnoDB tablespace to an earlier file format is to copy the data to a new table, in a tablespace that uses the earlier format. This can be done with the `ALTER TABLE` statement, as described in [Section 14.4.4.4, “Downgrading the File Format”](#).

The easiest way to determine the file format of an existing InnoDB tablespace is to examine the properties of the table it contains, using the `SHOW TABLE STATUS` command or querying the table `INFORMATION_SCHEMA.TABLES`. If the `Row_format` of the table is reported as `'Compressed'` or `'Dynamic'`, the tablespace containing the table uses the Barracuda format. Otherwise, it uses the prior InnoDB file format, Antelope.

Internal Details

Every InnoDB per-table tablespace (represented by a `*.ibd` file) file is labeled with a file format identifier. The system tablespace (represented by the `ibdata` files) is tagged with the “highest” file format in use in a group of InnoDB database files, and this tag is checked when the files are opened.

Creating a compressed table, or a table with `ROW_FORMAT=DYNAMIC`, updates the file header for the corresponding `.ibd` file and the table type in the InnoDB data dictionary with the identifier for the Barracuda file format. From that point forward, the table cannot be used with a version of InnoDB that does not support this new file format. To protect against anomalous behavior, InnoDB version 5.0.21 and later performs a compatibility check when the table is opened. (In many cases, the `ALTER TABLE` statement recreates a table and thus changes its properties. The special case of adding or dropping indexes without rebuilding the table is described in [Section 14.4.2, “Fast Index Creation in the InnoDB Storage Engine”](#).)

Definition of ib-file set

To avoid confusion, for the purposes of this discussion we define the term “ib-file set” to mean the set of operating system files that InnoDB manages as a unit. The ib-file set includes the following files:

- The system tablespace (one or more `ibdata` files) that contain internal system information (including internal catalogs and undo information) and may include user data and indexes.
- Zero or more single-table tablespaces (also called “file per table” files, named `*.ibd` files).
- InnoDB log files; usually two, `ib_logfile0` and `ib_logfile1`. Used for crash recovery and in backups.

An “ib-file set” does not include the corresponding `.frm` files that contain metadata about InnoDB tables. The `.frm` files are created and managed by MySQL, and can sometimes get out of sync with the internal metadata in InnoDB.

Multiple tables, even from more than one database, can be stored in a single “ib-file set”. (In MySQL, a “database” is a logical collection of tables, what other systems refer to as a “schema” or “catalog”.)

14.4.4.2.1. Compatibility Check When InnoDB Is Started

To prevent possible crashes or data corruptions when InnoDB opens an ib-file set, it checks that it can fully support the file formats in use within the ib-file set. If the system is restarted following a crash, or a “fast shutdown” (i.e., `innodb_fast_shutdown` [1553] is greater than zero), there may be on-disk data structures (such as redo or undo entries, or doublewrite pages) that are in a “too-new” format for the current software. During the recovery process, serious damage can be done to your data files if these data structures are accessed. The startup check of the file format occurs before any recovery process begins, thereby preventing consistency issues with the new tables or startup problems for the MySQL server.

Beginning with version InnoDB 1.0.1, the system tablespace records an identifier or tag for the “highest” file format used by any table in any of the tablespaces that is part of the ib-file set. Checks against this file format tag are controlled by the configuration parameter `innodb_file_format_check` [1554], which is `ON` by default.

If the file format tag in the system tablespace is newer or higher than the highest version supported by the particular currently executing software and if `innodb_file_format_check` [1554] is `ON`, the following error is issued when the server is started:

```
InnoDB: Error: the system tablespace is in a
file format that this version doesn't support
```

You can also set `innodb_file_format` [1554] to a file format name. Doing so prevents InnoDB from starting if the current software does not support the file format specified. It also sets the “high water mark”

to the value you specify. The ability to set `innodb_file_format_check` [1554] will be useful (with future releases of InnoDB) if you manually “downgrade” all of the tables in an ib-file set (as described in [Section 14.4.11, “Downgrading the InnoDB Storage Engine”](#)). You can then rely on the file format check at startup if you subsequently use an older version of InnoDB to access the ib-file set.

In some limited circumstances, you might want to start the server and use an ib-file set that is in a “too new” format (one that is not supported by the software you are using). If you set the configuration parameter `innodb_file_format_check` [1554] to `OFF`, InnoDB opens the database, but issues this warning message in the error log:

```
InnoDB: Warning: the system tablespace is in a
file format that this version doesn't support
```



Note

This is a very dangerous setting, as it permits the recovery process to run, possibly corrupting your database if the previous shutdown was a crash or “fast shutdown”. You should only set `innodb_file_format_check` [1554] to `OFF` if you are sure that the previous shutdown was done with `innodb_fast_shutdown=0`, so that essentially no recovery process occurs. In a future release, this parameter setting may be renamed from `OFF` to `UNSAFE`. (However, until there are newer releases of InnoDB that support additional file formats, even disabling the startup checking is in fact “safe”.)

The parameter `innodb_file_format_check` [1554] affects only what happens when a database is opened, not subsequently. Conversely, the parameter `innodb_file_format` [1554] (which enables a specific format) only determines whether or not a new table can be created in the enabled format and has no effect on whether or not a database can be opened.

The file format tag is a “high water mark”, and as such it is increased after the server is started, if a table in a “higher” format is created or an existing table is accessed for read or write (assuming its format is supported). If you access an existing table in a format higher than the format the running software supports, the system tablespace tag is not updated, but table-level compatibility checking applies (and an error is issued), as described in [Section 14.4.4.2.2, “Compatibility Check When a Table Is Opened”](#). Any time the high water mark is updated, the value of `innodb_file_format_check` [1554] is updated as well, so the command `SELECT @@innodb_file_format_check;` displays the name of the newest file format known to be used by tables in the currently open ib-file set and supported by the currently executing software.

To best illustrate this behavior, consider the scenario described in [Table 14.6, “InnoDB Data File Compatibility and Related InnoDB Parameters”](#). Imagine that some future version of InnoDB supports the Cheetah format and that an ib-file set has been used with that version.

Table 14.6. InnoDB Data File Compatibility and Related InnoDB Parameters

innodb file format check	innodb file format	Highest file format used in ib-file set	Highest file format supported by InnoDB	Result
OFF	Antelope or Barracuda	Barracuda	Barracuda	Database can be opened; tables can be created which require Antelope or Barracuda file format
OFF	Antelope or Barracuda	Cheetah	Barracuda	Database can be opened with a warning, since the database contains files in a “too new” format; tables can be created in Antelope or Barracuda file format; tables in Cheetah format cannot be accessed

innodb file format check	innodb file format	Highest file format used in ib-file set	Highest file format supported by InnoDB	Result
OFF	Cheetah	Barracuda	Barracuda	Database cannot be opened; <code>innodb_file_format [1554]</code> cannot be set to Cheetah
ON	Antelope or Barracuda	Barracuda	Barracuda	Database can be opened; tables can be created in Antelope or Barracuda file format
ON	Antelope or Barracuda	Cheetah	Barracuda	Database cannot be opened, since the database contains files in a “too new” format (Cheetah)
ON	Cheetah	Barracuda	Barracuda	Database cannot be opened; <code>innodb_file_format [1554]</code> cannot be set to Cheetah

14.4.4.2.2. Compatibility Check When a Table Is Opened

When a table is first accessed, InnoDB (including some releases prior to InnoDB 1.0) checks that the file format of the tablespace in which the table is stored is fully supported. This check prevents crashes or corruptions that would otherwise occur when tables using a “too new” data structure are encountered.

All tables using any file format supported by a release can be read or written (assuming the user has sufficient privileges). The setting of the system configuration parameter `innodb_file_format [1554]` can prevent creating a new table that uses specific file formats, even if they are supported by a given release. Such a setting might be used to preserve backward compatibility, but it does not prevent accessing any table that uses any supported format.

As noted in [Named File Formats](#), versions of MySQL older than 5.0.21 cannot reliably use database files created by newer versions if a new file format was used when a table was created. To prevent various error conditions or corruptions, InnoDB checks file format compatibility when it opens a file (for example, upon first access to a table). If the currently running version of InnoDB does not support the file format identified by the table type in the InnoDB data dictionary, MySQL reports the following error:

```
ERROR 1146 (42S02): Table 'test.t1' doesn't exist
```

InnoDB also writes a message to the error log:

```
InnoDB: table test/t1: unknown table type 33
```

The table type should be equal to the tablespace flags, which contains the file format version as discussed in [Section 14.4.4.3, “Identifying the File Format in Use”](#).

Versions of InnoDB prior to MySQL 4.1 did not include table format identifiers in the database files, and versions prior to MySQL 5.0.21 did not include a table format compatibility check. Therefore, there is no way to ensure proper operations if a table in a “too new” format is used with versions of InnoDB prior to 5.0.21.

The file format management capability in InnoDB 1.0 and higher (tablespace tagging and run-time checks) allows InnoDB to verify as soon as possible that the running version of software can properly process the tables existing in the database.

If you permit InnoDB to open a database containing files in a format it does not support (by setting the parameter `innodb_file_format_check [1554]` to `OFF`), the table-level checking described in this section still applies.

Users are *strongly* urged not to use database files that contain Barracuda file format tables with releases of InnoDB older than the MySQL 5.1 with the InnoDB Plugin. It is possible to “downgrade” such tables to the Antelope format with the procedure described in [Section 14.4.4.4, “Downgrading the File Format”](#).

14.4.4.3. Identifying the File Format in Use

After you enable a given `innodb_file_format` [1554], this change applies only to newly created tables rather than existing ones. If you do create a new table, the tablespace containing the table is tagged with the “earliest” or “simplest” file format that is required for the table's features. For example, if you enable file format Barracuda, and create a new table that is not compressed and does not use `ROW_FORMAT=DYNAMIC`, the new tablespace that contains the table is tagged as using file format Antelope.

It is easy to identify the file format used by a given tablespace or table. The table uses the Barracuda format if the `Row_format` reported by `SHOW CREATE TABLE` or `INFORMATION_SCHEMA.TABLES` is one of 'Compressed' or 'Dynamic'. (The `Row_format` is a separate column; ignore the contents of the `Create_options` column, which may contain the string `ROW_FORMAT`.) If the table in a tablespace uses neither of those features, the file uses the format supported by prior releases of InnoDB, now called file format Antelope. Then, the `Row_format` is one of 'Redundant' or 'Compact'.

Internal Details

The file format identifier is written as part of the tablespace flags (a 32-bit number) in the `*.ibd` file in the 4 bytes starting at position 54 of the file, most significant byte first. (The first byte of the file is byte zero.) On some systems, you can display these bytes in hexadecimal with the command `od -t x1 -j 54 -N 4 tablename.ibd`. If all bytes are zero, the tablespace uses the Antelope file format (which is the format used by the standard InnoDB storage engine up to version 5.1). Otherwise, the least significant bit should be set in the tablespace flags, and the file format identifier is written in the bits 5 through 11. (Divide the tablespace flags by 32 and take the remainder after dividing the integer part of the result by 128.)

14.4.4.4. Downgrading the File Format

Each InnoDB tablespace file (with a name matching `*.ibd`) is tagged with the file format used to create its table and indexes. The way to downgrade the tablespace is to re-create the table and its indexes. The easiest way to recreate a table and its indexes is to use the command:

```
ALTER TABLE t ROW_FORMAT=COMPACT;
```

on each table that you want to downgrade. The `COMPACT` row format uses the file format Antelope. It was introduced in MySQL 5.0.3.

14.4.4.5. Future InnoDB File Formats

The file format used by the standard built-in InnoDB in MySQL 5.1 is the [Antelope](#) format. The file format introduced with InnoDB Plugin 1.0 is the [Barracuda](#) format. Although no new features have been announced that would require additional new file formats, the InnoDB file format mechanism allows for future enhancements.

For the sake of completeness, these are the file format names that might be used for future file formats: Antelope, Barracuda, Cheetah, Dragon, Elk, Fox, Gazelle, Hornet, Impala, Jaguar, Kangaroo, Leopard, Moose, Nautilus, Ocelot, Porpoise, Quail, Rabbit, Shark, Tiger, Urchin, Viper, Whale, Xenops, Yak and Zebra. These file formats correspond to the internal identifiers 0..25.

14.4.5. How InnoDB Stores Variable-Length Columns

This section discusses how certain InnoDB features, such as table [compression](#) and off-page storage of long columns, are controlled by the `ROW_FORMAT` clause of the `CREATE TABLE` statement. It discusses

considerations for choosing the right row format and compatibility of row formats between MySQL releases.

14.4.5.1. Overview of InnoDB Row Storage

The storage for rows and associated columns affects performance for queries and DML operations. As more rows fit into a single disk page, queries and index lookups can work faster, less cache memory is required in the InnoDB buffer pool, and less I/O is required to write out updated values for the numeric and short string columns.

All data in InnoDB is stored in database pages that make up a [B-tree index](#) (the [clustered index](#) organized according to the [primary key](#) columns). Table data and indexes both use this type of structure. The nodes of the index data structure contain the values of all the columns in that row (for the clustered index) or the index columns and the primary key columns (for secondary indexes).

Variable-length columns are an exception to this rule. Columns such as [BLOB](#) and [VARCHAR](#) that are too long to fit on a B-tree page are stored on separately allocated disk pages called [overflow pages](#). We call such columns [off-page column](#). The values of these columns are stored on singly-linked lists of overflow pages, and each such column has its own list of one or more overflow pages. In some cases, all or a prefix of the long column value is stored in the B-tree, to avoid wasting storage and eliminating the need to read a separate page.

The [Barracuda](#) file format provides a new option ([KEY_BLOCK_SIZE](#)) to control how much column data is stored in the clustered index, and how much is placed on overflow pages.

14.4.5.2. Specifying the Row Format for a Table

You specify the row format for a table with the [ROW_FORMAT](#) clause of the [CREATE TABLE](#) and [ALTER TABLE](#) statements.

14.4.5.3. Barracuda File Format: [DYNAMIC](#) and [COMPRESSED](#) Row Formats

When [innodb_file_format](#) [1554] is set to Barracuda and a table is created with [ROW_FORMAT=DYNAMIC](#) or [ROW_FORMAT=COMPRESSED](#), long column values are stored fully off-page, and the clustered index record contains only a 20-byte pointer to the overflow page.

Whether any columns are stored off-page depends on the page size and the total size of the row. When the row is too long, InnoDB chooses the longest columns for off-page storage until the clustered index record fits on the B-tree page.

The [DYNAMIC](#) row format maintains the efficiency of storing the entire row in the index node if it fits (as do the [COMPACT](#) and [REDUNDANT](#) formats), but this new format avoids the problem of filling B-tree nodes with a large number of data bytes of long columns. The [DYNAMIC](#) format is based on the idea that if a portion of a long data value is stored off-page, it is usually most efficient to store all of the value off-page. With [DYNAMIC](#) format, shorter columns are likely to remain in the B-tree node, minimizing the number of overflow pages needed for any given row.

The [COMPRESSED](#) row format uses similar internal details for off-page storage as the [DYNAMIC](#) row format, with additional storage and performance considerations from the table and index data being compressed and using smaller page sizes. For full details about the [COMPRESSED](#) row format, see [Section 14.4.3, “InnoDB Data Compression”](#).

14.4.5.4. Antelope File Format: [COMPACT](#) and [REDUNDANT](#) Row Formats

Early versions of InnoDB used an unnamed file format (now called [Antelope](#)) for database files. With that format, tables were defined with [ROW_FORMAT=COMPACT](#) (or [ROW_FORMAT=REDUNDANT](#)) and InnoDB

stored up to the first 768 bytes of variable-length columns (such as `BLOB` and `VARCHAR`) in the index record within the B-tree node, with the remainder stored on the overflow pages.

To preserve compatibility with those prior versions, tables created with the newest InnoDB use the prefix format, unless one of `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED` is specified (or implied) on the `CREATE TABLE` statement.

With the Antelope file format, if the value of a column is 768 bytes or less, no overflow page is needed, and some savings in I/O may result, since the value is in the B-tree node. This works well for relatively short `BLOB` values, but may cause B-tree nodes to fill with data rather than key values, reducing their efficiency. Tables with many `BLOB` columns could cause B-tree nodes to become too full of data, and contain too few rows, making the entire index less efficient than if the rows were shorter or if the column values were stored off-page.

14.4.6. InnoDB `INFORMATION_SCHEMA` tables

The `INFORMATION_SCHEMA` is a MySQL feature that helps you monitor server activity to diagnose capacity and performance issues. Several InnoDB-related `INFORMATION_SCHEMA` tables (`INNODB_CMP`, `INNODB_CMP_RESET`, `INNODB_CMPMEM`, `INNODB_CMPMEM_RESET`, `INNODB_TRX`, `INNODB_LOCKS` and `INNODB_LOCK_WAITS`) contain live information about compressed InnoDB tables, the compressed InnoDB buffer pool, all transactions currently executing inside InnoDB, the locks that transactions hold and those that are blocking transactions waiting for access to a resource (a table or row).

The Information Schema tables are themselves plugins to the MySQL server, and must be activated by `INSTALL` statements. If they are installed, but the InnoDB storage engine plugin is not installed, these tables appear to be empty.

This section describes the InnoDB-related Information Schema tables and shows some examples of their use.

14.4.6.1. Information Schema Tables about Compression

Two new pairs of Information Schema tables provided by the InnoDB storage engine can give you some insight into how well compression is working overall. One pair of tables contains information about the number of compression operations and the amount of time spent performing compression. Another pair of tables contains information on the way memory is allocated for compression.

14.4.6.1.1. `INNODB_CMP` and `INNODB_CMP_RESET`

The tables `INNODB_CMP` and `INNODB_CMP_RESET` contain status information on the operations related to compressed tables, which are covered in [Section 14.4.3, “InnoDB Data Compression”](#). The compressed page size is in the column `PAGE_SIZE`.

These two tables have identical contents, but reading from `INNODB_CMP_RESET` resets the statistics on compression and uncompression operations. For example, if you archive the output of `INNODB_CMP_RESET` every 60 minutes, you see the statistics for each hourly period. If you monitor the output of `INNODB_CMP` (making sure never to read `INNODB_CMP_RESET`), you see the cumulated statistics since InnoDB was started.

For the table definition, see [Table 20.1, “Columns of `INNODB_CMP` and `INNODB_CMP_RESET`”](#).

14.4.6.1.2. `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET`

The tables `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` contain status information on the compressed pages that reside in the buffer pool. Please consult [Section 14.4.3, “InnoDB Data Compression”](#) for

further information on compressed tables and the use of the buffer pool. The tables `INNODB_CMP` and `INNODB_CMP_RESET` should provide more useful statistics on compression.

Internal Details

The InnoDB storage engine uses a so-called “buddy allocator” system to manage memory allocated to pages of various sizes, from 1KB to 16KB. Each row of the two tables described here corresponds to a single page size.

These two tables have identical contents, but reading from `INNODB_CMPMEM_RESET` resets the statistics on relocation operations. For example, if every 60 minutes you archived the output of `INNODB_CMPMEM_RESET`, it would show the hourly statistics. If you never read `INNODB_CMPMEM_RESET` and monitored the output of `INNODB_CMPMEM` instead, it would show the cumulated statistics since InnoDB was started.

For the table definition, see [Table 20.2, “Columns of INNODB_CMPMEM and INNODB_CMPMEM_RESET”](#).

14.4.6.1.3. Using the Compression Information Schema Tables

Example 14.1. Using the Compression Information Schema Tables

The following is sample output from a database that contains compressed tables (see [Section 14.4.3, “InnoDB Data Compression”](#), `INNODB_CMP`, and `INNODB_CMPMEM`).

The following table shows the contents of `INFORMATION_SCHEMA.INNODB_CMP` under a light [workload](#). The only compressed page size that the buffer pool contains is 8K. Compressing or uncompressing pages has consumed less than a second since the time the statistics were reset, because the columns `COMPRESS_TIME` and `UNCOMPRESS_TIME` are zero.

page size	compress ops	compress ops ok	compress time	uncompress ops	uncompress time
1024	0	0	0	0	0
2048	0	0	0	0	0
4096	0	0	0	0	0
8192	1048	921	0	61	0
16384	0	0	0	0	0

According to `INNODB_CMPMEM`, there are 6169 compressed 8KB pages in the buffer pool.

The following table shows the contents of `INFORMATION_SCHEMA.INNODB_CMPMEM` under a light [workload](#). Some memory is unusable due to fragmentation of the InnoDB memory allocator for compressed pages: `SUM(PAGE_SIZE*PAGES_FREE)=6784`. This is because small memory allocation requests are fulfilled by splitting bigger blocks, starting from the 16K blocks that are allocated from the main buffer pool, using the buddy allocation system. The fragmentation is this low, because some allocated blocks have been relocated (copied) to form bigger adjacent free blocks. This copying of `SUM(PAGE_SIZE*RELOCATION_OPS)` bytes has consumed less than a second (`SUM(RELOCATION_TIME)=0`).

page size	pages used	pages free	relocation ops	relocation time
1024	0	0	0	0
2048	0	1	0	0
4096	0	1	0	0
8192	6169	0	5	0

page size	pages used	pages free	relocation ops	relocation time
16384	0	0	0	0

14.4.6.2. Information Schema Tables about Transactions

Three InnoDB-related Information Schema tables make it easy to monitor transactions and diagnose possible locking problems. The three tables are `INNODB_TRX`, `INNODB_LOCKS` and `INNODB_LOCK_WAITS`.

14.4.6.2.1. `INNODB_TRX`

Contains information about every transaction currently executing inside InnoDB, including whether the transaction is waiting for a lock, when the transaction started, and the particular SQL statement the transaction is executing.

For the table definition, see [Table 20.3, “`INNODB_TRX` Columns”](#).

14.4.6.2.2. `INNODB_LOCKS`

Each transaction in InnoDB that is waiting for another transaction to release a lock (`INNODB_TRX.TRX_STATE= 'LOCK WAIT'`) is blocked by exactly one “blocking lock request”. That blocking lock request is for a row or table lock held by another transaction in an incompatible mode. The waiting or blocked transaction cannot proceed until the other transaction commits or rolls back, thereby releasing the requested lock. For every blocked transaction, `INNODB_LOCKS` contains one row that describes each lock the transaction has requested, and for which it is waiting. `INNODB_LOCKS` also contains one row for each lock that is blocking another transaction, whatever the state of the transaction that holds the lock (`'RUNNING'`, `'LOCK WAIT'`, `'ROLLING BACK'` or `'COMMITTING'`). The lock that is blocking a transaction is always held in a mode (read vs. write, shared vs. exclusive) incompatible with the mode of requested lock.

For the table definition, see [Table 20.4, “`INNODB_LOCKS` Columns”](#).

14.4.6.2.3. `INNODB_LOCK_WAITS`

Using this table, you can tell which transactions are waiting for a given lock, or for which lock a given transaction is waiting. This table contains one or more rows for each *blocked* transaction, indicating the lock it has requested and the lock(s) that is (are) blocking that request. The `REQUESTED_LOCK_ID` refers to the lock that a transaction is requesting, and the `BLOCKING_LOCK_ID` refers to the lock (held by another transaction) that is preventing the first transaction from proceeding. For any given blocked transaction, all rows in `INNODB_LOCK_WAITS` have the same value for `REQUESTED_LOCK_ID` and different values for `BLOCKING_LOCK_ID`.

For the table definition, see [Table 20.5, “`INNODB_LOCK_WAITS` Columns”](#).

14.4.6.2.4. Using the Transaction Information Schema Tables

Example 14.2. Identifying Blocking Transactions

It is sometimes helpful to be able to identify which transaction is blocking another. You can use the Information Schema tables to find out which transaction is waiting for another, and which resource is being requested.

Suppose you have the following scenario, with three users running concurrently. Each user (or session) corresponds to a MySQL thread, and executes one transaction after another. Consider the state of the system when these users have issued the following commands, but none has yet committed its transaction:

- User A:

```
BEGIN;
SELECT a FROM t FOR UPDATE;
SELECT SLEEP(100);
```

- User B:

```
SELECT b FROM t FOR UPDATE;
```

- User C:

```
SELECT c FROM t FOR UPDATE;
```

In this scenario, you may use this query to see who is waiting for whom:

```
SELECT r.trx_id waiting_trx_id,
       r.trx_mysql_thread_id waiting_thread,
       r.trx_query waiting_query,
       b.trx_id blocking_trx_id,
       b.trx_mysql_thread_id blocking_thread,
       b.trx_query blocking_query
FROM   information_schema.innodb_lock_waits w
INNER JOIN information_schema.innodb_trx b ON
        b.trx_id = w.blocking_trx_id
INNER JOIN information_schema.innodb_trx r ON
        r.trx_id = w.requesting_trx_id;
```

waiting trx id	waiting thread	waiting query	blocking trx id	blocking thread	blocking query
A4	6	SELECT b FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A4	6	SELECT b FROM t FOR UPDATE

In the above result, you can identify users by the “waiting query” or “blocking query”. As you can see:

- User B (trx id 'A4', thread 6) and User C (trx id 'A5', thread 7) are both waiting for User A (trx id 'A3', thread 5).
- User C is waiting for User B as well as User A.

You can see the underlying data in the tables [INNODB_TRX](#), [INNODB_LOCKS](#), and [INNODB_LOCK_WAITS](#).

The following table shows some sample contents of INFORMATION_SCHEMA.INNODB_TRX.

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
A3	RUNNING	2008-01-15 16:44:54	NULL	NULL	2	5	SELECT SLEEP(100)
A4	LOCK WAIT	2008-01-15 16:45:09	A4:1:3:2	2008-01-15 16:45:09	2	6	SELECT b FROM t FOR UPDATE
A5	LOCK WAIT	2008-01-15 16:45:14	A5:1:3:2	2008-01-15 16:45:14	2	7	SELECT c FROM t FOR UPDATE

The following table shows some sample contents of `INFORMATION_SCHEMA.INNODB_LOCKS`.

lock id	lock trx id	lock mode	lock type	lock table	lock index	lock space	lock page	lock rec	lock data
A3:1:3:2	A3	X	RECORD	`test`.`t`	`PRIMARY`	1	3	2	0x0200
A4:1:3:2	A4	X	RECORD	`test`.`t`	`PRIMARY`	1	3	2	0x0200
A5:1:3:2	A5	X	RECORD	`test`.`t`	`PRIMARY`	1	3	2	0x0200

The following table shows some sample contents of `INFORMATION_SCHEMA.INNODB_LOCK_WAITS`.

requesting trx id	requested lock id	blocking trx id	blocking lock id
A4	A4:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A4	A4:1:3:2

Example 14.3. More Complex Example of Transaction Data in Information Schema Tables

Sometimes you would like to correlate the internal InnoDB locking information with session-level information maintained by MySQL. For example, you might like to know, for a given InnoDB transaction ID, the corresponding MySQL session ID and name of the user that may be holding a lock, and thus blocking another transaction.

The following output from the `INFORMATION_SCHEMA` tables is taken from a somewhat loaded system.

As can be seen in the following tables, there are several transactions running.

The following `INNODB_LOCKS` and `INNODB_LOCK_WAITS` tables shows that:

- Transaction `77F` (executing an `INSERT`) is waiting for transactions `77E`, `77D` and `77B` to commit.
- Transaction `77E` (executing an `INSERT`) is waiting for transactions `77D` and `77B` to commit.
- Transaction `77D` (executing an `INSERT`) is waiting for transaction `77B` to commit.
- Transaction `77B` (executing an `INSERT`) is waiting for transaction `77A` to commit.
- Transaction `77A` is running, currently executing `SELECT`.
- Transaction `E56` (executing an `INSERT`) is waiting for transaction `E55` to commit.
- Transaction `E55` (executing an `INSERT`) is waiting for transaction `19C` to commit.
- Transaction `19C` is running, currently executing an `INSERT`.

Note that there may be an inconsistency between queries shown in the two tables `INNODB_TRX.TRX_QUERY` and `PROCESSLIST.INFO`. The current transaction ID for a thread, and the query being executed in that transaction, may be different in these two tables for any given thread. See [Section 14.4.6.3.3, “Possible Inconsistency with `PROCESSLIST`”](#) for an explanation.

The following table shows the contents of `INFORMATION_SCHEMA.PROCESSLIST` in a system running a heavy [workload](#).

ID	USER	HOST	DB	COMMAND	TIME	STATE	INFO
384	root	localhost	test	Query	10	update	insert into t2 values ...

InnoDB INFORMATION_SCHEMA tables

ID	USER	HOST	DB	COMMAND	TIME	STATE	INFO
257	root	localhost	test	Query	3	update	insert into t2 values ...
130	root	localhost	test	Query	0	update	insert into t2 values ...
61	root	localhost	test	Query	1	update	insert into t2 values ...
8	root	localhost	test	Query	1	update	insert into t2 values ...
4	root	localhost	test	Query	0	preparing	SELECT * FROM processlist
2	root	localhost	test	Sleep	566		NULL

The following table shows the contents of `INFORMATION_SCHEMA.INNODB_TRX` in a system running a heavy [workload](#).

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
77F	LOCK WAIT	2008-01-15 13:10:16	77F:806	2008-01-15 13:10:16	1	876	insert into t09 (D, B, C) values ...
77E	LOCK WAIT	2008-01-15 13:10:16	77E:806	2008-01-15 13:10:16	1	875	insert into t09 (D, B, C) values ...
77D	LOCK WAIT	2008-01-15 13:10:16	77D:806	2008-01-15 13:10:16	1	874	insert into t09 (D, B, C) values ...
77B	LOCK WAIT	2008-01-15 13:10:16	77B:733:12:1	2008-01-15 13:10:16	4	873	insert into t09 (D, B, C) values ...
77A	RUNNING	2008-01-15 13:10:16	NULL	NULL	4	872	select b, c from t09 where ...
E56	LOCK WAIT	2008-01-15 13:10:06	E56:743:6:2	2008-01-15 13:10:06	5	384	insert into t2 values ...
E55	LOCK WAIT	2008-01-15 13:10:06	E55:743:38:2	2008-01-15 13:10:13	965	257	insert into t2 values ...
19C	RUNNING	2008-01-15 13:09:10	NULL	NULL	2900	130	insert into t2 values ...
E15	RUNNING	2008-01-15 13:08:59	NULL	NULL	5395	61	insert into t2 values ...
51D	RUNNING	2008-01-15 13:08:47	NULL	NULL	9807	8	insert into t2 values ...

The following table shows the contents of `INFORMATION_SCHEMA.INNODB_LOCK_WAITS` in a system running a heavy [workload](#).

requesting trx id	requested lock id	blocking trx id	blocking lock id
77F	77F:806	77E	77E:806
77F	77F:806	77D	77D:806
77F	77F:806	77B	77B:806
77E	77E:806	77D	77D:806
77E	77E:806	77B	77B:806
77D	77D:806	77B	77B:806
77B	77B:733:12:1	77A	77A:733:12:1
E56	E56:743:6:2	E55	E55:743:6:2
E55	E55:743:38:2	19C	19C:743:38:2

The following table shows the contents of `INFORMATION_SCHEMA.INNODB_LOCKS` in a system running a heavy workload.

lock id	lock trx id	lock mode	lock type	lock table	lock index	lock space	lock page	lock rec	lock data
77F:806	77F	AUTO_INC	TABLE	`test`. .`t09`	NULL	NULL	NULL	NULL	NULL
77E:806	77E	AUTO_INC	TABLE	`test`. .`t09`	NULL	NULL	NULL	NULL	NULL
77D:806	77D	AUTO_INC	TABLE	`test`. .`t09`	NULL	NULL	NULL	NULL	NULL
77B:806	77B	AUTO_INC	TABLE	`test`. .`t09`	NULL	NULL	NULL	NULL	NULL
77B:733:12:1	77B	X	RECORD	`test`. .`t09`	`PRIMARY`	733	12	1	supremum pseudo- record
77A:733:12:1	77A	X	RECORD	`test`. .`t09`	`PRIMARY`	733	12	1	supremum pseudo- record
E56:743:6:2	E56	S	RECORD	`test`. .`t2`	`PRIMARY`	743	6	2	0, 0
E55:743:6:2	E55	X	RECORD	`test`. .`t2`	`PRIMARY`	743	6	2	0, 0
E55:743:38:2	E55	S	RECORD	`test`. .`t2`	`PRIMARY`	743	38	2	1922, 1922
19C:743:38:2	19C	X	RECORD	`test`. .`t2`	`PRIMARY`	743	38	2	1922, 1922

14.4.6.3. Special Locking Considerations for InnoDB INFORMATION_SCHEMA Tables

14.4.6.3.1. Understanding InnoDB Locking

When a transaction updates a row in a table, or locks it with `SELECT FOR UPDATE`, InnoDB establishes a list or queue of locks on that row. Similarly, InnoDB maintains a list of locks on a table for table-level locks transactions hold. If a second transaction wants to update a row or lock a table already locked by a prior

transaction in an incompatible mode, InnoDB adds a lock request for the row to the corresponding queue. For a lock to be acquired by a transaction, all incompatible lock requests previously entered into the lock queue for that row or table must be removed (the transactions holding or requesting those locks either commit or roll back).

A transaction may have any number of lock requests for different rows or tables. At any given time, a transaction may be requesting a lock that is held by another transaction, in which case it is blocked by that other transaction. The requesting transaction must wait for the transaction that holds the blocking lock to commit or rollback. If a transaction is not waiting for a lock, it is in the `'RUNNING'` state. If a transaction is waiting for a lock, it is in the `'LOCK WAIT'` state.

The table `INNODB_LOCKS` holds one or more row for each `'LOCK WAIT'` transaction, indicating the lock request(s) that is (are) preventing its progress. This table also contains one row describing each lock in a queue of locks pending for a given row or table. The table `INNODB_LOCK_WAITS` shows which locks already held by a transaction are blocking locks requested by other transactions.

14.4.6.3.2. Granularity of `INFORMATION_SCHEMA` Data

The data exposed by the transaction and locking tables represent a glimpse into fast-changing data. This is not like other (user) tables, where the data only changes when application-initiated updates occur. The underlying data is internal system-managed data, and can change very quickly.

For performance reasons, and to minimize the chance of misleading `JOINS` between the `INFORMATION_SCHEMA` tables, InnoDB collects the required transaction and locking information into an intermediate buffer whenever a `SELECT` on any of the tables is issued. This buffer is refreshed only if more than 0.1 seconds has elapsed since the last time the buffer was read. The data needed to fill the three tables is fetched atomically and consistently and is saved in this global internal buffer, forming a point-in-time “snapshot”. If multiple table accesses occur within 0.1 seconds (as they almost certainly do when MySQL processes a join among these tables), then the same snapshot is used to satisfy the query.

A correct result is returned when you `JOIN` any of these tables together in a single query, because the data for the three tables comes from the same snapshot. Because the buffer is not refreshed with every query of any of these tables, if you issue separate queries against these tables within a tenth of a second, the results are the same from query to query. On the other hand, two separate queries of the same or different tables issued more than a tenth of a second apart may see different results, since the data come from different snapshots.

Because InnoDB must temporarily stall while the transaction and locking data is collected, too frequent queries of these tables can negatively impact performance as seen by other users.

As these tables contain sensitive information (at least `INNODB_LOCKS.LOCK_DATA` and `INNODB_TRX.TRX_QUERY`), for security reasons, only the users with the `PROCESS` privilege are allowed to `SELECT` from them.

14.4.6.3.3. Possible Inconsistency with `PROCESSLIST`

As just described, while the transaction and locking data is correct and consistent when these `INFORMATION_SCHEMA` tables are populated, the underlying data changes so fast that similar glimpses at other, similarly fast-changing data, may not be in sync. Thus, you should be careful in comparing the data in the InnoDB transaction and locking tables with that in the MySQL table `PROCESSLIST`. The data from the `PROCESSLIST` table does not come from the same snapshot as the data about locking and transactions. Even if you issue a single `SELECT` (`JOINing` `INNODB_TRX` and `PROCESSLIST`, for example), the content of those tables is generally not consistent. `INNODB_TRX` may reference rows that are not present in `PROCESSLIST` or the currently executing SQL query of a transaction, shown in `INNODB_TRX.TRX_QUERY` may be different from the one in `PROCESSLIST.INFO`. The

query in `INNODB_TRX` is always consistent with the rest of `INNODB_TRX`, `INNODB_LOCKS` and `INNODB_LOCK_WAITS` when the data comes from the same snapshot.

14.4.7. InnoDB Performance and Scalability Enhancements

This section discusses recent InnoDB enhancements to performance and scalability, covering the performance features in InnoDB 1.1 with MySQL 5.5, and the features in the InnoDB Plugin for MySQL 5.1. This information is useful to any DBA or developer who is concerned with performance and scalability. Although some of the enhancements do not require any action on your part, knowing this information can still help you diagnose performance issues more quickly and modernize systems and applications that rely on older, inefficient behavior.

14.4.7.1. Overview of InnoDB Performance

InnoDB has always been highly efficient, and includes several unique architectural elements to assure high performance and scalability. The latest InnoDB storage engine includes new features that take advantage of advances in operating systems and hardware platforms, such as multi-core processors and improved memory allocation systems. In addition, new configuration options let you better control some InnoDB internal subsystems to achieve the best performance with your workload.

Starting with MySQL 5.5 and InnoDB 1.1, the built-in InnoDB storage engine within MySQL is upgraded to the full feature set and performance of the former InnoDB Plugin. This change makes these performance and scalability enhancements available to a much wider audience than before, and eliminates the separate installation step of the InnoDB Plugin. After learning about the InnoDB performance features in this section, continue with [Chapter 8, Optimization](#) to learn the best practices for overall MySQL performance, and [Section 8.5, “Optimizing for InnoDB Tables”](#) in particular for InnoDB tips and guidelines.

14.4.7.2. Faster Locking for Improved Scalability

In MySQL and InnoDB, multiple threads of execution access shared data structures. InnoDB synchronizes these accesses with its own implementation of [mutexes](#) and [read/write locks](#). InnoDB has historically protected the internal state of a read/write lock with an InnoDB mutex. On Unix and Linux platforms, the internal state of an InnoDB mutex is protected by a [Pthreads](#) mutex, as in IEEE Std 1003.1c (POSIX.1c).

On many platforms, there is a more efficient way to implement mutexes and read/write locks. [Atomic](#) operations can often be used to synchronize the actions of multiple threads more efficiently than Pthreads. Each operation to acquire or release a lock can be done in fewer CPU instructions, and thus result in less wasted time when threads are contending for access to shared data structures. This in turn means greater scalability on multi-core platforms.

InnoDB implements mutexes and read/write locks with the [built-in functions provided by the GNU Compiler Collection \(GCC\) for atomic memory access](#) instead of using the Pthreads approach previously used. More specifically, an InnoDB that is compiled with GCC version 4.1.2 or later uses the atomic builtins instead of a `pthread_mutex_t` to implement InnoDB mutexes and read/write locks.

On 32-bit Microsoft Windows, InnoDB has implemented mutexes (but not read/write locks) with hand-written assembler instructions. Beginning with Microsoft Windows 2000, functions for [Interlocked Variable Access](#) are available that are similar to the built-in functions provided by GCC. On Windows 2000 and higher, InnoDB makes use of the Interlocked functions. Unlike the old hand-written assembler code, the new implementation supports read/write locks and 64-bit platforms.

Solaris 10 introduced library functions for [atomic operations](#), and InnoDB uses these functions by default. When MySQL is compiled on Solaris 10 with a compiler that does not support the [built-in functions provided by the GNU Compiler Collection \(GCC\) for atomic memory access](#), InnoDB uses the library functions.

This change improves the scalability of InnoDB on multi-core systems. This feature is enabled out-of-the-box on the platforms where it is supported. You do not have to set any parameter or option to take advantage of the improved performance. On platforms where the GCC, Windows, or Solaris functions for atomic memory access are not available, InnoDB uses the traditional Pthreads method of implementing mutexes and read/write locks.

When MySQL starts, InnoDB writes a message to the log file indicating whether atomic memory access is used for mutexes, for mutexes and read/write locks, or neither. If suitable tools are used to build InnoDB and the target CPU supports the atomic operations required, InnoDB uses the built-in functions for mutexing. If, in addition, the compare-and-swap operation can be used on thread identifiers (`pthread_t`), then InnoDB uses the instructions for read-write locks as well.

Note: If you are building from source, ensure that the build process properly takes advantage of your platform capabilities.

For more information about the performance implications of locking, see [Section 8.10, “Optimizing Locking Operations”](#).

14.4.7.3. Using Operating System Memory Allocators

When InnoDB was developed, the memory allocators supplied with operating systems and run-time libraries were often lacking in performance and scalability. At that time, there were no memory allocator libraries tuned for multi-core CPUs. Therefore, InnoDB implemented its own memory allocator in the `mem` subsystem. This allocator is guarded by a single mutex, which may become a [bottleneck](#). InnoDB also implements a wrapper interface around the system allocator (`malloc` and `free`) that is likewise guarded by a single mutex.

Today, as multi-core systems have become more widely available, and as operating systems have matured, significant improvements have been made in the memory allocators provided with operating systems. New memory allocators perform better and are more scalable than they were in the past. The leading high-performance memory allocators include [Hoard](#), [libumem](#), [mtmalloc](#), [ptmalloc](#), [tbbmalloc](#), and [TCMalloc](#). Most workloads, especially those where memory is frequently allocated and released (such as multi-table joins), benefit from using a more highly tuned memory allocator as opposed to the internal, InnoDB-specific memory allocator.

You can control whether InnoDB uses its own memory allocator or an allocator of the operating system, by setting the value of the system configuration parameter `innodb_use_sys_malloc` [1575] in the MySQL option file (`my.cnf` or `my.ini`). If set to `ON` or `1` (the default), InnoDB uses the `malloc` and `free` functions of the underlying system rather than manage memory pools itself. This parameter is not dynamic, and takes effect only when the system is started. To continue to use the InnoDB memory allocator, set `innodb_use_sys_malloc` [1575] to `0`.



Note

When the InnoDB memory allocator is disabled, InnoDB ignores the value of the parameter `innodb_additional_mem_pool_size` [1548]. The InnoDB memory allocator uses an additional memory pool for satisfying allocation requests without having to fall back to the system memory allocator. When the InnoDB memory allocator is disabled, all such allocation requests are fulfilled by the system memory allocator.

On Unix-like systems that use dynamic linking, replacing the memory allocator may be as easy as making the environment variable `LD_PRELOAD` or `LD_LIBRARY_PATH` point to the dynamic library that implements the allocator. On other systems, some relinking may be necessary. Please refer to the documentation of the memory allocator library of your choice.

Since InnoDB cannot track all memory use when the system memory allocator is used (`innodb_use_sys_malloc` [1575] is ON), the section “BUFFER POOL AND MEMORY” in the output of the `SHOW ENGINE INNODB STATUS` command only includes the buffer pool statistics in the “Total memory allocated”. Any memory allocated using the `mem` subsystem or using `ut_malloc` is excluded.

For more information about the performance implications of InnoDB memory usage, see [Section 8.9, “Buffering and Caching”](#).

14.4.7.4. Controlling InnoDB Change Buffering

When `INSERT`, `UPDATE`, and `DELETE` operations are done to a table, often the values of indexed columns (particularly the values of secondary keys) are not in sorted order, requiring substantial I/O to bring secondary indexes up to date. InnoDB has an `insert buffer` that caches changes to secondary index entries when the relevant `page` is not in the `buffer pool`, thus avoiding I/O operations by not reading in the page from the disk. The buffered changes are merged when the page is loaded to the buffer pool, and the updated page is later flushed to disk using the normal mechanism. The InnoDB main thread merges buffered changes when the server is nearly idle, and during a `slow shutdown`.

Because it can result in fewer disk reads and writes, this feature is most valuable for workloads that are I/O-bound, for example applications with a high volume of DML operations such as bulk inserts.

However, the insert buffer occupies a part of the buffer pool, reducing the memory available to cache data pages. If the working set almost fits in the buffer pool, or if your tables have relatively few secondary indexes, it may be useful to disable insert buffering. If the working set entirely fits in the buffer pool, insert buffering does not impose any extra overhead, because it only applies to pages that are not in the buffer pool.

You can control the extent to which InnoDB performs insert buffering with the system configuration parameter `innodb_change_buffering` [1550]. You can turn on and off buffering for inserts, delete operations (when index records are initially marked for deletion) and purge operations (when index records are physically deleted). An update operation is represented as a combination of an insert and a delete. In MySQL 5.5 and higher, the default value is changed from `inserts` to `all`.

The allowed values of `innodb_change_buffering` [1550] are:

- `all`

The default value: buffer inserts, delete-marking operations, and purges.

- `none`

Do not buffer any operations.

- `inserts`

Buffer insert operations.

- `deletes`

Buffer delete-marking operations.

- `changes`

Buffer both inserts and delete-marking.

- `purges`

Buffer the physical deletion operations that happen in the background.

You can set the value of this parameter in the MySQL option file (`my.cnf` or `my.ini`) or change it dynamically with the `SET GLOBAL` command, which requires the `SUPER` privilege. Changing the setting affects the buffering of new operations; the merging of already buffered entries is not affected.

For more information about speeding up `INSERT`, `UPDATE`, and `DELETE` statements, see [Section 8.2.2, “Optimizing DML Statements”](#).

14.4.7.5. Controlling Adaptive Hash Indexing

If a table fits almost entirely in main memory, the fastest way to perform queries on it is to use [hash indexes](#) rather than [B-tree](#) lookups. MySQL monitors searches on each index defined for an InnoDB table. If it notices that certain index values are being accessed frequently, it automatically builds an in-memory hash table for that index. See [Section 14.3.11.4, “Adaptive Hash Indexes”](#) for background information and usage guidelines for the [adaptive hash index](#) feature and the `innodb_adaptive_hash_index` [1547] configuration option.

14.4.7.6. Changes Regarding Thread Concurrency

InnoDB uses operating system [threads](#) to process requests from user transactions. (Transactions may issue many requests to InnoDB before they commit or roll back.) On modern operating systems and servers with multi-core processors, where context switching is efficient, most workloads run well without any limit on the number of concurrent threads. Scalability improvements in MySQL 5.5 and up reduce the need to limit the number of concurrently executing threads inside InnoDB.

In situations where it is helpful to minimize context switching between threads, InnoDB can use a number of techniques to limit the number of concurrently executing operating system threads (and thus the number of requests that are processed at any one time). When InnoDB receives a new request from a user session, if the number of threads concurrently executing is at a pre-defined limit, the new request sleeps for a short time before it tries again. A request that cannot be rescheduled after the sleep is put in a first-in/first-out queue and eventually is processed. Threads waiting for locks are not counted in the number of concurrently executing threads.

You can limit the number of concurrent threads by setting the configuration parameter `innodb_thread_concurrency` [1573]. Once the number of executing threads reaches this limit, additional threads sleep for a number of microseconds, set by the configuration parameter `innodb_thread_sleep_delay` [1574], before being placed into the queue.

The default value for `innodb_thread_concurrency` [1573] and the implied default limit on the number of concurrent threads has been changed in various releases of MySQL and InnoDB. Currently, the default value of `innodb_thread_concurrency` [1573] is 0, so that by default there is no limit on the number of concurrently executing threads, as shown in [Table 14.7, “Changes to `innodb_thread_concurrency`”](#).

Table 14.7. Changes to `innodb_thread_concurrency`

InnoDB Version	MySQL Version	Default value	Default limit of concurrent threads	Value to allow unlimited threads
Built-in	Earlier than 5.1.11	20	No limit	20 or higher
Built-in	5.1.11 and newer	8	8	0
InnoDB before 1.0.3	(corresponding to Plugin)	8	8	0

InnoDB Version	MySQL Version	Default value	Default limit of concurrent threads	Value to allow unlimited threads
InnoDB 1.0.3 and newer	(corresponding to Plugin)	0	No limit	0

Note that InnoDB causes threads to sleep only when the number of concurrent threads is limited. When there is no limit on the number of threads, all contend equally to be scheduled. That is, if `innodb_thread_concurrency` [1573] is 0, the value of `innodb_thread_sleep_delay` [1574] is ignored.

When there is a limit on the number of threads, InnoDB reduces context switching overhead by permitting multiple requests made during the execution of a single SQL statement to enter InnoDB without observing the limit set by `innodb_thread_concurrency` [1573]. Since an SQL statement (such as a join) may comprise multiple row operations within InnoDB, InnoDB assigns “tickets” that allow a thread to be scheduled repeatedly with minimal overhead.

When a new SQL statement starts, a thread has no tickets, and it must observe `innodb_thread_concurrency` [1573]. Once the thread is entitled to enter InnoDB, it is assigned a number of tickets that it can use for subsequently entering InnoDB. If the tickets run out, `innodb_thread_concurrency` [1573] is observed again and further tickets are assigned. The number of tickets to assign is specified by the global option `innodb_concurrency_tickets` [1552], which is 500 by default. A thread that is waiting for a lock is given one ticket once the lock becomes available.

The correct values of these variables depend on your environment and workload. Try a range of different values to determine what value works for your applications. Before limiting the number of concurrently executing threads, review configuration options that may improve the performance of InnoDB on multi-core and multi-processor computers, such as `innodb_use_sys_malloc` and `innodb_adaptive_hash_index`.

For general performance information about MySQL thread handling, see [Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”](#).

14.4.7.7. Changes in the Read-Ahead Algorithm

A [read-ahead](#) request is an I/O request to prefetch multiple pages in the [buffer pool](#) asynchronously, in anticipation that these pages will be needed soon. InnoDB uses or has used two read-ahead algorithms to improve I/O performance:

Linear read-ahead is based on the access pattern of the pages in the buffer pool, not just their number. You can control when InnoDB performs a read-ahead operation by adjusting the number of sequential page accesses required to trigger an asynchronous read request, using the configuration parameter `innodb_read_ahead_threshold` [1568]. Before this parameter was added, InnoDB would only calculate whether to issue an asynchronous prefetch request for the entire next extent when it read in the last page of the current extent.

Random read-ahead is a former technique that has now been removed as of MySQL 5.5. If a certain number of pages from the same extent (64 consecutive pages) were found in the buffer pool, InnoDB asynchronously issued a request to prefetch the remaining pages of the extent. Random read-ahead added unnecessary complexity to the InnoDB code and often resulted in performance degradation rather than improvement. This feature is no longer part of InnoDB, and users should generally see equivalent or improved performance.

If the number of pages read from an extent of 64 pages is greater or equal to `innodb_read_ahead_threshold` [1568], InnoDB initiates an asynchronous read-ahead operation of the entire following extent. Thus, this parameter controls how sensitive InnoDB is to the pattern of page

accesses within an extent in deciding whether to read the following extent asynchronously. The higher the value, the more strict the access pattern check. For example, if you set the value to 48, InnoDB triggers a linear read-ahead request only when 48 pages in the current extent have been accessed sequentially. If the value is 8, InnoDB would trigger an asynchronous read-ahead even if as few as 8 pages in the extent were accessed sequentially.

The new configuration parameter `innodb_read_ahead_threshold` [1568] can be set to any value from 0-64. The default value is 56, meaning that an asynchronous read-ahead is performed only when 56 of the 64 pages in the extent are accessed sequentially. You can set the value of this parameter in the MySQL option file (`my.cnf` or `my.ini`), or change it dynamically with the `SET GLOBAL` command, which requires the `SUPER` privilege.

The `SHOW ENGINE INNODB STATUS` command displays statistics to help you evaluate the effectiveness of the read-ahead algorithm. See [Section 14.4.8.8, “More Read-Ahead Statistics”](#) for more information.

For more information about I/O performance, see [Section 8.5.7, “Optimizing InnoDB Disk I/O”](#) and [Section 8.11.3, “Optimizing Disk I/O”](#).

14.4.7.8. Multiple Background I/O Threads

InnoDB uses background [threads](#) to service various types of I/O requests. You can configure the number of background threads that service read and write I/O on data pages, using the configuration parameters `innodb_read_io_threads` [1568] and `innodb_write_io_threads` [1575]. These parameters signify the number of background threads used for read and write requests respectively. They are effective on all supported platforms. You can set the value of these parameters in the MySQL option file (`my.cnf` or `my.ini`); you cannot change them dynamically. The default value for these parameters is 4 and the permissible values range from 1-64.

These parameters replace `innodb_file_io_threads` from earlier versions of MySQL. If you try to set a value for this obsolete parameter, a warning is written to the log file and the value is ignored. This parameter only applied to Windows platforms. (On non-Windows platforms, there was only one thread each for read and write.)

The purpose of this change is to make InnoDB more scalable on high end systems. Each background thread can handle up to 256 pending I/O requests. A major source of background I/O is the [read-ahead](#) requests. InnoDB tries to balance the load of incoming requests in such way that most of the background threads share work equally. InnoDB also attempts to allocate read requests from the same extent to the same thread to increase the chances of coalescing the requests together. If you have a high end I/O subsystem and you see more than 64 × `innodb_read_io_threads` [1568] pending read requests in `SHOW ENGINE INNODB STATUS`, you might gain by increasing the value of `innodb_read_io_threads` [1568].

For more information about InnoDB I/O performance, see [Section 8.5.7, “Optimizing InnoDB Disk I/O”](#).

14.4.7.9. Asynchronous I/O on Linux

Starting in InnoDB 1.1 with MySQL 5.5, the [asynchronous I/O](#) capability that InnoDB has had on Windows systems is now available on Linux systems. (Other Unix-like systems continue to use synchronous I/O calls.) This feature improves the scalability of heavily I/O-bound systems, which typically show many pending reads/writes in the output of the command `SHOW ENGINE INNODB STATUS\G`.

Running with a large number of InnoDB I/O threads, and especially running multiple such instances on the same server machine, can exceed capacity limits on Linux systems. In this case, you can fix the error:

```
EAGAIN: The specified maxevents exceeds the user's limit of available events.
```

by writing a higher limit to `/proc/sys/fs/aio-max-nr`.

In general, if a problem with the asynchronous I/O subsystem in the OS prevents InnoDB from starting, set the option `innodb_use_native_aio=0` [1574] in the configuration file. This new configuration option applies to Linux systems only, and cannot be changed once the server is running.

For more information about InnoDB I/O performance, see [Section 8.5.7, “Optimizing InnoDB Disk I/O”](#).

14.4.7.10. Group Commit

InnoDB, like any other ACID-compliant database engine, flushes the redo log of a transaction before it is committed. Historically, InnoDB used group commit functionality to group multiple such flush requests together to avoid one flush for each commit. With group commit, InnoDB issues a single write to the log file to perform the commit action for multiple user transactions that commit at about the same time, significantly improving throughput.

Group commit in InnoDB worked until MySQL 4.x, and works once again with MySQL 5.1 with the InnoDB Plugin, and MySQL 5.5 and higher. The introduction of support for the distributed transactions and Two Phase Commit (2PC) in MySQL 5.0 interfered with the InnoDB group commit functionality. This issue is now resolved.

The group commit functionality inside InnoDB works with the Two Phase Commit protocol in MySQL. Re-enabling of the group commit functionality fully ensures that the ordering of commit in the MySQL binlog and the InnoDB logfile is the same as it was before. It means it is **totally safe to use MySQL Enterprise Backup with InnoDB 1.0.4** (that is, the InnoDB Plugin with MySQL 5.1) and above. When the binlog is enabled, you typically also set the configuration option `sync_binlog=0`, because group commit for the binary log is only supported if it is set to 0.

Group commit is transparent; you do not need to do anything to take advantage of this significant performance improvement.

For more information about performance of COMMIT and other transactional operations, see [Section 8.5.2, “Optimizing InnoDB Transaction Management”](#).

14.4.7.11. Controlling the Master Thread I/O Rate

The `master thread` in InnoDB is a thread that performs various tasks in the background. Most of these tasks are I/O related, such as flushing dirty pages from the buffer pool or writing changes from the insert buffer to the appropriate secondary indexes. The master thread attempts to perform these tasks in a way that does not adversely affect the normal working of the server. It tries to estimate the free I/O bandwidth available and tune its activities to take advantage of this free capacity. Historically, InnoDB has used a hard coded value of 100 IOPs (input/output operations per second) as the total I/O capacity of the server.

The parameter `innodb_io_capacity` [1559] indicates the overall I/O capacity available to InnoDB. This parameter should be set to approximately the number of I/O operations that the system can perform per second. The value depends on your system configuration. When `innodb_io_capacity` [1559] is set, the master threads estimates the I/O bandwidth available for background tasks based on the set value. Setting the value to 100 reverts to the old behavior.

You can set the value of `innodb_io_capacity` [1559] to any number 100 or greater. The default value is 200, reflecting that the performance of typical modern I/O devices is higher than in the early days of MySQL. Typically, values around the previous default of 100 are appropriate for consumer-level storage devices, such as hard drives up to 7200 RPMs. Faster hard drives, RAID configurations, and SSDs benefit from higher values.

You can set the value of this parameter in the MySQL option file (`my.cnf` or `my.ini`) or change it dynamically with the `SET GLOBAL` command, which requires the `SUPER` privilege.

For more information about InnoDB I/O performance, see [Section 8.5.7, “Optimizing InnoDB Disk I/O”](#).

14.4.7.12. Controlling the Flushing Rate of Dirty Pages

InnoDB performs certain tasks in the background, including [flushing of dirty pages](#) (those pages that have been changed but are not yet written to the database files) from the [buffer pool](#), a task performed by the [master thread](#). Currently, InnoDB aggressively flushes buffer pool pages if the percentage of dirty pages in the buffer pool exceeds [innodb_max_dirty_pages_pct \[1565\]](#).

InnoDB uses a new algorithm to estimate the required rate of flushing, based on the speed of redo log generation and the current rate of flushing. The intent is to smooth overall performance by ensuring that buffer flush activity keeps up with the need to keep the buffer pool “clean”. Automatically adjusting the rate of flushing can help to avoid steep dips in throughput, when excessive buffer pool flushing limits the I/O capacity available for ordinary read and write activity.

InnoDB uses its log files in a circular fashion. Before reusing a portion of a log file, InnoDB flushes to disk all dirty buffer pool pages whose redo entries are contained in that portion of the log file, a process known as a [sharp checkpoint](#). If a workload is write-intensive, it generates a lot of redo information, all written to the log file. If all available space in the log files is used up, a sharp checkpoint occurs, causing a temporary reduction in throughput. This situation can happen even though [innodb_max_dirty_pages_pct \[1565\]](#) is not reached.

InnoDB uses a heuristic-based algorithm to avoid such a scenario, by measuring the number of dirty pages in the buffer pool and the rate at which redo is being generated. Based on these numbers, InnoDB decides how many dirty pages to flush from the buffer pool each second. This self-adapting algorithm is able to deal with sudden changes in the workload.

Internal benchmarking has also shown that this algorithm not only maintains throughput over time, but can also improve overall throughput significantly.

Because adaptive flushing is a new feature that can significantly affect the I/O pattern of a workload, a new configuration parameter lets you turn off this feature. The default value of the boolean parameter [innodb_adaptive_flushing \[1547\]](#) is `TRUE`, enabling the new algorithm. You can set the value of this parameter in the MySQL option file (`my.cnf` or `my.ini`) or change it dynamically with the `SET GLOBAL` command, which requires the `SUPER` privilege.

For more information about InnoDB I/O performance, see [Section 8.5.7, “Optimizing InnoDB Disk I/O”](#).

14.4.7.13. Using the PAUSE Instruction in InnoDB Spin Loops

Synchronization inside InnoDB frequently involves the use of [spin loops](#): while waiting, InnoDB executes a tight loop of instructions repeatedly to avoid having the InnoDB [process](#) and [threads](#) be rescheduled by the operating system. If the spin loops are executed too quickly, system resources are wasted, imposing a performance penalty on transaction throughput. Most modern processors implement the `PAUSE` instruction for use in spin loops, so the processor can be more efficient.

InnoDB uses a `PAUSE` instruction in its spin loops on all platforms where such an instruction is available. This technique increases overall performance with CPU-bound workloads, and has the added benefit of minimizing power consumption during the execution of the spin loops.

You do not have to do anything to take advantage of this performance improvement.

For performance considerations for InnoDB locking operations, see [Section 8.10, “Optimizing Locking Operations”](#).

14.4.7.14. Control of Spin Lock Polling

Many InnoDB [mutexes](#) and [rw-locks](#) are reserved for a short time. On a multi-core system, it can be more efficient for a thread to continuously check if it can acquire a mutex or rw-lock for a while before sleeping. If the mutex or rw-lock becomes available during this polling period, the thread can continue immediately, in the same time slice. However, too-frequent polling by multiple threads of a shared object can cause “cache ping pong”, different processors invalidating portions of each others' cache. InnoDB minimizes this issue by waiting a random time between subsequent polls. The delay is implemented as a busy loop.

You can control the maximum delay between testing a mutex or rw-lock using the parameter [innodb_spin_wait_delay \[1570\]](#). The duration of the delay loop depends on the C compiler and the target processor. (In the 100MHz Pentium era, the unit of delay was one microsecond.) On a system where all processor cores share a fast cache memory, you might reduce the maximum delay or disable the busy loop altogether by setting `innodb_spin_wait_delay=0`. On a system with multiple processor chips, the effect of cache invalidation can be more significant and you might increase the maximum delay.

The default value of [innodb_spin_wait_delay \[1570\]](#) is 6. The spin wait delay is a dynamic global parameter that you can specify in the MySQL option file (`my.cnf` or `my.ini`) or change at runtime with the command `SET GLOBAL innodb_spin_wait_delay=delay`, where *delay* is the desired maximum delay. Changing the setting requires the `SUPER` privilege.

For performance considerations for InnoDB locking operations, see [Section 8.10, “Optimizing Locking Operations”](#).

14.4.7.15. Making Buffer Pool Scan Resistant

Rather than using a strictly [LRU](#) algorithm, InnoDB uses a technique to minimize the amount of data that is brought into the [buffer pool](#) and never accessed again. The goal is to make sure that frequently accessed (“hot”) pages remain in the buffer pool, even as [read-ahead](#) and [full table scans](#) bring in new blocks that might or might not be accessed afterward.

Newly read blocks are inserted into the middle of the list representing the buffer pool. of the LRU list. All newly read pages are inserted at a location that by default is $3/8$ from the tail of the LRU list. The pages are moved to the front of the list (the most-recently used end) when they are accessed in the buffer pool for the first time. Thus pages that are never accessed never make it to the front portion of the LRU list, and “age out” sooner than with a strict LRU approach. This arrangement divides the LRU list into two segments, where the pages downstream of the insertion point are considered “old” and are desirable victims for LRU eviction.

For an explanation of the inner workings of the InnoDB buffer pool and the specifics of its LRU replacement algorithm, see [Section 8.9.1, “The InnoDB Buffer Pool”](#).

You can control the insertion point in the LRU list, and choose whether InnoDB applies the same optimization to blocks brought into the buffer pool by table or index scans. The configuration parameter [innodb_old_blocks_pct \[1566\]](#) controls the percentage of “old” blocks in the LRU list. The default value of [innodb_old_blocks_pct \[1566\]](#) is 37, corresponding to the original fixed ratio of 3/8. The value range is 5 (new pages in the buffer pool age out very quickly) to 95 (only 5% of the buffer pool is reserved for hot pages, making the algorithm close to the familiar LRU strategy).

The optimization that keeps the buffer pool from being churned by read-ahead can avoid similar problems due to table or index scans. In these scans, a data page is typically accessed a few times in quick succession and is never touched again. The configuration parameter [innodb_old_blocks_time \[1566\]](#) specifies the time window (in milliseconds) after the first access to a page during which it can be accessed without being moved to the front (most-recently used end) of the LRU list. The default value of [innodb_old_blocks_time \[1566\]](#) is 0, corresponding to the original behavior of moving a page to the most-recently used end of the buffer pool list when it is first accessed in the buffer pool. Increasing this value makes more and more blocks likely to age out faster from the buffer pool.

Both the new parameters `innodb_old_blocks_pct` [1566] and `innodb_old_blocks_time` [1566] are dynamic, global and can be specified in the MySQL option file (`my.cnf` or `my.ini`) or changed at runtime with the `SET GLOBAL` command. Changing the setting requires the `SUPER` privilege.

To help you gauge the effect of setting these parameters, the `SHOW ENGINE INNODB STATUS` command reports additional statistics. The `BUFFER POOL AND MEMORY` section now looks like:

```
Total memory allocated 1107296256; in additional pool allocated 0
Dictionary memory allocated 80360
Buffer pool size      65535
Free buffers          0
Database pages       63920
Old database pages   23600
Modified db pages    34969
Pending reads        32
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 414946, not young 2930673
1274.75 young/s, 16521.90 non-young/s
Pages read 486005, created 3178, written 160585
2132.37 reads/s, 3.40 creates/s, 323.74 writes/s
Buffer pool hit rate 950 / 1000, young-making rate 30 / 1000 not 392 / 1000
Pages read ahead 1510.10/s, evicted without access 0.00/s
LRU len: 63920, unzip_LRU len: 0
I/O sum[43690]:cur[221], unzip sum[0]:cur[0]
```

- `Old database pages` is the number of pages in the “old” segment of the LRU list.
- `Pages made young` and `not young` is the total number of “old” pages that have been made young or not respectively.
- `young/s` and `non-young/s` is the rate at which page accesses to the “old” pages have resulted in making such pages young or otherwise respectively since the last invocation of the command.
- `young-making rate` and `not` provides the same rate but in terms of overall buffer pool accesses instead of accesses just to the “old” pages.

Because the effects of these parameters can vary widely based on your hardware configuration, your data, and the details of your workload, always benchmark to verify the effectiveness before changing these settings in any performance-critical or production environment.

In mixed workloads where most of the activity is OLTP type with periodic batch reporting queries which result in large scans, setting the value of `innodb_old_blocks_time` [1566] during the batch runs can help keep the working set of the normal workload in the buffer pool.

When scanning large tables that cannot fit entirely in the buffer pool, setting `innodb_old_blocks_pct` [1566] to a small value keeps the data that is only read once from consuming a significant portion of the buffer pool. For example, setting `innodb_old_blocks_pct=5` restricts this data that is only read once to 5% of the buffer pool.

When scanning small tables that do fit into memory, there is less overhead for moving pages around within the buffer pool, so you can leave `innodb_old_blocks_pct` [1566] at its default value, or even higher, such as `innodb_old_blocks_pct=50`.

The effect of the `innodb_old_blocks_time` [1566] parameter is harder to predict than the `innodb_old_blocks_pct` [1566] parameter, is relatively small, and varies more with the workload. To arrive at an optimal value, conduct your own benchmarks if the performance improvement from adjusting `innodb_old_blocks_pct` [1566] is not sufficient.

For more information about the InnoDB buffer pool, see [Section 8.9.1, “The InnoDB Buffer Pool”](#).

14.4.7.16. Improvements to Crash Recovery Performance

A number of optimizations speed up certain steps of the [recovery](#) that happens on the next startup after a crash. In particular, scanning the [redo log](#) and applying the redo log are faster than in MySQL 5.1 and earlier, due to improved algorithms for memory management. You do not need to take any actions to take advantage of this performance enhancement. If you kept the size of your redo log files artificially low because recovery took a long time, you can consider increasing the file size.

For more information about InnoDB recovery, see [Section 14.3.7.1, “The InnoDB Recovery Process”](#).

14.4.7.17. Integration with MySQL PERFORMANCE_SCHEMA

Starting with InnoDB 1.1 with MySQL 5.5, you can profile certain internal InnoDB operations using the MySQL [Performance Schema feature](#). This type of tuning is primarily for expert users, those who push the limits of MySQL performance, read the MySQL source code, and evaluate optimization strategies to overcome performance bottlenecks. DBAs can also use this feature for capacity planning, to see whether their typical workload encounters any performance bottlenecks with a particular combination of CPU, RAM, and disk storage; and if so, to judge whether performance can be improved by increasing the capacity of some part of the system.

To use this feature to examine InnoDB performance:

- You must be running MySQL 5.5 or higher. You must build the database server from source, enabling the Performance Schema feature by building with the `--with-perfschema` option. Since the Performance Schema feature introduces some performance overhead, you should use it on a test or development system rather than on a production system.
- You must be running InnoDB 1.1 or higher.
- You must be generally familiar with how to use the [Performance Schema feature](#), for example to query tables in the `performance_schema` database.
- Examine the following kinds of InnoDB objects by querying the appropriate `performance_schema` tables. The items associated with InnoDB all contain the substring `innodb` in the `NAME` column.

For the definitions of the `*_instances` tables, see [Section 21.7.2, “Performance Schema Instance Tables”](#). For the definitions of the `*_summary_*` tables, see [Section 21.7.4, “Performance Schema Summary Tables”](#). For the definition of the `thread` table, see [Section 21.7.5, “Performance Schema Miscellaneous Tables”](#). For the definition of the `*_current_*` and `*_history_*` tables, see [Section 21.7.3, “Performance Schema Wait Event Tables”](#).

- [Mutexes](#) in the `mutex_instances` table. (Mutexes and RW-locks related to the InnoDB buffer pool are not included in this coverage; the same applies to the output of the `SHOW ENGINE INNODB MUTEX` command.)
- [RW-locks](#) in the `rwlock_instances` table.
- RW-locks in the `rwlock_instances` table.
- File I/O operations in the `file_instances`, `file_summary_by_event_name`, and `file_summary_by_instance` tables.
- [Threads](#) in the `PROCESSLIST` table.
- During performance testing, examine the performance data in the `events_waits_current` and `events_waits_history_long` tables. If you are interested especially in InnoDB-related objects,

use the clause `where name like "%innodb%"` to see just those entries; otherwise, examine the performance statistics for the overall MySQL server.

- You must be running MySQL 5.5, with the Performance Schema enabled by building with the `--with-perfschema` build option.

For more information about the MySQL Performance Schema, see [Chapter 21, MySQL Performance Schema](#).

14.4.7.18. Improvements to Performance from Multiple Buffer Pools

This performance enhancement is primarily useful for people with a large [buffer pool](#) size, typically in the multi-gigabyte range. To take advantage of this speedup, you must set the new [innodb_buffer_pool_instances \[1549\]](#) configuration option, and you might also adjust the [innodb_buffer_pool_size \[1549\]](#) value.

When the InnoDB buffer pool is large, many data requests can be satisfied by retrieving from memory. You might encounter bottlenecks from multiple threads trying to access the buffer pool at once. Starting in InnoDB 1.1 and MySQL 5.5, you can enable multiple buffer pools to minimize this contention. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pools randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool, and is protected by its own buffer pool mutex.

To enable this feature, set the [innodb_buffer_pool_instances](#) configuration option to a value greater than 1 (the default) up to 64 (the maximum). This option takes effect only when you set the [innodb_buffer_pool_size](#) to a size of 1 gigabyte or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of [innodb_buffer_pool_instances \[1549\]](#) and [innodb_buffer_pool_size \[1549\]](#) so that each buffer pool instance is at least 1 gigabyte.

For more information about the InnoDB buffer pool, see [Section 8.9.1, “The InnoDB Buffer Pool”](#).

14.4.7.19. Better Scalability with Multiple Rollback Segments

Starting in InnoDB 1.1 with MySQL 5.5, the limit on concurrent [transactions](#) is greatly expanded, removing a bottleneck with the InnoDB [rollback segment](#) that affected high-capacity systems. The limit applies to concurrent transactions that change any data; read-only transactions do not count against that maximum.

The single rollback segment is now divided into 128 segments, each of which can support up to 1023 transactions that perform writes, for a total of approximately 128K concurrent transactions. The original transaction limit was 1023.

Each transaction is assigned to one of the rollback segments, and remains tied to that rollback segment for the duration. This enhancement improves both scalability (higher number of concurrent transactions) and performance (less contention when different transactions access the rollback segments).

To take advantage of this feature, you do not need to create any new database or tables, or reconfigure anything. You must do a [slow shutdown](#) before upgrading from MySQL 5.1 or earlier, or some time afterward. InnoDB makes the required changes inside the [system tablespace](#) automatically, the first time you restart after performing a slow shutdown.

For more information about performance of InnoDB under high transactional load, see [Section 8.5.2, “Optimizing InnoDB Transaction Management”](#).

14.4.7.20. Better Scalability with Improved Purge Scheduling

Starting in InnoDB 1.1 with MySQL 5.5, the [purge](#) operations (a type of garbage collection) that InnoDB performs automatically can be done in a separate thread, rather than as part of the [master thread](#). This change improves scalability, because the main database operations run independently from maintenance work happening in the background.

To enable this feature, set the configuration option `innodb_purge_threads=1`, as opposed to the default of 0, which combines the purge operation into the master thread.

You might not notice a significant speedup, because the purge thread might encounter new types of contention; the single purge thread really lays the groundwork for further tuning and possibly multiple purge threads in the future. There is another new configuration option, `innodb_purge_batch_size` with a default of 20 and maximum of 5000. This option is mainly intended for experimentation and tuning of purge operations, and should not be interesting to typical users.

For more information about InnoDB I/O performance, see [Section 8.5.7, “Optimizing InnoDB Disk I/O”](#).

14.4.7.21. Improved Log Sys Mutex

This is another performance improvement that comes for free, with no user action or configuration needed. The details here are intended for performance experts who delve into the InnoDB source code, or interpret reports with keywords such as “mutex” and “log_sys”.

The [mutex](#) known as the log sys mutex has historically done double duty, controlling access to internal data structures related to log records and the [LSN](#), as well as pages in the [buffer pool](#) that are changed when a [mini-transaction](#) is committed. Starting in InnoDB 1.1 with MySQL 5.5, these two kinds of operations are protected by separate mutexes, with a new `log_buf` mutex controlling writes to buffer pool pages due to mini-transactions.

For performance considerations for InnoDB locking operations, see [Section 8.10, “Optimizing Locking Operations”](#).

14.4.7.22. Separate Flush List Mutex

Starting with InnoDB 1.1 with MySQL 5.5, concurrent access to the [buffer pool](#) is faster. Operations involving the [flush list](#), a data structure related to the buffer pool, are now controlled by a separate [mutex](#) and do not block access to the buffer pool. You do not need to configure anything to take advantage of this speedup; it is fully automatic.

For more information about the InnoDB buffer pool, see [Section 8.9.1, “The InnoDB Buffer Pool”](#).

14.4.8. Changes for Flexibility, Ease of Use and Reliability

This chapter describes several recently added InnoDB features that offer new flexibility and improve ease of use, reliability and performance. The [Barracuda](#) file format improves efficiency for storing large variable-length columns, and enables table compression. Configuration options that once were unchangeable after startup, are now flexible and can be changed dynamically. Some improvements are automatic, such as faster and more efficient `TRUNCATE TABLE`. Others allow you the flexibility to control InnoDB behavior; for example, you can control whether certain problems cause errors or just warnings. And informational messages and error reporting continue to be made more user-friendly.

14.4.8.1. The Barracuda File Format

InnoDB has started using named file formats to improve compatibility in upgrade and downgrade situations, or heterogeneous systems running different levels of MySQL. Many important InnoDB features, such as table compression and the `DYNAMIC` row format for more efficient BLOB storage, require creating tables in the [Barracuda](#) file format. The original file format, which previously didn't have a name, is known now as [Antelope](#).

To create new tables that take advantage of the Barracuda features, enable that file format using the configuration parameter `innodb_file_format` [1554]. The value of this parameter determines whether a newly created table or index can use compression or the new `DYNAMIC` row format.

To preclude the use of new features that would make your database inaccessible to the built-in InnoDB in MySQL 5.1 and prior releases, omit `innodb_file_format` [1554] or set it to `Antelope`.

You can set the value of `innodb_file_format` [1554] on the command line when you start `mysqld`, or in the option file `my.cnf` (Unix operating systems) or `my.ini` (Windows). You can also change it dynamically with the `SET GLOBAL` statement.

For more information about managing file formats, see [Section 14.4.4, “InnoDB File-Format Management”](#).

14.4.8.2. Dynamic Control of System Configuration Parameters

In MySQL 5.5 and higher, you can change certain system configuration parameters without shutting down and restarting the server, as was necessary in MySQL 5.1 and lower. This increases uptime, and makes it easier to test and prototype new SQL and application code. The following sections explain these parameters.

14.4.8.2.1. Dynamically Changing `innodb_file_per_table`

Since MySQL version 4.1, InnoDB has provided two alternatives for how tables are stored on disk. You can create a new table and its indexes in the shared `system tablespace`, physically stored in the `ibdata` files. Or, you can store a new table and its indexes in a separate tablespace (a `.ibd` file). The storage layout for each InnoDB table is determined by the configuration parameter `innodb_file_per_table` [1555] at the time the table is created.

In MySQL 5.5 and higher, the configuration parameter `innodb_file_per_table` [1555] is dynamic, and can be set `ON` or `OFF` using the `SET GLOBAL`. Previously, the only way to set this parameter was in the MySQL option file (`my.cnf` or `my.ini`), and changing it required shutting down and restarting the server.

The default setting is `OFF`, so new tables and indexes are created in the system tablespace. Dynamically changing the value of this parameter requires the `SUPER` privilege and immediately affects the operation of all connections.

Tables created when `innodb_file_per_table` [1555] is enabled can use the `Barracuda` file format, and `TRUNCATE` returns the disk space for those tables to the operating system. The Barracuda file format in turn enables features such as table compression and the `DYNAMIC` row format. Tables created when `innodb_file_per_table` [1555] is off cannot use these features. To take advantage of those features for an existing table, you can turn on the file-per-table setting and run `ALTER TABLE t ENGINE=INNODB` for that table.

When you redefine the primary key for an InnoDB table, the table is re-created using the current settings for `innodb_file_per_table` [1555] and `innodb_file_format` [1554]. This behavior does not apply when adding or dropping InnoDB secondary indexes, as explained in [Section 14.4.2, “Fast Index Creation in the InnoDB Storage Engine”](#). When a secondary index is created without rebuilding the table, the index is stored in the same file as the table data, regardless of the current `innodb_file_per_table` [1555] setting.

14.4.8.2.2. Dynamically Changing `innodb_stats_on_metadata`

In MySQL 5.5 and higher, you can change the setting of `innodb_stats_on_metadata` [1571] dynamically at runtime, to control whether or not InnoDB performs statistics gathering when metadata statements are executed. To change the setting, issue the statement `SET GLOBAL innodb_stats_on_metadata=mode`, where `mode` is either `ON` or `OFF` (or `1` or `0`). Changing this setting requires the `SUPER` privilege and immediately affects the operation of all connections.

This setting is related to the feature described in [Section 14.4.8.5, “Controlling Optimizer Statistics Estimation”](#).

14.4.8.2.3. Dynamically Changing `innodb_lock_wait_timeout`

The length of time a transaction waits for a resource, before giving up and rolling back the statement, is determined by the value of the configuration parameter `innodb_lock_wait_timeout` [1560]. (In MySQL 5.0.12 and earlier, the entire transaction was rolled back, not just the statement.) Your application can try the statement again (usually after waiting for a while), or roll back the entire transaction and restart.

The error returned when the timeout period is exceeded is:

```
ERROR HY000: Lock wait timeout exceeded; try restarting transaction
```

In MySQL 5.5 and higher, the configuration parameter `innodb_lock_wait_timeout` [1560] can be set at runtime with the `SET GLOBAL` or `SET SESSION` statement. Changing the `GLOBAL` setting requires the `SUPER` privilege and affects the operation of all clients that subsequently connect. Any client can change the `SESSION` setting for `innodb_lock_wait_timeout` [1560], which affects only that client.

In MySQL 5.1 and earlier, the only way to set this parameter was in the MySQL option file (`my.cnf` or `my.ini`), and changing it required shutting down and restarting the server.

14.4.8.2.4. Dynamically Changing `innodb_adaptive_hash_index`

As described in [Section 14.4.7.5, “Controlling Adaptive Hash Indexing”](#), it may be desirable, depending on your workload, to dynamically enable or disable the `adaptive hash indexing` scheme InnoDB uses to improve query performance.

The start-up option `innodb_adaptive_hash_index` [1547] allows the adaptive hash index to be disabled. It is enabled by default. You can modify this parameter through the `SET GLOBAL` statement, without restarting the server. Changing the setting requires the `SUPER` privilege.

Disabling the adaptive hash index empties the hash table immediately. Normal operations can continue while the hash table is emptied, and executing queries that were using the hash table access the index B-trees directly instead. When the adaptive hash index is re-enabled, the hash table is populated again during normal operation.

14.4.8.3. `TRUNCATE TABLE` Reclaims Space

When you `truncate` a table that is stored in a `.ibd` file of its own (because `innodb_file_per_table` [1555] was enabled when the table was created), and if the table is not referenced in a `FOREIGN KEY` constraint, the table is dropped and re-created in a new `.ibd` file. This operation is much faster than deleting the rows one by one. The operating system can reuse the disk space, in contrast to tables within the InnoDB `system tablespace`, where only InnoDB can use the space after they are truncated. `Physical backups` can also be smaller, without big blocks of unused space in the middle of the system tablespace.

Previous versions of InnoDB would re-use the existing `.ibd` file, thus releasing the space only to InnoDB for storage management, but not to the operating system. Note that when the table is truncated, the count of rows affected by the `TRUNCATE TABLE` statement is an arbitrary number.



Note

If there is a referential constraint between two columns in the same table, that table can still be truncated using this fast technique.

If there are referential constraints between the table being truncated and other tables, the truncate operation fails. This is a change to the previous behavior, which

would transform the `TRUNCATE` operation to a `DELETE` operation that removed all the rows and triggered `ON DELETE` operations on `child tables`.

14.4.8.4. InnoDB Strict Mode

To guard against ignored typos and syntax errors in SQL, or other unintended consequences of various combinations of operational modes and SQL statements, InnoDB provides a `strict mode` of operations. In this mode, InnoDB raises error conditions in certain cases, rather than issuing a warning and processing the specified statement (perhaps with unintended behavior). This is analogous to `sql_mode` in MySQL, which controls what SQL syntax MySQL accepts, and determines whether it silently ignores errors, or validates input syntax and data values. Since strict mode is relatively new, some statements that execute without errors with earlier versions of MySQL might generate errors unless you disable strict mode.

The setting of InnoDB strict mode affects the handling of syntax errors on the `CREATE TABLE`, `ALTER TABLE` and `CREATE INDEX` statements. The strict mode also enables a record size check, so that an `INSERT` or `UPDATE` never fails due to the record being too large for the selected page size.

We recommend running in strict mode when using the `ROW_FORMAT` and `KEY_BLOCK_SIZE` clauses on `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements. Without strict mode, InnoDB ignores conflicting clauses and creates the table or index, with only a warning in the message log. The resulting table might have different behavior than you intended, such as having no compression when you tried to create a compressed table. When InnoDB strict mode is on, such problems generate an immediate error and the table or index is not created, avoiding a troubleshooting session later.

InnoDB strict mode is set with the configuration parameter `innodb_strict_mode` [1572], which can be specified as `on` or `off`. You can set the value on the command line when you start `mysqld`, or in the configuration file `my.cnf` or `my.ini`. You can also enable or disable InnoDB strict mode at run time with the statement `SET [GLOBAL|SESSION] innodb_strict_mode=mode`, where `mode` is either `ON` or `OFF`. Changing the `GLOBAL` setting requires the `SUPER` privilege and affects the operation of all clients that subsequently connect. Any client can change the `SESSION` setting for `innodb_strict_mode` [1572], and the setting affects only that client.

14.4.8.5. Controlling Optimizer Statistics Estimation

The MySQL query optimizer uses estimated statistics about key distributions to choose the indexes for an execution plan, based on the relative `selectivity` of the index. Certain operations cause InnoDB to sample random pages from each index on a table to estimate the `cardinality` of the index. (This technique is known as `random dives`.) These operations include the `ANALYZE TABLE` statement, the `SHOW TABLE STATUS` statement, and accessing the table for the first time after a restart.

To give you control over the quality of the statistics estimate (and thus better information for the query optimizer), you can now change the number of sampled pages using the parameter `innodb_stats_sample_pages` [1571]. Previously, the number of sampled pages was always 8, which could be insufficient to produce an accurate estimate, leading to poor index choices by the query optimizer. This technique is especially important for large tables and tables used in `joins`. Unnecessary `full table scans` for such tables can be a substantial performance issue.

You can set the global parameter `innodb_stats_sample_pages` [1571], at run time. The default value for this parameter is 8, preserving the same behavior as in past releases.



Note

The value of `innodb_stats_sample_pages` [1571] affects the index sampling for *all* tables and indexes. There are the following potentially significant impacts when you change the index sample size:

- Small values like 1 or 2 can result in very inaccurate estimates of cardinality.
- Increasing the `innodb_stats_sample_pages` [1571] value might require more disk reads. Values much larger than 8 (say, 100), can cause a big slowdown in the time it takes to open a table or execute `SHOW TABLE STATUS`.
- The optimizer might choose very different query plans based on different estimates of index selectivity.

To disable the cardinality estimation for metadata statements such as `SHOW TABLE STATUS`, execute the statement `SET GLOBAL innodb_stats_on_metadata=OFF` (or 0). The ability to set this option dynamically is also relatively new.

All InnoDB tables are opened, and the statistics are re-estimated for all associated indexes, when the `mysql` client starts if the auto-rehash setting is set on (the default). To improve the start up time of the `mysql` client, you can turn auto-rehash off. The auto-rehash feature enables automatic name completion of database, table, and column names for interactive users.

Whatever value of `innodb_stats_sample_pages` [1571] works best for a system, set the option and leave it at that value. Choose a value that results in reasonably accurate estimates for all tables in your database without requiring excessive I/O. Because the statistics are automatically recalculated at various times other than on execution of `ANALYZE TABLE`, it does not make sense to increase the index sample size, run `ANALYZE TABLE`, then decrease sample size again. The more accurate statistics calculated by `ANALYZE` running with a high value of `innodb_stats_sample_pages` [1571] can be wiped away later.

Although it is not possible to specify the sample size on a per-table basis, smaller tables generally require fewer index samples than larger tables do. If your database has many large tables, consider using a higher value for `innodb_stats_sample_pages` [1571] than if you have mostly smaller tables.

14.4.8.6. Better Error Handling when Dropping Indexes

For optimal performance with DML statements, InnoDB requires an index to exist on `foreign key` columns, so that `UPDATE` and `DELETE` operations on a `parent table` can easily check whether corresponding rows exist in the `child table`. MySQL creates or drops such indexes automatically when needed, as a side-effect of `CREATE TABLE`, `CREATE INDEX`, and `ALTER TABLE` statements.

When you drop an index, InnoDB checks whether the index is not used for checking a foreign key constraint. It is still OK to drop the index if there is another index that can be used to enforce the same constraint. InnoDB prevents you from dropping the last index that can enforce a particular referential constraint.

The message that reports this error condition is:

```
ERROR 1553 (HY000): Cannot drop index 'fooIdx':
needed in a foreign key constraint
```

This message is friendlier than the earlier message it replaces:

```
ERROR 1025 (HY000): Error on rename of './db2/#sql-18eb_3'
to './db2/foo'(errno: 150)
```

A similar change in error reporting applies to an attempt to drop the primary key index. For tables without an explicit `PRIMARY KEY`, InnoDB creates an implicit `clustered index` using the first columns of the table that are declared `UNIQUE` and `NOT NULL`. When you drop such an index, InnoDB automatically copies the table and rebuilds the index using a different `UNIQUE NOT NULL` group of columns or a system-generated

key. Since this operation changes the primary key, it uses the slow method of copying the table and re-creating the index, rather than the Fast Index Creation technique from [Section 14.4.2.3, “Implementation Details of Fast Index Creation”](#).

Previously, an attempt to drop an implicit clustered index (the first `UNIQUE NOT NULL` index) failed if the table did not contain a `PRIMARY KEY`:

```
ERROR 42000: This table type requires a primary key
```

14.4.8.7. More Compact Output of `SHOW ENGINE INNODB MUTEX`

The statement `SHOW ENGINE INNODB MUTEX` displays information about InnoDB [mutexes](#) and [rw-locks](#). Although this information is useful for tuning on multi-core systems, the amount of output can be overwhelming on systems with a big [buffer pool](#). There is one mutex and one rw-lock in each 16K buffer pool block, and there are 65,536 blocks per gigabyte. It is unlikely that a single block mutex or rw-lock from the buffer pool could become a performance bottleneck.

`SHOW ENGINE INNODB MUTEX` now skips the mutexes and rw-locks of buffer pool blocks. It also does not list any mutexes or rw-locks that have never been waited on (`os_waits=0`). Thus, `SHOW ENGINE INNODB MUTEX` only displays information about mutexes and rw-locks outside of the buffer pool that have caused at least one OS-level [wait](#).

14.4.8.8. More Read-Ahead Statistics

As described in [Section 14.4.7.7, “Changes in the Read-Ahead Algorithm”](#), a read-ahead request is an asynchronous I/O request issued in anticipation that a page will be used in the near future. Knowing how many pages are read through this read-ahead mechanism, and how many of them are evicted from the buffer pool without ever being accessed, can be useful to help fine-tune the parameter [innodb_read_ahead_threshold \[1568\]](#).

`SHOW ENGINE INNODB STATUS` output displays the global status variables [Innodb_buffer_pool_read_ahead](#) and [Innodb_buffer_pool_read_ahead_evicted](#). These variables indicate the number of pages brought into the [buffer pool](#) by read-ahead requests, and the number of such pages [evicted](#) from the buffer pool without ever being accessed respectively. These counters provide global values since the last server restart.

`SHOW ENGINE INNODB INNODB STATUS` also shows the rate at which the read-ahead pages are read in and the rate at which such pages are evicted without being accessed. The per-second averages are based on the statistics collected since the last invocation of `SHOW ENGINE INNODB INNODB STATUS` and are displayed in the [BUFFER POOL AND MEMORY](#) section of the output.

Since the InnoDB read-ahead mechanism has been simplified to remove random read-ahead, the status variables [Innodb_buffer_pool_read_ahead_rnd](#) and [Innodb_buffer_pool_read_ahead_seq](#) are no longer part of the `SHOW ENGINE INNODB STATUS` output.

14.4.9. Installing the InnoDB Storage Engine

When you use the InnoDB storage engine 1.1 and above, with MySQL 5.5 and above, you do not need to do anything special to install: everything comes configured as part of the MySQL source and binary distributions. This is a change from earlier releases of the InnoDB Plugin, where you were required to match up MySQL and InnoDB version numbers and update your build and configuration processes.

The InnoDB storage engine is included in the MySQL distribution, starting from MySQL 5.1.38. From MySQL 5.1.46 and up, this is the only download location for the InnoDB storage engine; it is not available from the InnoDB web site.

If you used any scripts or configuration files with the earlier InnoDB storage engine from the InnoDB web site, be aware that the filename of the shared library as supplied by MySQL is `ha_innodb_plugin.so` or `ha_innodb_plugin.dll`, as opposed to `ha_innodb.so` or `ha_innodb.dll` in the older Plugin downloaded from the InnoDB web site. You might need to change the applicable file names in your startup or configuration scripts.

Because the InnoDB storage engine has now replaced the built-in InnoDB, you no longer need to specify options like `--ignore-builtin-innodb` and `--plugin-load` during startup.

To take best advantage of current InnoDB features, we recommend specifying the following options in your configuration file:

```
innodb_file_per_table=1
innodb_file_format=barracuda
innodb_strict_mode=1
```

For information about these new features, see [Section 14.4.8.2.1, “Dynamically Changing `innodb_file_per_table`”](#), [Section 14.4.4, “InnoDB File-Format Management”](#), and [Section 14.4.8.4, “InnoDB Strict Mode”](#). You might need to continue to use the previous values for these parameters in some replication and similar configurations involving both new and older versions of MySQL.

14.4.10. Upgrading the InnoDB Storage Engine

Prior to MySQL 5.5, some upgrade scenarios involved upgrading the separate instance of InnoDB known as the InnoDB Plugin. In MySQL 5.5 and higher, the features of the InnoDB Plugin have been folded back into built-in InnoDB, so the upgrade procedure for InnoDB is the same as the one for the MySQL server. For details, see [Section 2.12.1, “Upgrading MySQL”](#).

14.4.11. Downgrading the InnoDB Storage Engine

14.4.11.1. Overview

Prior to MySQL 5.5, some downgrade scenarios involved switching the separate instance of InnoDB known as the InnoDB Plugin back to the built-in InnoDB storage engine. In MySQL 5.5 and higher, the features of the InnoDB Plugin have been folded back into built-in InnoDB, so the downgrade procedure for InnoDB is the same as the one for the MySQL server. For details, see [Section 2.12.2, “Downgrading MySQL”](#).

14.4.12. InnoDB Storage Engine Change History

14.4.12.1. Changes in InnoDB Storage Engine 1.x

Since InnoDB 1.1 is tightly integrated with MySQL 5.5, for changes after the initial InnoDB 1.1 release, see the [MySQL 5.5 Release Notes](#).

14.4.12.2. Changes in InnoDB Storage Engine 1.1 (April 13, 2010)

For an overview of the changes, see [this introduction article for MySQL 5.5 with InnoDB 1.1](#). The following is a condensed version of the change log.

Fix for Bug #52580: Crash in `ha_innobase::open` on executing INSERT with concurrent ALTER TABLE.

Change in MySQL Bug #51557 releases the mutex `LOCK_open` before `ha_innobase::open()`, causing racing condition for index translation table creation. Fix it by adding `dict_sys` mutex for the operation.

Add support for multiple buffer pools.

Fix Bug #26590: MySQL does not allow more than 1023 open transactions. Create additional rollback segments on startup. Reduce the upper limit of total rollback segments from 256 to 128. This is because we can't use the sign bit. It has not caused problems in the past because we only created one segment. InnoDB has always had the capability to use the additional rollback segments, therefore this patch is backward compatible. The only requirement to maintain backward compatibility has been to ensure that the additional segments are created after the double write buffer. This is to avoid breaking assumptions in the existing code.

Implement Performance Schema in InnoDB. Objects in four different modules in InnoDB have been performance instrumented, these modules are: mutexes, rwlocks, file I/O, and threads. We mostly preserved the existing APIs, but APIs would point to instrumented function wrappers if performance schema is defined. There are 4 different defines that control the instrumentation of each module. The feature is off by default, and will be compiled in with a special build option, and require a configure option to turn it on when the server boots.

Implement the `buf_pool_watch` for DeleteBuffering in the page hash table. This serves two purposes. It allows multiple watches to be set at the same time (by multiple purge threads) and it removes a race condition when the read of a block completes about the time the buffer pool watch is being set.

Introduce a new mutex to protect `flush_list`. Redesign `mtr_commit()` in a way that `log_sys` mutex is not held while all `mtr_memos` are popped and is released just after the modified blocks are inserted into the `flush_list`. This should reduce contention on `log_sys` mutex.

Implement the global variable `innodb_change_buffering` [1550], with the following values:

- `none`: buffer nothing
- `inserts`: buffer inserts (like InnoDB so far)
- `deletes`: buffer delete-marks
- `changes`: buffer inserts and delete-marks
- `purges`: buffer delete-marks and deletes
- `all`: buffer all operations (insert, delete-mark, delete)

The default is `all`. All values except `none` and `inserts` will make InnoDB write new-format records to the insert buffer, even for inserts.

Provide support for native AIO on Linux.

14.4.12.3. Changes in InnoDB Plugin 1.0.x

The InnoDB 1.0.x releases that accompany MySQL 5.1 have their own change history. Changes up to InnoDB 1.0.8 are listed at <http://dev.mysql.com/doc/innodb-plugin/1.0/en/innodb-changes.html>. Changes from InnoDB 1.0.9 and up are incorporated in the [MySQL 5.1 Release Notes](#).

14.4.13. Third-Party Software

Oracle acknowledges that certain Third Party and Open Source software has been used to develop or is incorporated in the InnoDB storage engine. This appendix includes required third-party license information.

14.4.13.1. Performance Patches from Google

Oracle gratefully acknowledges the following contributions from Google, Inc. to improve InnoDB performance:

- Replacing InnoDB's use of Pthreads mutexes with calls to GCC atomic builtins, as discussed in [Section 14.4.7.2, "Faster Locking for Improved Scalability"](#). This change means that InnoDB mutex and rw-lock operations take less CPU time, and improves throughput on those platforms where the atomic operations are available.
- Controlling master thread I/O rate, as discussed in [Section 14.4.7.11, "Controlling the Master Thread I/O Rate"](#). The master thread in InnoDB is a thread that performs various tasks in the background. Historically, InnoDB has used a hard coded value as the total I/O capacity of the server. With this change, user can control the number of I/O operations that can be performed per second based on their own workload.

Changes from the Google contributions were incorporated in the following source code files: `btr0cur.c`, `btr0sea.c`, `buf0buf.c`, `buf0buf.ic`, `ha_innodb.cc`, `log0log.c`, `log0log.h`, `os0sync.h`, `row0sel.c`, `srv0srv.c`, `srv0srv.h`, `srv0start.c`, `sync0arr.c`, `sync0rw.c`, `sync0rw.h`, `sync0rw.ic`, `sync0sync.c`, `sync0sync.h`, `sync0sync.ic`, and `univ.i`.

These contributions are incorporated subject to the conditions contained in the file `COPYING.Google`, which are reproduced here.

```
Copyright (c) 2008, 2009, Google Inc.
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above
  copyright notice, this list of conditions and the following
  disclaimer in the documentation and/or other materials
  provided with the distribution.
* Neither the name of the Google Inc. nor the names of its
  contributors may be used to endorse or promote products
  derived from this software without specific prior written
  permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

14.4.13.2. Multiple Background I/O Threads Patch from Percona

Oracle gratefully acknowledges the contribution of Percona, Inc. to improve InnoDB performance by implementing configurable background threads, as discussed in [Section 14.4.7.8, "Multiple Background I/O Threads"](#). InnoDB uses background threads to service various types of I/O requests. The change provides another way to make InnoDB more scalable on high end systems.

Changes from the Percona, Inc. contribution were incorporated in the following source code files: `ha_innodb.cc`, `os0file.c`, `os0file.h`, `srv0srv.c`, `srv0srv.h`, and `srv0start.c`.

This contribution is incorporated subject to the conditions contained in the file `COPYING.Percona`, which are reproduced here.

```
Copyright (c) 2008, 2009, Percona Inc.
```

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the Percona Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

14.4.13.3. Performance Patches from Sun Microsystems

Oracle gratefully acknowledges the following contributions from Sun Microsystems, Inc. to improve InnoDB performance:

- Introducing the PAUSE instruction inside spin loops, as discussed in [Section 14.4.7.13, "Using the PAUSE Instruction in InnoDB Spin Loops"](#). This change increases performance in high concurrency, CPU-bound workloads.
- Enabling inlining of functions and prefetch with Sun Studio.

Changes from the Sun Microsystems, Inc. contribution were incorporated in the following source code files: [univ.i](#), [utOut.c](#), and [utOut.h](#).

This contribution is incorporated subject to the conditions contained in the file [COPYING.Sun_Microsystems](#), which are reproduced here.

Copyright (c) 2009, Sun Microsystems, Inc.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Sun Microsystems, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,

INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

14.4.14. List of Parameters Changed in InnoDB 1.1 and InnoDB Plugin 1.0

14.4.14.1. New Parameters

Throughout the course of development, InnoDB 1.1 and its predecessor the InnoDB Plugin introduced new configuration parameters. The following table summarizes those parameters:

Table 14.8. InnoDB 1.1 New Parameter Summary

Name	Cmd-Line	Option File	System Var	Scope	Dynamic	Default
<code>innodb_adaptive_flushing</code>	YES	YES	YES	GLOBAL	YES	TRUE
<code>innodb_buffer_pool_instances</code>	YES	YES	YES	GLOBAL	YES	TRUE
<code>innodb_change_buffering</code>	YES	YES	YES	GLOBAL	YES	<code>inserts</code>
<code>innodb_file_format</code>	YES	YES	YES	GLOBAL	YES	Antelope
<code>innodb_file_format_check</code>	YES	YES	YES	GLOBAL	NO	1
<code>innodb_file_format_max</code>	YES	YES	YES	GLOBAL	YES	Antelope for a new database; Barracuda if any tables using that file format exist in the database
<code>innodb_io_capacity</code>	YES	YES	YES	GLOBAL	YES	200
<code>innodb_old_blocks_pct</code>	YES	YES	YES	GLOBAL	YES	37
<code>innodb_old_blocks_time</code>	YES	YES	YES	GLOBAL	YES	0
<code>innodb_purge_batch_size</code>	YES	YES	YES	GLOBAL	YES	0
<code>innodb_purge_threads</code>	YES	YES	YES	GLOBAL	YES	0
<code>innodb_read_ahead_threshold</code>	YES	YES	YES	GLOBAL	YES	56
<code>innodb_read_io_threads</code>	YES	YES	YES	GLOBAL	NO	4
<code>innodb_spin_wait_delay</code>	YES	YES	YES	GLOBAL	YES	6
<code>innodb_stats_sample_pages</code>	YES	YES	YES	GLOBAL	YES	8
<code>innodb_strict_mode</code>	YES	YES	YES	GLOBAL SESSION	YES	FALSE
<code>innodb_use_native_aio</code>	YES	YES	YES	GLOBAL	NO	TRUE
<code>innodb_use_sys_malloc</code>	YES	YES	YES	GLOBAL	NO	TRUE
<code>innodb_write_io_threads</code>	YES	YES	YES	GLOBAL	NO	4

- [innodb_adaptive_flushing \[1547\]](#)

Whether InnoDB uses a new algorithm to estimate the required rate of flushing. The default value is `TRUE`. This parameter was added in InnoDB storage engine 1.0.4. See [Section 14.4.7.12, “Controlling the Flushing Rate of Dirty Pages”](#) for more information.

- [innodb_change_buffering \[1550\]](#)

Whether InnoDB performs insert buffering. The default value is `"inserts"` (buffer insert operations). This parameter was added in InnoDB storage engine 1.0.3. See [Section 14.4.7.4, “Controlling InnoDB Change Buffering”](#) for more information.

- [innodb_file_format \[1554\]](#)

The default file format for new InnoDB tables. The default is Antelope. To enable support for table compression, change it to Barracuda. This parameter was added in InnoDB storage engine 1.0.1. See [Section 14.4.4.1, “Enabling File Formats”](#) for more information.

- [innodb_file_format_check \[1554\]](#) and [innodb_file_format_max \[1555\]](#)

Controls whether InnoDB performs file format compatibility checking when opening a database. The default value is `innodb-file-format-check=1`, with [innodb_file_format_max \[1555\]](#) set to the highest format that is used in the database (either `Barracuda` or `Antelope`). See [Section 14.4.4.2.1, “Compatibility Check When InnoDB Is Started”](#) for more information.

- [innodb_io_capacity \[1559\]](#)

The number of I/O operations that can be performed per second. The allowable value range is any number 100 or greater, and the default value is 200. This parameter was added in InnoDB storage engine 1.0.4. To reproduce the earlier behavior, use a value of 100. See [Section 14.4.7.11, “Controlling the Master Thread I/O Rate”](#) for more information.

- [innodb_old_blocks_pct \[1566\]](#)

Controls the desired percentage of “old” blocks in the LRU list of the buffer pool. The default value is 37 and the allowable value range is 5 to 95. This parameter was added in InnoDB storage engine 1.0.5. See [Section 14.4.7.15, “Making Buffer Pool Scan Resistant”](#) for more information.

- [innodb_old_blocks_time \[1566\]](#)

The time in milliseconds since the first access to a block during which it can be accessed again without being made “young”. The default value is 0 which means that blocks are moved to the “young” end of the LRU list at the first access. This parameter was added in InnoDB storage engine 1.0.5. See [Section 14.4.7.15, “Making Buffer Pool Scan Resistant”](#) for more information.

- [innodb_read_ahead_threshold \[1568\]](#)

Control the sensitivity of the linear read ahead. The allowable value range is 0 to 64 and the default value is 56. This parameter was added in InnoDB storage engine 1.0.4. See [Section 14.4.7.7, “Changes in the Read-Ahead Algorithm”](#) for more information.

- [innodb_read_io_threads \[1568\]](#)

The number of background I/O threads used for reads. The allowable value range is 1 to 64 and the default value is 4. This parameter was added in InnoDB storage engine 1.0.4. See [Section 14.4.7.8, “Multiple Background I/O Threads”](#) for more information.

- [innodb_spin_wait_delay \[1570\]](#)

Maximum delay between polling for a spin lock. The allowable value range is 0 (meaning unlimited) or positive integers and the default value is 6. This parameter was added in InnoDB storage engine 1.0.4. See [Section 14.4.7.14, “Control of Spin Lock Polling”](#) for more information.

- [innodb_stats_sample_pages \[1571\]](#)

The number of index pages to sample when calculating statistics. The allowable value range is 1-unlimited and the default value is 8. This parameter was added in InnoDB storage engine 1.0.2. See [Section 14.4.8.5, “Controlling Optimizer Statistics Estimation”](#) for more information.

- [innodb_strict_mode \[1572\]](#)

Whether InnoDB raises error conditions in certain cases, rather than issuing a warning. This parameter was added in InnoDB storage engine 1.0.2. See [Section 14.4.8.4, “InnoDB Strict Mode”](#) for more information.

- [innodb_use_sys_malloc \[1575\]](#)

Whether InnoDB uses its own memory allocator or an allocator of the operating system. The default value is ON (use an allocator of the underlying system). This parameter was added in InnoDB storage engine 1.0.3. See [Section 14.4.7.3, “Using Operating System Memory Allocators”](#) for more information.

- [innodb_write_io_threads \[1575\]](#)

The number of background I/O threads used for writes. The allowable value range is 1 to 64 and the default value is 4. This parameter was added in InnoDB storage engine 1.0.4. See [Section 14.4.7.8, “Multiple Background I/O Threads”](#) for more information.

14.4.14.2. Deprecated Parameters

Beginning in InnoDB storage engine 1.0.4, the following configuration parameter has been removed:

- [innodb_file_io_threads](#)

This parameter has been replaced by two new parameters [innodb_read_io_threads \[1568\]](#) and [innodb_write_io_threads \[1575\]](#). See [Section 14.4.7.8, “Multiple Background I/O Threads”](#) for more information.

14.4.14.3. Parameters with New Defaults

For better out-of-the-box performance, the following InnoDB configuration parameters have new default values since MySQL 5.1:

Table 14.9. InnoDB Parameters with New Defaults

Name	Old Default	New Default
innodb_additional_mem_pool_size	1MB	8MB
innodb_buffer_pool_size	8MB	128MB
innodb_change_buffering	inserts	all
innodb_file_format_check	ON	1
innodb_log_buffer_size	1MB	8MB
innodb_max_dirty_pages_pct	90	75

Name	Old Default	New Default
<code>innodb_sync_spin_loops</code>	20	30
<code>innodb_thread_concurrency</code>	8	0

14.5. The **MyISAM** Storage Engine

Before MySQL 5.5.5, **MyISAM** is the default storage engine. (The default was changed to **InnoDB** in MySQL 5.5.5.) **MyISAM** is based on the older (and no longer available) **ISAM** storage engine but has many useful extensions.

Table 14.10. **MyISAM Storage Engine Features**

Storage limits	256TB	Transactions	No	Locking granularity	Table
MVCC	No	Geospatial data type support	Yes	Geospatial indexing support	Yes
B-tree indexes	Yes	Hash indexes	No	Full-text search indexes	Yes
Clustered indexes	No	Data caches	No	Index caches	Yes
Compressed data	Yes ^a	Encrypted data ^b	Yes	Cluster database support	No
Replication support ^c	Yes	Foreign key support	No	Backup / point-in-time recovery ^d	Yes
Query cache support	Yes	Update statistics for data dictionary	Yes		

^aCompressed **MyISAM** tables are supported only when using the compressed row format. Tables using the compressed row format with **MyISAM** are read only.

^bImplemented in the server (via encryption functions), rather than in the storage engine.

^cImplemented in the server, rather than in the storage engine.

^dImplemented in the server, rather than in the storage engine.

Each **MyISAM** table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format. The data file has an `.MYD` (**MYData**) extension. The index file has an `.MYI` (**MYIndex**) extension.

To specify explicitly that you want a **MyISAM** table, indicate that with an **ENGINE** table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

As of MySQL 5.5.5, it is normally necessary to use **ENGINE** to specify the **MyISAM** storage engine because **InnoDB** is the default engine. Before 5.5.5, this is unnecessary because **MyISAM** is the default engine unless the default has been changed. To ensure that **MyISAM** is used in situations where the default might have been changed, include the **ENGINE** option explicitly.

You can check or repair **MyISAM** tables with the `mysqlcheck` client or `myisamchk` utility. You can also compress **MyISAM** tables with `myisampack` to take up much less space. See [Section 4.5.3, “mysqlcheck — A Table Maintenance Program”](#), [Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”](#), and [Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#).

MyISAM tables have the following characteristics:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirements for binary portability are that the machine uses two's-complement

signed integers and IEEE floating-point format. These requirements are widely used among mainstream machines. Binary compatibility might not be applicable to embedded systems, which sometimes have peculiar processors.

There is no significant speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it takes little more processing to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- All numeric key values are stored with the high byte first to permit better index compression.
- Large files (up to 63-bit file length) are supported on file systems and operating systems that support large files.
- There is a limit of $(2^{32})^2$ (1.844E+19) rows in a MyISAM table.
- The maximum number of indexes per MyISAM table is 64.

The maximum number of columns per index is 16.

- The maximum key length is 1000 bytes. This can also be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.
- When rows are inserted in sorted order (as when you are using an `AUTO_INCREMENT` column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.
- Internal handling of one `AUTO_INCREMENT` column per table is supported. MyISAM automatically updates this column for `INSERT` and `UPDATE` operations. This makes `AUTO_INCREMENT` columns faster (at least 10%). Values at the top of the sequence are not reused after being deleted. (When an `AUTO_INCREMENT` column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The `AUTO_INCREMENT` value can be reset with `ALTER TABLE` or `myisamchk`.
- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.
- MyISAM supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. A free block can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again. See [Section 8.10.3, “Concurrent Inserts”](#).
- You can put the data file and index file in different directories on different physical devices to get more speed with the `DATA DIRECTORY` and `INDEX DIRECTORY` table options to `CREATE TABLE`. See [Section 13.1.17, “CREATE TABLE Syntax”](#).
- `BLOB` and `TEXT` columns can be indexed.
- `NULL` values are permitted in indexed columns. This takes 0 to 1 bytes per key.
- Each character column can have a different character set. See [Section 10.1, “Character Set Support”](#).
- There is a flag in the MyISAM index file that indicates whether the table was closed correctly. If `mysqld` is started with the `--myisam-recover-options [454]` option, MyISAM tables are automatically checked when opened, and are repaired if the table wasn't closed properly.

- `myisamchk` marks tables as checked if you run it with the `--update-state [350]` option. `myisamchk --fast` checks only those tables that don't have this mark.
- `myisamchk --analyze` stores statistics for portions of keys, as well as for entire keys.
- `myisampack` can pack `BLOB` and `VARCHAR` columns.

`MyISAM` also supports the following features:

- Support for a true `VARCHAR` type; a `VARCHAR` column starts with a length stored in one or two bytes.
- Tables with `VARCHAR` columns may have fixed or dynamic row length.
- The sum of the lengths of the `VARCHAR` and `CHAR` columns in a table may be up to 64KB.
- Arbitrary length `UNIQUE` constraints.

Additional Resources

- A forum dedicated to the `MyISAM` storage engine is available at <http://forums.mysql.com/list.php?21>.

14.5.1. `MyISAM` Startup Options

The following options to `mysqld` can be used to change the behavior of `MyISAM` tables. For additional information, see [Section 5.1.3, “Server Command Options”](#).

Table 14.11. `MyISAM` Option/Variable Reference

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
bulk_insert_buffer_size [486]	Yes	Yes	Yes		Both	Yes
concurrent_insert [490]	Yes	Yes	Yes		Global	Yes
delay-key-write [441]	Yes	Yes			Global	Yes
- Variable: <code>delay_key_write</code>			Yes		Global	Yes
have_rtree_keys [505]			Yes		Global	No
key_buffer_size [508]	Yes	Yes	Yes		Global	Yes
log-isam [448]	Yes	Yes				
myisam-block-size [454]	Yes	Yes				
myisam_data_pointer_size [527]	Yes	Yes	Yes		Global	Yes
myisam_max_sort_size [528]	Yes	Yes	Yes		Global	Yes
myisam_mmap_size [528]	Yes	Yes	Yes		Global	No
myisam-recover [454]	Yes	Yes				
- Variable: <code>myisam_recover_options</code>						
myisam-recover-options [454]	Yes	Yes				
- Variable: <code>myisam_recover_options</code>						

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
myisam_recover_options [529]			Yes		Global	No
myisam_repair_threads [529]	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size [530]	Yes	Yes	Yes		Both	Yes
myisam_stats_method [530]	Yes	Yes	Yes		Both	Yes
myisam_use_mmap [531]	Yes	Yes	Yes		Global	Yes
skip-concurrent-insert [460]	Yes	Yes				
- Variable: concurrent_insert						
tmp_table_size [568]	Yes	Yes	Yes		Both	Yes

- `--myisam-recover-options=mode` [454]

Set the mode for automatic recovery of crashed [MyISAM](#) tables.

- `--delay-key-write=ALL` [441]

Don't flush key buffers between writes for any [MyISAM](#) table.



Note

If you do this, you should not access [MyISAM](#) tables from another program (such as from another MySQL server or with `myisamchk`) when the tables are in use. Doing so risks index corruption. Using `--external-locking` [444] does not eliminate this risk.

The following system variables affect the behavior of [MyISAM](#) tables. For additional information, see [Section 5.1.4, "Server System Variables"](#).

- `bulk_insert_buffer_size` [486]

The size of the tree cache used in bulk insert optimization.



Note

This is a limit *per thread*!

- `myisam_max_sort_file_size` [528]

The maximum size of the temporary file that MySQL is permitted to use while re-creating a [MyISAM](#) index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

- `myisam_sort_buffer_size` [530]

Set the size of the buffer used when recovering tables.

Automatic recovery is activated if you start `mysqld` with the `--myisam-recover-options` [454] option. In this case, when the server opens a [MyISAM](#) table, it checks whether the table is marked as crashed or whether the open count variable for the table is not 0 and you are running the server with external locking disabled. If either of these conditions is true, the following happens:

- The server checks the table for errors.
- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.
- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

If the recovery wouldn't be able to recover all rows from previously completed statements and you didn't specify `FORCE` in the value of the `--myisam-recover-options` [454] option, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

Note that if the automatic recovery value includes `BACKUP`, the recovery process creates files with names of the form `tbl_name-datetime.BAK`. You should have a `cron` script that automatically moves these files from the database directories to backup media.

14.5.2. Space Needed for Keys

`MyISAM` tables use B-tree indexes. You can roughly calculate the size for the index file as $(key_length + 4) / 0.67$, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if a string column has a lot of trailing space or is a `VARCHAR` column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In `MyISAM` tables, you can also prefix compress numbers by specifying the `PACK_KEYS=1` table option when you create the table. Numbers are stored with the high byte first, so this helps when you have many integer keys that have an identical prefix.

14.5.3. `MyISAM` Table Storage Formats

`MyISAM` supports three different storage formats. Two of them, fixed and dynamic format, are chosen automatically depending on the type of columns you are using. The third, compressed format, can be created only with the `myisampack` utility (see Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only `MyISAM` Tables”).

When you use `CREATE TABLE` or `ALTER TABLE` for a table that has no `BLOB` or `TEXT` columns, you can force the table format to `FIXED` or `DYNAMIC` with the `ROW_FORMAT` table option.

See Section 13.1.17, “`CREATE TABLE Syntax`”, for information about `ROW_FORMAT`.

You can decompress (unpack) compressed `MyISAM` tables using `myisamchk --unpack` [352]; see Section 4.6.3, “`myisamchk` — `MyISAM` Table-Maintenance Utility”, for more information.

14.5.3.1. Static (Fixed-Length) Table Characteristics

Static format is the default for MyISAM tables. It is used when the table contains no variable-length columns (`VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT`). Each row is stored using a fixed number of bytes.

Of the three MyISAM storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats due to the ease with which rows in the data file can be found on disk: To look up a row based on a row number in the index, multiply the row number by the row length to calculate the row position. Also, when scanning a table, it is very easy to read a constant number of rows with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format MyISAM file. In this case, `myisamchk` can easily determine where each row starts and ends, so it can usually reclaim all rows except the partially written one. Note that MyISAM table indexes can always be reconstructed based on the data rows.



Note

Fixed-length row format is only available for tables without `BLOB` or `TEXT` columns. Creating a table with these columns with an explicit `ROW_FORMAT` clause will not raise an error or warning; the format specification will be ignored.

Static-format tables have these characteristics:

- `CHAR` and `VARCHAR` columns are space-padded to the specified column width, although the column type is not altered. `BINARY` and `VARBINARY` columns are padded with `0x00` bytes to the column width.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because rows are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of rows and want to return free disk space to the operating system. To do this, use `OPTIMIZE TABLE` or `myisamchk -r`.
- Usually require more disk space than dynamic-format tables.

14.5.3.2. Dynamic Table Characteristics

Dynamic storage format is used if a MyISAM table contains any variable-length columns (`VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT`), or if the table was created with the `ROW_FORMAT=DYNAMIC` table option.

Dynamic format is a little more complex than static format because each row has a header that indicates how long it is. A row can become fragmented (stored in noncontiguous pieces) when it is made longer as a result of an update.

You can use `OPTIMIZE TABLE` or `myisamchk -r` to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

Dynamic-format tables have these characteristics:

- All string columns are dynamic except those with a length less than four.
- Each row is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). Note that this does not include columns that contain `NULL`.

values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Nonempty strings are saved as a length byte plus the string contents.

- Much less disk space usually is required than for fixed-length tables.
- Each row uses only as much space as is required. However, if a row becomes larger, it is split into as many pieces as are required, resulting in row fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run `OPTIMIZE TABLE` or `myisamchk -r` from time to time to improve performance. Use `myisamchk -ei` to obtain table statistics.
- More difficult than static-format tables to reconstruct after a crash, because rows may be fragmented into many pieces and links (fragments) may be missing.
- The expected row length for dynamic-sized rows is calculated using the following expression:

```

3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
+ (length of strings)
+ (number of NULL columns + 7) / 8

```

There is a penalty of 6 bytes for each link. A dynamic row is linked whenever an update causes an enlargement of the row. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, another link is created. You can find the number of links using `myisamchk -ed`. All links may be removed with `OPTIMIZE TABLE` or `myisamchk -r`.

14.5.3.3. Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the `myisampack` tool. Compressed tables can be uncompressed with `myisamchk`.

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is helpful when using slow disks (such as CD-ROMs).
- Each row is compressed separately, so there is very little access overhead. The header for a row takes up one to three bytes depending on the biggest row in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:
 - Suffix space compression.
 - Prefix space compression.
 - Numbers with a value of zero are stored using one bit.
 - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.
 - If a column has only a small set of possible values, the data type is converted to `ENUM`.
 - A column may use any combination of the preceding compression types.
- Can be used for fixed-length or dynamic-length rows.

**Note**

While a compressed table is read only, and you cannot therefore update or add rows in the table, DDL (Data Definition Language) operations are still valid. For example, you may still use `DROP` to drop the table, and `TRUNCATE TABLE` to empty the table.

14.5.4. MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted. The following discussion describes how this can happen and how to handle it.

14.5.4.1. Corrupted MyISAM Tables

Even though the MyISAM table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get corrupted tables if any of the following events occur:

- The `mysqld` process is killed in the middle of a write.
- An unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware failures.
- You are using an external program (such as `myisamchk`) to modify a table that is being modified by the server at the same time.
- A software bug in the MySQL or MyISAM code.

Typical symptoms of a corrupt table are:

- You get the following error while selecting data from the table:

```
Incorrect key file for table: '...'. Try to repair it
```

- Queries don't find rows in the table or return incomplete results.

You can check the health of a MyISAM table using the `CHECK TABLE` statement, and repair a corrupted MyISAM table with `REPAIR TABLE`. When `mysqld` is not running, you can also check or repair a table with the `myisamchk` command. See [Section 13.7.2.2, “CHECK TABLE Syntax”](#), [Section 13.7.2.5, “REPAIR TABLE Syntax”](#), and [Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of a server crash. You can verify this easily by looking for a recent `restarted mysqld` message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption may have occurred during normal operation. This is a bug. You should try to create a reproducible test case that demonstrates the problem. See [Section C.5.4.2, “What to Do If MySQL Keeps Crashing”](#), and [MySQL Internals: Porting to Other Systems](#).

14.5.4.2. Problems from Tables Not Being Closed Properly

Each MyISAM index file (`.MYI` file) has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from `CHECK TABLE` or `myisamchk`, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because a `FLUSH TABLES` operation was performed or because there is no room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can become incorrect only under these conditions:

- A `MyISAM` table is copied without first issuing `LOCK TABLES` and `FLUSH TABLES`.
- MySQL has crashed between an update and the final close. (Note that the table may still be okay, because MySQL always issues writes for everything between each statement.)
- A table was modified by `myisamchk --recover` or `myisamchk --update-state` at the same time that it was in use by `mysqld`.
- Multiple `mysqld` servers are using the table and one server performed a `REPAIR TABLE` or `CHECK TABLE` on the table while it was in use by another server. In this setup, it is safe to use `CHECK TABLE`, although you might get the warning from other servers. However, `REPAIR TABLE` should be avoided because when one server replaces the data file with a new one, this is not known to the other servers.

In general, it is a bad idea to share a data directory among multiple servers. See [Section 5.3, “Running Multiple MySQL Instances on One Machine”](#), for additional discussion.

14.6. The MEMORY Storage Engine

The `MEMORY` storage engine (formerly known as `HEAP`) creates special-purpose tables with contents that are stored in memory. Because the data is vulnerable to crashes, hardware issues, or power outages, only use these tables as temporary work areas or read-only caches for data pulled from other tables.

Table 14.12. MEMORY Storage Engine Features

Storage limits	RAM	Transactions	No	Locking granularity	Table
MVCC	No	Geospatial data type support	No	Geospatial indexing support	No
B-tree indexes	Yes	Hash indexes	Yes	Full-text search indexes	No
Clustered indexes	No	Data caches	N/A	Index caches	N/A
Compressed data	No	Encrypted data^a	Yes	Cluster database support	No
Replication support^b	Yes	Foreign key support	No	Backup / point-in-time recovery^c	Yes

Query cache support	Yes	Update statistics for data dictionary	Yes		
----------------------------	-----	--	-----	--	--

^aImplemented in the server (via encryption functions), rather than in the storage engine.

^bImplemented in the server, rather than in the storage engine.

^cImplemented in the server, rather than in the storage engine.

When to Use MEMORY or MySQL Cluster. Developers looking to deploy applications that use the MEMORY storage engine for important, highly available, or frequently updated data should consider whether MySQL Cluster is a better choice. A typical use case for the MEMORY engine involves these characteristics:

- Operations involving transient, non-critical data such as session management or caching. When the MySQL server halts or restarts, the data in MEMORY tables is lost.
- In-memory storage for fast access and low latency. Data volume can fit entirely in memory without causing the operating system to swap out virtual memory pages.
- A read-only or read-mostly data access pattern (limited updates).

MySQL Cluster offers the same features as the MEMORY engine with higher performance levels, and provides additional features not available with MEMORY:

- Row-level locking and multiple-thread operation for low contention between clients.
- Scalability even with statement mixes that include writes.
- Optional disk-backed operation for data durability.
- Shared-nothing architecture and multiple-host operation with no single point of failure, enabling 99.999% availability.
- Automatic data distribution across nodes; application developers need not craft custom sharding or partitioning solutions.
- Support for variable-length data types (including BLOB and TEXT) not supported by MEMORY.

For a white paper with more detailed comparison of the MEMORY storage engine and MySQL Cluster, see [Scaling Web Services with MySQL Cluster: An Alternative to the MySQL Memory Storage Engine](#). This white paper includes a performance study of the two technologies and a step-by-step guide describing how existing MEMORY users can migrate to MySQL Cluster.

Performance Characteristics

MEMORY performance is constrained by contention resulting from single-thread execution and table lock overhead when processing updates. This limits scalability when load increases, particularly for statement mixes that include writes.

Despite the in-memory processing for MEMORY tables, they are not necessarily faster than InnoDB tables on a busy server, for general-purpose queries, or under a read/write workload. In particular, the table locking involved with performing updates can slow down concurrent usage of MEMORY tables from multiple sessions.

Depending on the kinds of queries performed on a MEMORY table, you might create indexes as either the default hash data structure (for looking up single values based on a unique key), or a general-purpose B-tree data structure (for all kinds of queries involving equality, inequality, or range operators such as less than or greater than). The following sections illustrate the syntax for creating both kinds of indexes. A common performance issue is using the default hash indexes in workloads where B-tree indexes are more efficient.

Physical Characteristics of MEMORY Tables

The MEMORY storage engine associates each table with one disk file, which stores the table definition (not the data). The file name begins with the table name and has an extension of `.frm`.

MEMORY tables have the following characteristics:

- Space for MEMORY tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows are put in a linked list and are reused when you insert new data into the table. MEMORY tables also have none of the problems commonly associated with deletes plus inserts in hashed tables.
- MEMORY tables use a fixed-length row-storage format. Variable-length types such as VARCHAR are stored using a fixed length.
- MEMORY tables cannot contain BLOB or TEXT columns.
- MEMORY includes support for AUTO_INCREMENT columns.
- Non-TEMPORARY MEMORY tables are shared among all clients, just like any other non-TEMPORARY table.

DDL Operations for MEMORY Tables

To create a MEMORY table, specify the clause `ENGINE=MEMORY` on the `CREATE TABLE` statement.

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

As indicated by the engine name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast for single-value lookups, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in `.frm` files on disk, but they are empty when the server restarts.

This example shows how you might create, use, and remove a MEMORY table:

```
mysql> CREATE TABLE test ENGINE=MEMORY
->     SELECT ip,SUM(downloads) AS down
->     FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

The maximum size of MEMORY tables is limited by the `max_heap_table_size` [521] system variable, which has a default value of 16MB. To enforce different size limits for MEMORY tables, change the value of this variable. The value in effect for `CREATE TABLE`, or a subsequent `ALTER TABLE` or `TRUNCATE TABLE`, is the value used for the life of the table. A server restart also sets the maximum size of existing MEMORY tables to the global `max_heap_table_size` [521] value. You can set the size for individual tables as described later in this section.

Indexes

The MEMORY storage engine supports both HASH and BTREE indexes. You can specify one or the other for a given index by adding a `USING` clause as shown here:

```
CREATE TABLE lookup
  (id INT, INDEX USING HASH (id))
ENGINE = MEMORY;
CREATE TABLE lookup
  (id INT, INDEX USING BTREE (id))
```

```
ENGINE = MEMORY;
```

For general characteristics of B-tree and hash indexes, see [Section 8.3.1, “How MySQL Uses Indexes”](#).

`MEMORY` tables can have up to 64 indexes per table, 16 columns per index and a maximum key length of 3072 bytes.

If a `MEMORY` table hash index has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a `BTREE` index to avoid this problem.

`MEMORY` tables can have nonunique keys. (This is an uncommon feature for implementations of hash indexes.)

Columns that are indexed can contain `NULL` values.

User-Created and Temporary Tables

`MEMORY` table contents are stored in memory, which is a property that `MEMORY` tables share with internal temporary tables that the server creates on the fly while processing queries. However, the two types of tables differ in that `MEMORY` tables are not subject to storage conversion, whereas internal temporary tables are:

- If an internal temporary table becomes too large, the server automatically converts it to on-disk storage, as described in [Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#).
- User-created `MEMORY` tables are never converted to disk tables.

Loading Data

To populate a `MEMORY` table when the MySQL server starts, you can use the `--init-file [445]` option. For example, you can put statements such as `INSERT INTO ... SELECT` or `LOAD DATA INFILE` into this file to load the table from a persistent data source. See [Section 5.1.3, “Server Command Options”](#), and [Section 13.2.6, “LOAD DATA INFILE Syntax”](#).

For loading data into `MEMORY` tables accessed by other sessions concurrently, `MEMORY` supports `INSERT DELAYED`. See [Section 13.2.5.2, “INSERT DELAYED Syntax”](#).

MEMORY Tables and Replication

A server's `MEMORY` tables become empty when it is shut down and restarted. If the server is a replication master, its slaves are not aware that these tables have become empty, so you see out-of-date content if you select data from the tables on the slaves. To synchronize master and slave `MEMORY` tables, when a `MEMORY` table is used on a master for the first time since it was started, a `DELETE` statement is written to the master's binary log, to empty the table on the slaves also. The slave still has outdated data in the table during the interval between the master's restart and its first use of the table. To avoid this interval when a direct query to the slave could return stale data, use the `--init-file [445]` option to populate the `MEMORY` table on the master at startup.

Managing Memory Use

The server needs sufficient memory to maintain all `MEMORY` tables that are in use at the same time.

Memory is not reclaimed if you delete individual rows from a `MEMORY` table. Memory is reclaimed only when the entire table is deleted. Memory that was previously used for deleted rows is re-used for new

rows within the same table. To free all the memory used by a `MEMORY` table when you no longer require its contents, execute `DELETE` or `TRUNCATE TABLE` to remove all rows, or remove the table altogether using `DROP TABLE`. To free up the memory used by deleted rows, use `ALTER TABLE ENGINE=MEMORY` to force a table rebuild.

The memory needed for one row in a `MEMORY` table is calculated using the following expression:

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) * 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) * 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

`ALIGN()` represents a round-up factor to cause the row length to be an exact multiple of the `char` pointer size. `sizeof(char*)` is 4 on 32-bit machines and 8 on 64-bit machines.

As mentioned earlier, the `max_heap_table_size` [521] system variable sets the limit on the maximum size of `MEMORY` tables. To control the maximum size for individual tables, set the session value of this variable before creating each table. (Do not change the global `max_heap_table_size` [521] value unless you intend the value to be used for `MEMORY` tables created by all clients.) The following example creates two `MEMORY` tables, with a maximum size of 1MB and 2MB, respectively:

```
mysql> SET max_heap_table_size = 1024*1024;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.01 sec)

mysql> SET max_heap_table_size = 1024*1024*2;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t2 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.00 sec)
```

Both tables revert to the server's global `max_heap_table_size` [521] value if the server restarts.

You can also specify a `MAX_ROWS` table option in `CREATE TABLE` statements for `MEMORY` tables to provide a hint about the number of rows you plan to store in them. This does not enable the table to grow beyond the `max_heap_table_size` [521] value, which still acts as a constraint on maximum table size. For maximum flexibility in being able to use `MAX_ROWS`, set `max_heap_table_size` [521] at least as high as the value to which you want each `MEMORY` table to be able to grow.

Additional Resources

A forum dedicated to the `MEMORY` storage engine is available at <http://forums.mysql.com/list.php?92>.

14.7. The `CSV` Storage Engine

The `CSV` storage engine stores data in text files using comma-separated values format.

The `CSV` storage engine is always compiled into the MySQL server.

To examine the source for the `CSV` engine, look in the `storage/csv` directory of a MySQL source distribution.

When you create a `CSV` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine also creates a data file. Its name begins with the table name and has a `.CSV` extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

```
mysql> CREATE TABLE test (i INT NOT NULL, c CHAR(10) NOT NULL)
```

```

-> ENGINE = CSV;
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM test;
+-----+-----+
| i     | c           |
+-----+-----+
| 1    | record one |
| 2    | record two |
+-----+-----+
2 rows in set (0.00 sec)
    
```

Creating a CSV table also creates a corresponding Metafile that stores the state of the table and the number of rows that exist in the table. The name of this file is the same as the name of the table with the extension `CSM`.

If you examine the `test.CSV` file in the database directory created by executing the preceding statements, its contents should look like this:

```

"1","record one"
"2","record two"
    
```

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel or StarOffice Calc.

14.7.1. Repairing and Checking CSV Tables

The CSV storage engines supports the `CHECK` and `REPAIR` statements to verify and if possible repair a damaged CSV table.

When running the `CHECK` statement, the CSV file will be checked for validity by looking for the correct field separators, escaped fields (matching or missing quotation marks), the correct number of fields compared to the table definition and the existence of a corresponding CSV metafile. The first invalid row discovered will report an error. Checking a valid table produces output like that shown below:

```

mysql> check table csvtest;
+-----+-----+-----+-----+
| Table          | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.csvtest  | check   | status   | OK       |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
    
```

A check on a corrupted table returns a fault:

```

mysql> check table csvtest;
+-----+-----+-----+-----+
| Table          | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.csvtest  | check   | error    | Corrupt  |
+-----+-----+-----+-----+
1 row in set (0.01 sec)
    
```

If the check fails, the table is marked as crashed (corrupt). Once a table has been marked as corrupt, it is automatically repaired when you next run `CHECK` or execute a `SELECT` statement. The corresponding corrupt status and new status will be displayed when running `CHECK`:

```

mysql> check table csvtest;
    
```

```

+-----+-----+-----+-----+
| Table          | Op      | Msg_type | Msg_text      |
+-----+-----+-----+-----+
| test.csvtest   | check   | warning  | Table is marked as crashed |
| test.csvtest   | check   | status   | OK            |
+-----+-----+-----+-----+
2 rows in set (0.08 sec)
    
```

To repair a table you can use [REPAIR](#), this copies as many valid rows from the existing CSV data as possible, and then replaces the existing CSV file with the recovered rows. Any rows beyond the corrupted data are lost.

```

mysql> repair table csvtest;
+-----+-----+-----+-----+
| Table          | Op      | Msg_type | Msg_text      |
+-----+-----+-----+-----+
| test.csvtest   | repair  | status   | OK            |
+-----+-----+-----+-----+
1 row in set (0.02 sec)
    
```



Warning

Note that during repair, only the rows from the CSV file up to the first damaged row are copied to the new table. All other rows from the first damaged row to the end of the table are removed, even valid rows.

14.7.2. CSV Limitations

The [CSV](#) storage engine does not support indexing.

Partitioning is not supported for tables using the [CSV](#) storage engine.

All tables that you create using the [CSV](#) storage engine must have the [NOT NULL](#) attribute on all columns. However, for backward compatibility, you can continue to use tables with nullable columns that were created in previous MySQL releases. (Bug #32050)

14.8. The [ARCHIVE](#) Storage Engine

The [ARCHIVE](#) storage engine is used for storing large amounts of data without indexes in a very small footprint.

Table 14.13. [ARCHIVE](#) Storage Engine Features

Storage limits	None	Transactions	No	Locking granularity	Table
MVCC	No	Geospatial data type support	Yes	Geospatial indexing support	No
B-tree indexes	No	Hash indexes	No	Full-text search indexes	No
Clustered indexes	No	Data caches	No	Index caches	No
Compressed data	Yes	Encrypted data^a	Yes	Cluster database support	No
Replication support^b	Yes	Foreign key support	No	Backup / point-in-time recovery^c	Yes
Query cache support	Yes	Update statistics for data dictionary	Yes		

^aImplemented in the server (via encryption functions), rather than in the storage engine.

^bImplemented in the server, rather than in the storage engine.

^cImplemented in the server, rather than in the storage engine.

The `ARCHIVE` storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_ARCHIVE_STORAGE_ENGINE` option.

To examine the source for the `ARCHIVE` engine, look in the `storage/archive` directory of a MySQL source distribution.

You can check whether the `ARCHIVE` storage engine is available with the `SHOW ENGINES` statement.

When you create an `ARCHIVE` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine creates other files, all having names beginning with the table name. The data file has an extension of `.ARZ`. An `.ARN` file may appear during optimization operations.

The `ARCHIVE` engine supports `INSERT` and `SELECT`, but not `DELETE`, `REPLACE`, or `UPDATE`. It does support `ORDER BY` operations, `BLOB` columns, and basically all but spatial data types (see [Section 12.17.4.1, “MySQL Spatial Data Types”](#)). The `ARCHIVE` engine uses row-level locking.

The `ARCHIVE` engine supports the `AUTO_INCREMENT` column attribute. The `AUTO_INCREMENT` column can have either a unique or nonunique index. Attempting to create an index on any other column results in an error. The `ARCHIVE` engine also supports the `AUTO_INCREMENT` table option in `CREATE TABLE` and `ALTER TABLE` statements to specify the initial sequence value for a new table or reset the sequence value for an existing table, respectively.

The `ARCHIVE` engine ignores `BLOB` columns if they are not requested and scans past them while reading.

Storage: Rows are compressed as they are inserted. The `ARCHIVE` engine uses `zlib` lossless data compression (see <http://www.zlib.net/>). You can use `OPTIMIZE TABLE` to analyze the table and pack it into a smaller format (for a reason to use `OPTIMIZE TABLE`, see later in this section). The engine also supports `CHECK TABLE`. There are several types of insertions that are used:

- An `INSERT` statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A `SELECT` forces a flush to occur, unless the only insertions that have come in were `INSERT DELAYED` (those flush as necessary). See [Section 13.2.5.2, “INSERT DELAYED Syntax”](#).
- A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A `SELECT` never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

Retrieval: On retrieval, rows are uncompressed on demand; there is no row cache. A `SELECT` operation performs a complete table scan: When a `SELECT` occurs, it finds out how many rows are currently available and reads that number of rows. `SELECT` is performed as a consistent read. Note that lots of `SELECT` statements during insertion can deteriorate the compression, unless only bulk or delayed inserts are used. To achieve better compression, you can use `OPTIMIZE TABLE` or `REPAIR TABLE`. The number of rows in `ARCHIVE` tables reported by `SHOW TABLE STATUS` is always accurate. See [Section 13.7.2.4, “OPTIMIZE TABLE Syntax”](#), [Section 13.7.2.5, “REPAIR TABLE Syntax”](#), and [Section 13.7.5.37, “SHOW TABLE STATUS Syntax”](#).

Additional Resources

- A forum dedicated to the `ARCHIVE` storage engine is available at <http://forums.mysql.com/list.php?112>.

14.9. The `BLACKHOLE` Storage Engine

The `BLACKHOLE` storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

To enable the `BLACKHOLE` storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_BLACKHOLE_STORAGE_ENGINE` option.

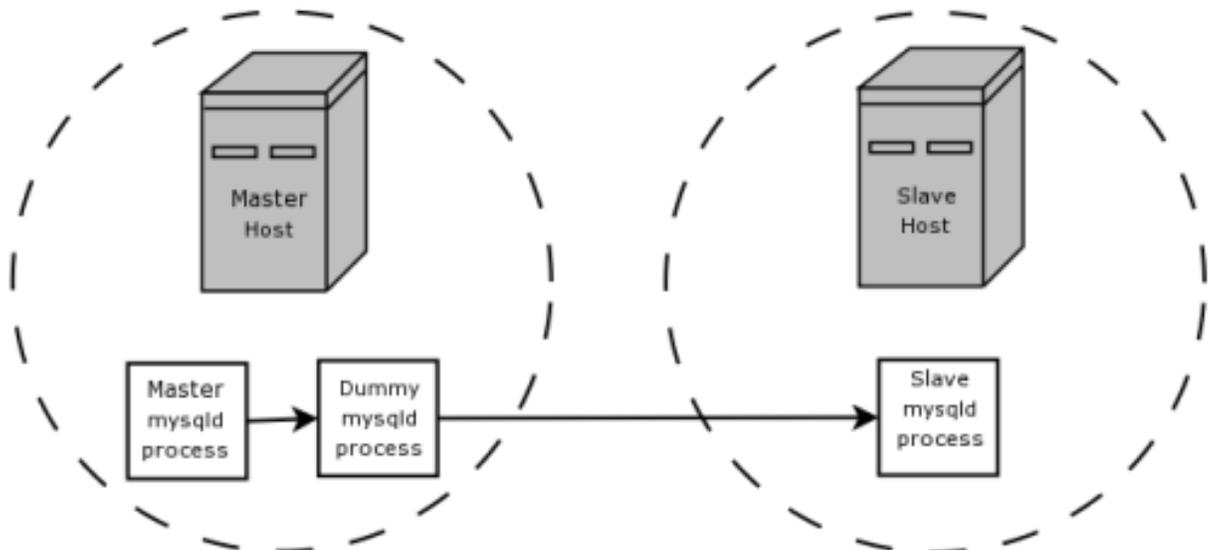
To examine the source for the `BLACKHOLE` engine, look in the `sql` directory of a MySQL source distribution.

When you create a `BLACKHOLE` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. There are no other files associated with the table.

The `BLACKHOLE` storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

You can check whether the `BLACKHOLE` storage engine is available with the `SHOW ENGINES` statement.

Inserts into a `BLACKHOLE` table do not store any data, but if the binary log is enabled, the SQL statements are logged (and replicated to slave servers). This can be useful as a repeater or filter mechanism. Suppose that your application requires slave-side filtering rules, but transferring all binary log data to the slave first results in too much traffic. In such a case, it is possible to set up on the master host a “dummy” slave process whose default storage engine is `BLACKHOLE`, depicted as follows:



The master writes to its binary log. The “dummy” `mysqld` process acts as a slave, applying the desired combination of `replicate-do-*` and `replicate-ignore-*` rules, and writes a new, filtered binary log of its own. (See [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).) This filtered log is provided to the slave.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional `mysqld` process on the replication master host. This type of setup can be repeated with additional replication slaves.

`INSERT` triggers for `BLACKHOLE` tables work as expected. However, because the `BLACKHOLE` table does not actually store any data, `UPDATE` and `DELETE` triggers are not activated: The `FOR EACH ROW` clause in the trigger definition does not apply because there are no rows.

Other possible uses for the `BLACKHOLE` storage engine include:

- Verification of dump file syntax.
- Measurement of the overhead from binary logging, by comparing performance using `BLACKHOLE` with and without binary logging enabled.
- `BLACKHOLE` is essentially a “no-op” storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

The `BLACKHOLE` engine is transaction-aware, in the sense that committed transactions are written to the binary log and rolled-back transactions are not.

Blackhole Engine and Auto Increment Columns

The Blackhole engine is a no-op engine. Any operations performed on a table using Blackhole will have no effect. This should be born in mind when considering the behavior of primary key columns that auto increment. The engine will not automatically increment field values, and does not retain auto increment field state. This has important implications in replication.

Consider the following replication scenario where all three of the following conditions apply:

1. On a master server there is a blackhole table with an auto increment field that is a primary key.
2. On a slave the same table exists but using the MyISAM engine.
3. Inserts are performed into the master's table without explicitly setting the auto increment value in the `INSERT` statement itself or through using a `SET INSERT_ID` statement.

In this scenario replication will fail with a duplicate entry error on the primary key column.

In statement based replication, the value of `INSERT_ID` in the context event will always be the same. Replication will therefore fail due to trying insert a row with a duplicate value for a primary key column.

In row based replication, the value that the engine returns for the row always be the same for each insert. This will result in the slave attempting to replay two insert log entries using the same value for the primary key column, and so replication will fail.

Column Filtering

When using row-based replication, (`binlog_format=ROW` [1899]), a slave where the last columns are missing from a table is supported, as described in the section [Section 16.4.1.8, “Replication with Differing Table Definitions on Master and Slave”](#).

This filtering works on the slave side, that is, the columns are copied to the slave before they are filtered out. There are at least two cases where it is not desirable to copy the columns to the slave:

1. If the data is confidential, so the slave server should not have access to it.

- If the master has many slaves, filtering before sending to the slaves may reduce network traffic.

Master column filtering can be achieved using the [BLACKHOLE](#) engine. This is carried out in a way similar to how master table filtering is achieved - by using the [BLACKHOLE](#) engine and the `--replicate-do-table` [1878] or `--replicate-ignore-table` [1878] option.

The setup for the master is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N,
                 secret_col_1, ..., secret_col_M) ENGINE=MyISAM;
```

The setup for the trusted slave is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=BLACKHOLE;
```

The setup for the untrusted slave is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=MyISAM;
```

14.10. The MERGE Storage Engine

The [MERGE](#) storage engine, also known as the [MRG_MyISAM](#) engine, is a collection of identical [MyISAM](#) tables that can be used as one. “Identical” means that all tables have identical column and index information. You cannot merge [MyISAM](#) tables in which the columns are listed in a different order, do not have exactly the same columns, or have the indexes in different order. However, any or all of the [MyISAM](#) tables can be compressed with [myisampack](#). See [Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#). Differences in table options such as [AVG_ROW_LENGTH](#), [MAX_ROWS](#), or [PACK_KEYS](#) do not matter.

An alternative to a [MERGE](#) table is a partitioned table, which stores partitions of a single table in separate files. Partitioning enables some operations to be performed more efficiently and is not limited to the [MyISAM](#) storage engine. For more information, see [Chapter 18, Partitioning](#).

When you create a [MERGE](#) table, MySQL creates two files on disk. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format, and an `.MRG` file contains the names of the underlying [MyISAM](#) tables that should be used as one. The tables do not have to be in the same database as the [MERGE](#) table.

You can use [SELECT](#), [DELETE](#), [UPDATE](#), and [INSERT](#) on [MERGE](#) tables. You must have [SELECT](#) [697], [DELETE](#) [696], and [UPDATE](#) [697] privileges on the [MyISAM](#) tables that you map to a [MERGE](#) table.



Note

The use of [MERGE](#) tables entails the following security issue: If a user has access to [MyISAM](#) table *t*, that user can create a [MERGE](#) table *m* that accesses *t*. However, if the user's privileges on *t* are subsequently revoked, the user can continue to access *t* by doing so through *m*.

Use of [DROP TABLE](#) with a [MERGE](#) table drops only the [MERGE](#) specification. The underlying tables are not affected.

To create a [MERGE](#) table, you must specify a `UNION=(list-of-tables)` option that indicates which [MyISAM](#) tables to use. You can optionally specify an `INSERT_METHOD` option to control how inserts into the [MERGE](#) table take place. Use a value of `FIRST` or `LAST` to cause inserts to be made in the first or last underlying table, respectively. If you specify no `INSERT_METHOD` option or if you specify it with a value of `NO`, inserts into the [MERGE](#) table are not permitted and attempts to do so result in an error.

The following example shows how to create a MERGE table:

```
mysql> CREATE TABLE t1 (
->   a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
->   message CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE t2 (
->   a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
->   message CHAR(20)) ENGINE=MyISAM;
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
->   a INT NOT NULL AUTO_INCREMENT,
->   message CHAR(20), INDEX(a))
->   ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Note that column `a` is indexed as a PRIMARY KEY in the underlying MyISAM tables, but not in the MERGE table. There it is indexed but not as a PRIMARY KEY because a MERGE table cannot enforce uniqueness over the set of underlying tables. (Similarly, a column with a UNIQUE index in the underlying tables should be indexed in the MERGE table but not as a UNIQUE index.)

After creating the MERGE table, you can use it to issue queries that operate on the group of tables as a whole:

```
mysql> SELECT * FROM total;
+-----+-----+
| a | message |
+-----+-----+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+-----+-----+
```

To remap a MERGE table to a different collection of MyISAM tables, you can use one of the following methods:

- DROP the MERGE table and re-create it.
- Use ALTER TABLE `tbl_name` UNION=(...) to change the list of underlying tables.

It is also possible to use ALTER TABLE ... UNION=() (that is, with an empty UNION clause) to remove all of the underlying tables.

The underlying table definitions and indexes must conform closely to the definition of the MERGE table. Conformance is checked when a table that is part of a MERGE table is opened, not when the MERGE table is created. If any table fails the conformance checks, the operation that triggered the opening of the table fails. This means that changes to the definitions of tables within a MERGE may cause a failure when the MERGE table is accessed. The conformance checks applied to each table are:

- The underlying table and the MERGE table must have the same number of columns.
- The column order in the underlying table and the MERGE table must match.
- Additionally, the specification for each corresponding column in the parent MERGE table and the underlying tables are compared and must satisfy these checks:
 - The column type in the underlying table and the MERGE table must be equal.
 - The column length in the underlying table and the MERGE table must be equal.

- The column of the underlying table and the `MERGE` table can be `NULL`.
- The underlying table must have at least as many indexes as the `MERGE` table. The underlying table may have more indexes than the `MERGE` table, but cannot have fewer.



Note

A known issue exists where indexes on the same columns must be in identical order, in both the `MERGE` table and the underlying `MyISAM` table. See Bug #33653.

Each index must satisfy these checks:

- The index type of the underlying table and the `MERGE` table must be the same.
- The number of index parts (that is, multiple columns within a compound index) in the index definition for the underlying table and the `MERGE` table must be the same.
- For each index part:
 - Index part lengths must be equal.
 - Index part types must be equal.
 - Index part languages must be equal.
 - Check whether index parts can be `NULL`.

If a `MERGE` table cannot be opened or used because of a problem with an underlying table, `CHECK TABLE` displays information about which table caused the problem.

Additional Resources

- A forum dedicated to the `MERGE` storage engine is available at <http://forums.mysql.com/list.php?93>.

14.10.1. `MERGE` Table Advantages and Disadvantages

`MERGE` tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with `myisampack`, and then create a `MERGE` table to use them as one.
- Obtain more speed. You can split a large read-only table based on some criteria, and then put individual tables on different disks. A `MERGE` table structured this way could be much faster than using a single large table.
- Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the underlying tables for some queries and use a `MERGE` table for others. You can even have many different `MERGE` tables that use overlapping sets of tables.
- Perform more efficient repairs. It is easier to repair individual smaller tables that are mapped to a `MERGE` table than to repair a single large table.
- Instantly map many tables as one. A `MERGE` table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, `MERGE` table collections are *very* fast to create or remap. (You must still specify the index definitions when you create a `MERGE` table, even though no indexes are created.)

- If you have a set of tables from which you create a large table on demand, you can instead create a [MERGE](#) table from them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each [MyISAM](#) table is bound by this limit, but a collection of [MyISAM](#) tables is not.
- You can create an alias or synonym for a [MyISAM](#) table by defining a [MERGE](#) table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and `memcpy()` calls for each read).

The disadvantages of [MERGE](#) tables are:

- You can use only identical [MyISAM](#) tables for a [MERGE](#) table.
- Some [MyISAM](#) features are unavailable in [MERGE](#) tables. For example, you cannot create [FULLTEXT](#) indexes on [MERGE](#) tables. (You can create [FULLTEXT](#) indexes on the underlying [MyISAM](#) tables, but you cannot search the [MERGE](#) table with a full-text search.)
- If the [MERGE](#) table is nontemporary, all underlying [MyISAM](#) tables must be nontemporary. If the [MERGE](#) table is temporary, the [MyISAM](#) tables can be any mix of temporary and nontemporary.
- [MERGE](#) tables use more file descriptors than [MyISAM](#) tables. If 10 clients are using a [MERGE](#) table that maps to 10 tables, the server uses $(10 \times 10) + 10$ file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Index reads are slower. When you read an index, the [MERGE](#) storage engine needs to issue a read on all underlying tables to check which one most closely matches a given index value. To read the next index value, the [MERGE](#) storage engine needs to search the read buffers to find the next value. Only when one index buffer is used up does the storage engine need to read the next index block. This makes [MERGE](#) indexes much slower on [eq_ref \[837\]](#) searches, but not much slower on [ref \[837\]](#) searches. For more information about [eq_ref \[837\]](#) and [ref \[837\]](#), see [Section 13.8.2, “EXPLAIN Syntax”](#).

14.10.2. MERGE Table Problems

The following are known problems with [MERGE](#) tables:

- In versions of MySQL Server prior to 5.1.23, it was possible to create temporary merge tables with nontemporary child [MyISAM](#) tables.

From versions 5.1.23, [MERGE](#) children were locked through the parent table. If the parent was temporary, it was not locked and so the children were not locked either. Parallel use of the [MyISAM](#) tables corrupted them.

- If you use [ALTER TABLE](#) to change a [MERGE](#) table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying [MyISAM](#) tables are copied into the altered table, which then uses the specified storage engine.
- The [INSERT_METHOD](#) table option for a [MERGE](#) table indicates which underlying [MyISAM](#) table to use for inserts into the [MERGE](#) table. However, use of the [AUTO_INCREMENT](#) table option for that [MyISAM](#) table has no effect for inserts into the [MERGE](#) table until at least one row has been inserted directly into the [MyISAM](#) table.
- A [MERGE](#) table cannot maintain uniqueness constraints over the entire table. When you perform an [INSERT](#), the data goes into the first or last [MyISAM](#) table (as determined by the [INSERT_METHOD](#) option). MySQL ensures that unique key values remain unique within that [MyISAM](#) table, but not over all the underlying tables in the collection.

- Because the `MERGE` engine cannot enforce uniqueness over the set of underlying tables, `REPLACE` does not work as expected. The two key facts are:
 - `REPLACE` can detect unique key violations only in the underlying table to which it is going to write (which is determined by the `INSERT_METHOD` option). This differs from violations in the `MERGE` table itself.
 - If `REPLACE` detects a unique key violation, it will change only the corresponding row in the underlying table it is writing to; that is, the first or last table, as determined by the `INSERT_METHOD` option.

Similar considerations apply for `INSERT ... ON DUPLICATE KEY UPDATE`.

- `MERGE` tables do not support partitioning. That is, you cannot partition a `MERGE` table, nor can any of a `MERGE` table's underlying `MyISAM` tables be partitioned.
- You should not use `ANALYZE TABLE`, `REPAIR TABLE`, `OPTIMIZE TABLE`, `ALTER TABLE`, `DROP TABLE`, `DELETE` without a `WHERE` clause, or `TRUNCATE TABLE` on any of the tables that are mapped into an open `MERGE` table. If you do so, the `MERGE` table may still refer to the original table and yield unexpected results. To work around this problem, ensure that no `MERGE` tables remain open by issuing a `FLUSH TABLES` statement prior to performing any of the named operations.

The unexpected results include the possibility that the operation on the `MERGE` table will report table corruption. If this occurs after one of the named operations on the underlying `MyISAM` tables, the corruption message is spurious. To deal with this, issue a `FLUSH TABLES` statement after modifying the `MyISAM` tables.

- `DROP TABLE` on a table that is in use by a `MERGE` table does not work on Windows because the `MERGE` storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not permit open files to be deleted, so you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table.
- The definition of the `MyISAM` tables and the `MERGE` table are checked when the tables are accessed (for example, as part of a `SELECT` or `INSERT` statement). The checks ensure that the definitions of the tables and the parent `MERGE` table definition match by comparing column order, types, sizes and associated indexes. If there is a difference between the tables, an error is returned and the statement fails. Because these checks take place when the tables are opened, any changes to the definition of a single table, including column changes, column ordering, and engine alterations will cause the statement to fail.
- The order of indexes in the `MERGE` table and its underlying tables should be the same. If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table, and then use `ALTER TABLE` to add a nonunique index on the `MERGE` table, the index ordering is different for the tables if there was already a nonunique index in the underlying table. (This happens because `ALTER TABLE` puts `UNIQUE` indexes before nonunique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.
- If you encounter an error message similar to `ERROR 1017 (HY000): Can't find file: 'tbl_name.MRG' (errno: 2)`, it generally indicates that some of the underlying tables do not use the `MyISAM` storage engine. Confirm that all of these tables are `MyISAM`.
- The maximum number of rows in a `MERGE` table is 2^{64} (~1.844E+19; the same as for a `MyISAM` table). It is not possible to merge multiple `MyISAM` tables into a single `MERGE` table that would have more than this number of rows.
- The `MERGE` storage engine does not support `INSERT DELAYED` statements.
- Use of underlying `MyISAM` tables of differing row formats with a parent `MERGE` table is currently known to fail. See Bug #32364.

- You cannot change the union list of a nontemporary `MERGE` table when `LOCK TABLES` is in effect. The following does *not* work:

```
CREATE TABLE m1 ... ENGINE=MRG_MYISAM ...;
LOCK TABLES t1 WRITE, t2 WRITE, m1 WRITE;
ALTER TABLE m1 ... UNION=(t1,t2) ...;
```

However, you can do this with a temporary `MERGE` table.

- You cannot create a `MERGE` table with `CREATE ... SELECT`, neither as a temporary `MERGE` table, nor as a nontemporary `MERGE` table. For example:

```
CREATE TABLE m1 ... ENGINE=MRG_MYISAM ... SELECT ...;
```

Attempts to do this result in an error: `tbl_name` is not `BASE TABLE`.

- In some cases, differing `PACK_KEYS` table option values among the `MERGE` and underlying tables cause unexpected results if the underlying tables contain `CHAR` or `BINARY` columns. As a workaround, use `ALTER TABLE` to ensure that all involved tables have the same `PACK_KEYS` value. (Bug #50646)

14.11. The `FEDERATED` Storage Engine

The `FEDERATED` storage engine lets you access data from a remote MySQL database without using replication or cluster technology. Querying a local `FEDERATED` table automatically pulls the data from the remote (federated) tables. No data is stored on the local tables.

To include the `FEDERATED` storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_FEDERATED_STORAGE_ENGINE` option.

The `FEDERATED` storage engine is not enabled by default in the running server; to enable `FEDERATED`, you must start the MySQL server binary using the `--federated` option.

To examine the source for the `FEDERATED` engine, look in the `storage/federated` directory of a MySQL source distribution.

14.11.1. `FEDERATED` Storage Engine Overview

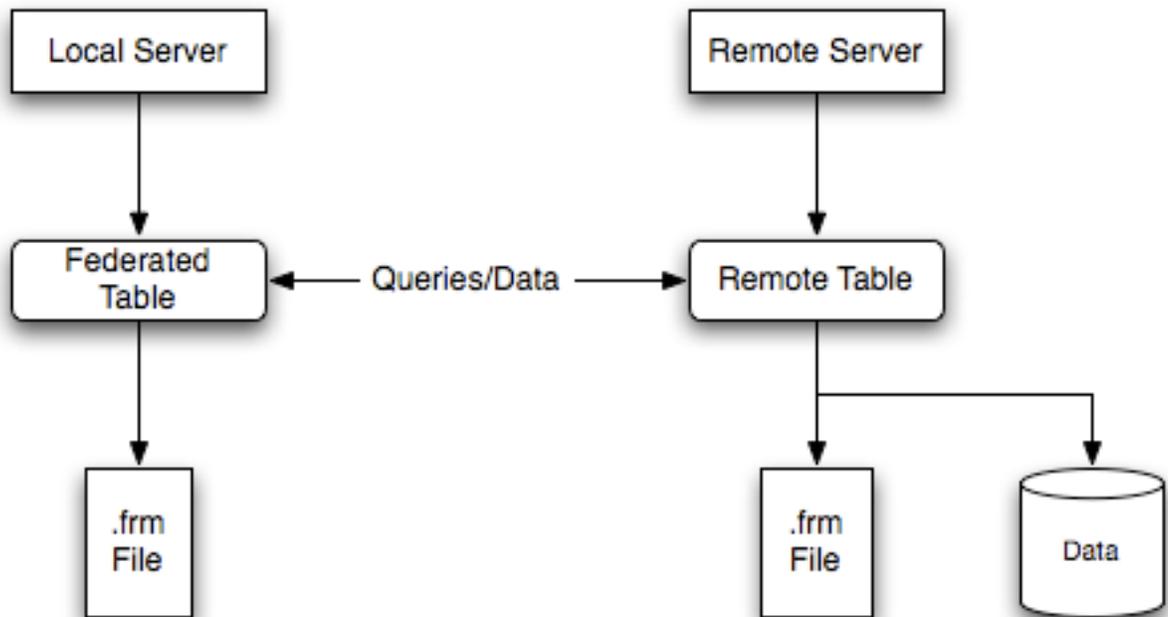
When you create a table using one of the standard storage engines (such as `MyISAM`, `CSV` or `InnoDB`), the table consists of the table definition and the associated data. When you create a `FEDERATED` table, the table definition is the same, but the physical storage of the data is handled on a remote server.

A `FEDERATED` table consists of two elements:

- A *remote server* with a database table, which in turn consists of the table definition (stored in the `.frm` file) and the associated table. The table type of the remote table may be any type supported by the remote `mysqld` server, including `MyISAM` or `InnoDB`.
- A *local server* with a database table, where the table definition matches that of the corresponding table on the remote server. The table definition is stored within the `.frm` file. However, there is no data file on the local server. Instead, the table definition includes a connection string that points to the remote table.

When executing queries and statements on a `FEDERATED` table on the local server, the operations that would normally insert, update or delete information from a local data file are instead sent to the remote server for execution, where they update the data file on the remote server or return matching rows from the remote server.

The basic structure of a `FEDERATED` table setup is shown in [Figure 14.1, “FEDERATED Table Structure”](#).

Figure 14.1. FEDERATED Table Structure

When a client issues an SQL statement that refers to a **FEDERATED** table, the flow of information between the local server (where the SQL statement is executed) and the remote server (where the data is physically stored) is as follows:

1. The storage engine looks through each column that the **FEDERATED** table has and constructs an appropriate SQL statement that refers to the remote table.
2. The statement is sent to the remote server using the MySQL client API.
3. The remote server processes the statement and the local server retrieves any result that the statement produces (an affected-rows count or a result set).
4. If the statement produces a result set, each column is converted to internal storage engine format that the **FEDERATED** engine expects and can use to display the result to the client that issued the original statement.

The local server communicates with the remote server using MySQL client C API functions. It invokes `mysql_real_query()` to send the statement. To read a result set, it uses `mysql_store_result()` and fetches rows one at a time using `mysql_fetch_row()`.

14.11.2. How to Create **FEDERATED** Tables

To create a **FEDERATED** table you should follow these steps:

1. Create the table on the remote server. Alternatively, make a note of the table definition of an existing table, perhaps using the `SHOW CREATE TABLE` statement.
2. Create the table on the local server with an identical table definition, but adding the connection information that links the local table to the remote table.

For example, you could create the following table on the remote server:

```
CREATE TABLE test_table (  
  id      INT(20) NOT NULL AUTO_INCREMENT,  
  name    VARCHAR(32) NOT NULL DEFAULT '',  
  other   INT(20) NOT NULL DEFAULT '0',  
  PRIMARY KEY (id),  
  INDEX name (name),  
  INDEX other_key (other)  
)  
ENGINE=MyISAM  
DEFAULT CHARSET=latin1;
```

To create the local table that will be federated to the remote table, there are two options available. You can either create the local table and specify the connection string (containing the server name, login, password) to be used to connect to the remote table using the `CONNECTION`, or you can use an existing connection that you have previously created using the `CREATE SERVER` statement.



Important

When you create the local table it *must* have an identical field definition to the remote table.



Note

You can improve the performance of a `FEDERATED` table by adding indexes to the table on the host. The optimization will occur because the query sent to the remote server will include the contents of the `WHERE` clause and will be sent to the remote server and subsequently executed locally. This reduces the network traffic that would otherwise request the entire table from the server for local processing.

14.11.2.1. Creating a `FEDERATED` Table Using `CONNECTION`

To use the first method, you must specify the `CONNECTION` string after the engine type in a `CREATE TABLE` statement. For example:

```
CREATE TABLE federated_table (  
  id      INT(20) NOT NULL AUTO_INCREMENT,  
  name    VARCHAR(32) NOT NULL DEFAULT '',  
  other   INT(20) NOT NULL DEFAULT '0',  
  PRIMARY KEY (id),  
  INDEX name (name),  
  INDEX other_key (other)  
)  
ENGINE=FEDERATED  
DEFAULT CHARSET=latin1  
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```



Note

`CONNECTION` replaces the `COMMENT` used in some previous versions of MySQL.

The `CONNECTION` string contains the information required to connect to the remote server containing the table that will be used to physically store the data. The connection string specifies the server name, login credentials, port number and database/table information. In the example, the remote table is on the server `remote_host`, using port 9306. The name and port number should match the host name (or IP address) and port number of the remote MySQL server instance you want to use as your remote table.

The format of the connection string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Where:

- *scheme*: A recognized connection protocol. Only `mysql` is supported as the *scheme* value at this point.
- *user_name*: The user name for the connection. This user must have been created on the remote server, and must have suitable privileges to perform the required actions (`SELECT`, `INSERT`, `UPDATE`, and so forth) on the remote table.
- *password*: (Optional) The corresponding password for *user_name*.
- *host_name*: The host name or IP address of the remote server.
- *port_num*: (Optional) The port number for the remote server. The default is 3306.
- *db_name*: The name of the database holding the remote table.
- *tbl_name*: The name of the remote table. The name of the local and the remote table do not have to match.

Sample connection strings:

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'  
CONNECTION='mysql://username@hostname/database/tablename'  
CONNECTION='mysql://username:password@hostname/database/tablename'
```

14.11.2.2. Creating a **FEDERATED** Table Using **CREATE SERVER**

If you are creating a number of **FEDERATED** tables on the same server, or if you want to simplify the process of creating **FEDERATED** tables, you can use the **CREATE SERVER** statement to define the server connection parameters, just as you would with the **CONNECTION** string.

The format of the **CREATE SERVER** statement is:

```
CREATE SERVER server_name  
FOREIGN DATA WRAPPER wrapper_name  
OPTIONS (option [, option] ...)
```

The *server_name* is used in the connection string when creating a new **FEDERATED** table.

For example, to create a server connection identical to the **CONNECTION** string:

```
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

You would use the following statement:

```
CREATE SERVER fedlink  
FOREIGN DATA WRAPPER mysql  
OPTIONS (USER 'fed_user', HOST 'remote_host', PORT 9306, DATABASE 'federated');
```

To create a **FEDERATED** table that uses this connection, you still use the **CONNECTION** keyword, but specify the name you used in the **CREATE SERVER** statement.

```
CREATE TABLE test_table (  
  id      INT(20) NOT NULL AUTO_INCREMENT,  
  name    VARCHAR(32) NOT NULL DEFAULT '',  
  other   INT(20) NOT NULL DEFAULT '0',  
  PRIMARY KEY (id),  
  INDEX name (name),  
  INDEX other_key (other)  
)
```

```
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='fedlink/test_table';
```

The connection name in this example contains the name of the connection (`fedlink`) and the name of the table (`test_table`) to link to, separated by a slash. If you specify only the connection name without a table name, the table name of the local table is used instead.

For more information on `CREATE SERVER`, see [Section 13.1.16, “CREATE SERVER Syntax”](#).

The `CREATE SERVER` statement accepts the same arguments as the `CONNECTION` string. The `CREATE SERVER` statement updates the rows in the `mysql.servers` table. See the following table for information on the correspondence between parameters in a connection string, options in the `CREATE SERVER` statement, and the columns in the `mysql.servers` table. For reference, the format of the `CONNECTION` string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Description	CONNECTION string	CREATE SERVER option	mysql.servers column
Connection scheme	<code>scheme</code>	<code>wrapper_name</code>	Wrapper
Remote user	<code>user_name</code>	USER	Username
Remote password	<code>password</code>	PASSWORD	Password
Remote host	<code>host_name</code>	HOST	Host
Remote port	<code>port_num</code>	PORT	Port
Remote database	<code>db_name</code>	DATABASE	Db

14.11.3. FEDERATED Storage Engine Notes and Tips

You should be aware of the following points when using the `FEDERATED` storage engine:

- `FEDERATED` tables may be replicated to other slaves, but you must ensure that the slave servers are able to use the user/password combination that is defined in the `CONNECTION` string (or the row in the `mysql.servers` table) to connect to the remote server.

The following items indicate features that the `FEDERATED` storage engine does and does not support:

- The remote server must be a MySQL server.
- The remote table that a `FEDERATED` table points to *must* exist before you try to access the table through the `FEDERATED` table.
- It is possible for one `FEDERATED` table to point to another, but you must be careful not to create a loop.
- A `FEDERATED` table does not support indexes per se. Because access to the table is handled remotely, it is the remote table that supports the indexes. Care should be taken when creating a `FEDERATED` table since the index definition from an equivalent `MyISAM` or other table may not be supported. For example, creating a `FEDERATED` table with an index prefix on `VARCHAR`, `TEXT` or `BLOB` columns will fail. The following definition in `MyISAM` is valid:

```
CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=MYISAM;
```

The key prefix in this example is incompatible with the `FEDERATED` engine, and the equivalent statement will fail:

```
CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=FEDERATED
CONNECTION='MYSQL://127.0.0.1:3306/TEST/T1';
```

If possible, you should try to separate the column and index definition when creating tables on both the remote server and the local server to avoid these index issues.

- Internally, the implementation uses `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but not `HANDLER`.
- The `FEDERATED` storage engine supports `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE TABLE`, and indexes. It does not support `ALTER TABLE`, or any Data Definition Language statements that directly affect the structure of the table, other than `DROP TABLE`. The current implementation does not use prepared statements.
- `FEDERATED` accepts `INSERT ... ON DUPLICATE KEY UPDATE` statements, but if a duplicate-key violation occurs, the statement fails with an error.
- Performance on a `FEDERATED` table when performing bulk inserts (for example, on a `INSERT INTO ... SELECT ...` statement) is slower than with other table types because each selected row is treated as an individual `INSERT` statement on the `FEDERATED` table.
- Transactions are not supported.
- `FEDERATED` performs bulk-insert handling such that multiple rows are sent to the remote table in a batch. This provides a performance improvement and enables the remote table to perform improvement. Also, if the remote table is transactional, it enables the remote storage engine to perform statement rollback properly should an error occur. This capability has the following limitations:
 - The size of the insert cannot exceed the maximum packet size between servers. If the insert exceeds this size, it is broken into multiple packets and the rollback problem can occur.
 - Bulk-insert handling does not occur for `INSERT ... ON DUPLICATE KEY UPDATE`.
- There is no way for the `FEDERATED` engine to know if the remote table has changed. The reason for this is that this table must work like a data file that would never be written to by anything other than the database system. The integrity of the data in the local table could be breached if there was any change to the remote database.
- When using a `CONNECTION` string, you cannot use an '@' character in the password. You can get round this limitation by using the `CREATE SERVER` statement to create a server connection.
- The `insert_id` [506] and `timestamp` [567] options are not propagated to the data provider.
- Any `DROP TABLE` statement issued against a `FEDERATED` table drops only the local table, not the remote table.
- `FEDERATED` tables do not work with the query cache.
- User-defined partitioning is not supported for `FEDERATED` tables.

14.11.4. FEDERATED Storage Engine Resources

The following additional resources are available for the `FEDERATED` storage engine:

- A forum dedicated to the `FEDERATED` storage engine is available at <http://forums.mysql.com/list.php?105>.

14.12. The EXAMPLE Storage Engine

The `EXAMPLE` storage engine is a stub engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

To enable the `EXAMPLE` storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_EXAMPLE_STORAGE_ENGINE` option.

To examine the source for the `EXAMPLE` engine, look in the `storage/example` directory of a MySQL source distribution.

When you create an `EXAMPLE` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. No other files are created. No data can be stored into the table. Retrievals return an empty result.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't »
      have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

The `EXAMPLE` storage engine does not support indexing.

14.13. Other Storage Engines

Other storage engines may be available from third parties and community members that have used the Custom Storage Engine interface.



Note

Third party engines are not supported by MySQL. For further information, documentation, installation guides, bug reporting or for any help or assistance with these engines, please contact the developer of the engine directly.

Third party engines that are known to be available include the following:

- **PrimeBase XT (PBXT):** PBXT has been designed for modern, web-based, high concurrency environments.
- **RitmarkFS:** RitmarkFS enables you to access and manipulate the file system using SQL queries. RitmarkFS also supports file system replication and directory change tracking.
- **Distributed Data Engine:** The Distributed Data Engine is an Open Source project that is dedicated to provide a Storage Engine for distributed data according to workload statistics.
- **mdbtools:** A pluggable storage engine that enables read-only access to Microsoft Access `.mdb` database files.
- **solidDB for MySQL:** solidDB Storage Engine for MySQL is an open source, transactional storage engine for MySQL Server. It is designed for mission-critical implementations that require a robust, transactional database. solidDB Storage Engine for MySQL is a multi-threaded storage engine that supports full ACID compliance with all expected transaction isolation levels, row-level locking, and Multi-Version Concurrency Control (MVCC) with nonblocking reads and writes.

For more information on developing a customer storage engine that can be used with the Pluggable Storage Engine Architecture, see [MySQL Internals: Writing a Custom Storage Engine](#).

Chapter 15. High Availability and Scalability

Table of Contents

15.1. Oracle VM Template for MySQL Enterprise Edition	1724
15.2. Overview of MySQL with DRBD/Pacemaker/Corosync/Oracle Linux	1725
15.3. Overview of MySQL with Windows Failover Clustering	1727
15.4. Using MySQL within an Amazon EC2 Instance	1729
15.4.1. Setting Up MySQL on an EC2 AMI	1730
15.4.2. EC2 Instance Limitations	1731
15.4.3. Deploying a MySQL Database Using EC2	1731
15.5. Using ZFS Replication	1734
15.5.1. Using ZFS for File System Replication	1736
15.5.2. Configuring MySQL for ZFS Replication	1736
15.5.3. Handling MySQL Recovery with ZFS	1737
15.6. Using MySQL with memcached	1737
15.6.1. Installing memcached	1739
15.6.2. Using memcached	1740
15.6.3. Developing a memcached Application	1760
15.6.4. Getting memcached Statistics	1787
15.6.5. memcached FAQ	1796
15.7. MySQL Proxy	1799
15.7.1. MySQL Proxy Supported Platforms	1800
15.7.2. Installing MySQL Proxy	1800
15.7.3. MySQL Proxy Command Options	1803
15.7.4. MySQL Proxy Scripting	1813
15.7.5. Using MySQL Proxy	1828
15.7.6. MySQL Proxy FAQ	1833

MySQL is deployed into many applications demanding availability and scalability.

Availability refers to the ability to cope with, and if necessary recover from, failures on the host, including failures of MySQL, the operating system, or the hardware and maintenance activity that may otherwise cause downtime. Scalability refers to the ability to spread both the database and the load of your application queries across multiple MySQL servers.

Because each application has different operational and availability requirements, MySQL offers a range of certified and supported solutions, delivering the appropriate levels of High Availability (HA) and scalability to meet service level requirements. Such solutions extend from replication, through virtualization and geographically redundant, multi-data center solutions delivering 99.999% uptime.

Selecting the right high availability solution for an application largely depends on:

- The level of availability required.
- The type of application being deployed.
- Accepted best practices within your own environment.

The primary solutions supported by MySQL include:

- MySQL Replication. Learn more: [Chapter 16, Replication](#)

- MySQL Cluster. Learn more: [Chapter 17, MySQL Cluster NDB 7.2](#)
- Oracle VM Template for MySQL. Learn more: [Section 15.1, “Oracle VM Template for MySQL Enterprise Edition”](#).

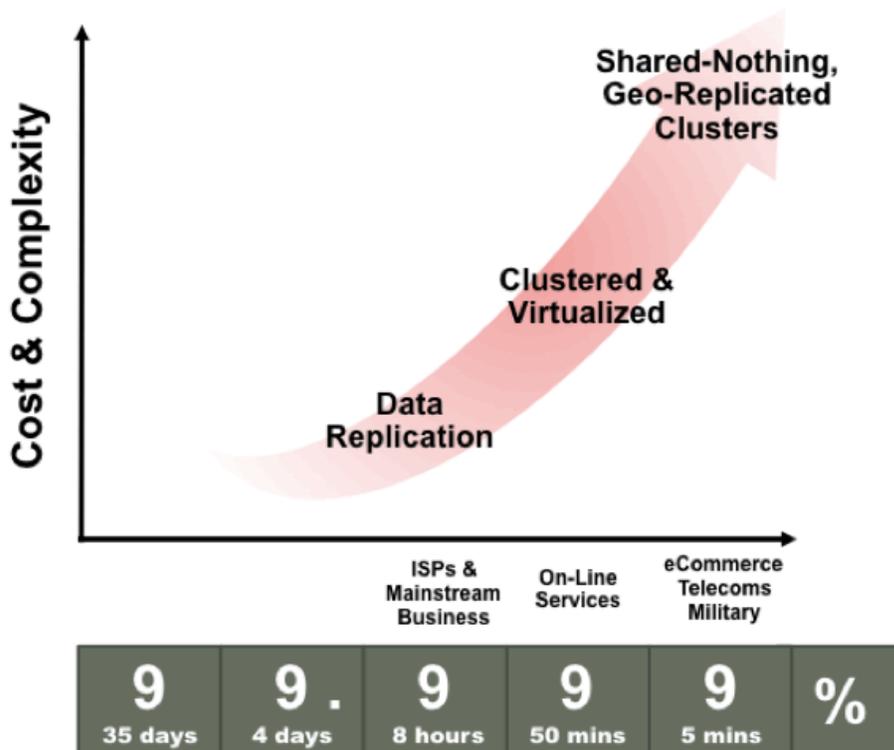
Further options are available using third-party solutions such as DRBD (Distributed Replicated Block Device) and Heartbeat, and more complex scenarios can be solved through a combination of these technologies.

Each architecture used to achieve highly available database services is differentiated by the levels of uptime it offers. These architectures can be grouped into three main categories:

- Data Replication.
- Clustered & Virtualized Systems.
- Shared-Nothing, Geographically-Replicated Clusters.

As illustrated in the following figure, each of these architectures offers progressively higher levels of uptime, which must be balanced against potentially greater levels of cost and complexity that each can incur. Simply deploying a high availability architecture is not a guarantee of actually delivering HA. In fact, a poorly implemented and maintained shared-nothing cluster could easily deliver lower levels of availability than a simple data replication solution.

Figure 15.1. Tradeoffs: Cost and Complexity versus Availability



The following figure maps common application types to architectures, based on best practices observed from the MySQL user base. It serves as a reference point to investigate which HA architectures can best serve your requirements.

Figure 15.2. High Availability Architectures for Common Application Types

Applications	Data Replication	Clustered / Virtualized	Shared-Nothing, Geo-Replicated Cluster
E-Commerce / Trading		⊙	⊙
Session Management		⊙	⊙
User Authentication / Accounting		⊙	⊙
Feeds, Blogs, Wikis	⊙	⊙	
Websites	⊙	⊙	
OLTP		⊙	⊙
Data Warehouse/BI	⊙	⊙	
Content Management	⊙	⊙	
CRM		⊙	
Collaboration	⊙	⊙	
Packaged Software		⊙	
Telco Apps (HLR/HSS/SDP...)			⊙

The following table compares the HA and Scalability capabilities of the various MySQL solutions:

Table 15.1. Feature Comparison of MySQL HA Solutions

Requirement	MySQL Replication	MySQL Replication + Linux Heartbeat	Heartbeat + DRBD	Oracle VM Template	MySQL Cluster
Availability					
Platform Support	All Supported by MySQL Server	Linux	Linux	Oracle Linux	All Supported by MySQL Cluster
Automated IP Failover	No	Yes	Yes	Yes	Depends on Connector and Configuration
Automated Database Failover	No	No	Yes	Yes	Yes
Automatic Data Resynchronization	No	No	Yes	N/A - Shared Storage	Yes
Typical Failover Time	User / Script Dependent	Configuration Dependent, 60 seconds and Above	Configuration Dependent, 60 seconds and Above	Configuration Dependent, 60 seconds and Above	1 Second and Less
Synchronous Replication	No, Asynchronous and Semisynchronous	No, Asynchronous and Semisynchronous	Yes	N/A - Shared Storage	Yes

Requirement	MySQL Replication	MySQL Replication + Linux Heartbeat	Heartbeat + DRBD	Oracle VM Template	MySQL Cluster
Shared Storage	No, Distributed	No, Distributed	No, Distributed	Yes	No, Distributed
Geographic redundancy support	Yes	Yes	Yes, via MySQL Replication	Yes, via MySQL Replication	Yes, via MySQL Replication
Update Schema On-Line	No	No	No	No	Yes
Scalability					
Number of Nodes	One Master, Multiple Slaves	One Master, Multiple Slaves	One Active (primary), one Passive (secondary) Node	One Active (primary), one Passive (secondary) Node	255
Built-in Load Balancing	Reads, via MySQL Replication	Reads, via MySQL Replication	Reads, via MySQL Replication	Reads, via MySQL Replication & During Failover	Yes, Reads and Writes
Supports Read-Intensive Workloads	Yes	Yes	Yes	Yes	Yes
Supports Write-Intensive Workloads	Yes, via Application-Level Sharding	Yes, via Application-Level Sharding	Yes, via Application-Level Sharding to Multiple Active/Passive Pairs	Yes, via Application-Level Sharding to Multiple Active/Passive Pairs	Yes, via Auto-Sharding
Scale On-Line (add nodes, repartition, etc.)	No	No	No	No	Yes

15.1. Oracle VM Template for MySQL Enterprise Edition

Virtualization is a key technology to enable data center efficiency and high availability while providing the foundation for cloud computing. Integrating MySQL Enterprise Edition with Oracle Linux, the Oracle VM Template is the fastest, easiest, and most reliable way to provision virtualized MySQL instances, enabling users to meet the explosive demand for highly available services.

The Oracle VM Template enables rapid deployment and eliminates manual configuration efforts. It provides a pre-installed and pre-configured virtualized MySQL 5.5 Enterprise Edition software image running on Oracle Linux and Oracle VM, certified for production use. The MySQL software image has undergone extensive integration and quality assurance testing as part of the development process.

In addition to rapid provisioning, MySQL users also benefit from the integrated high availability features of Oracle VM which are designed to enable organizations to meet stringent SLA (Service Level Agreement) demands through a combination of:

- **Automatic recovery from failures**, with Oracle VM automatically restarting failed instances on available servers in the server pool after outages of the physical server, VM or MySQL database.

- **Live Migration**, enabling operations staff to move running instances of MySQL to alternative hosts within a server pool during maintenance operations.

Instructions for the creation, deployment and use of the Oracle VM Template for MySQL Enterprise Edition are available from:

- The Oracle VM Template for MySQL Enterprise Edition whitepaper: http://www.mysql.com/why-mysql/white-papers/mysql_wp_oracle-vm-template-for-mee.php.
- The README file accompanying the download of the Template.

To download the Oracle VM Template for MySQL Enterprise, go to <http://edelivery.oracle.com/oraclevm> and follow these instructions:

- Complete your registration information (Name, Company Name, Email Address and Country) and click on the download agreement.
- Select "Oracle VM Templates" from the "Select a Product Pack" pull-down menu and click "Go".
- Select MySQL Enterprise from the list of Oracle VM Templates.
- Download and unzip the files and refer to the README for further instructions.

15.2. Overview of MySQL with DRBD/Pacemaker/Corosync/Oracle Linux

DRBD (Distributed Replication Block Device) is one of the leading solutions for MySQL HA (High Availability). When combined with Pacemaker and Corosync, users have:

- An end-to-end, integrated stack of mature and proven open source technologies, fully supported by Oracle (as part of MySQL Enterprise Edition).
- Automatic failover and recovery for service continuity.
- Mirroring, via synchronous replication, to ensure failover between nodes without the risk of losing committed transactions.
- Building of HA clusters from commodity hardware, without the requirement for shared-storage.

The following figure illustrates the stack that can be used to deliver a level of High Availability for the MySQL service.

At the lowest level, 2 hosts are required in order to provide physical redundancy; if using a virtual environment, those 2 hosts should be on different physical machines. It is an important feature that no shared storage is required. At any point in time, the services will be active on one host and in standby mode on the other.

Pacemaker and Corosync combine to provide the clustering layer that sits between the services and the underlying hosts and operating systems. Pacemaker is responsible for starting and stopping services, ensuring that they are running on exactly one host, thus delivering high availability and avoiding data corruption. Corosync provides the underlying messaging infrastructure between the nodes that enables Pacemaker to do its job; it also handles the nodes membership within the cluster and informs Pacemaker of any changes.

Figure 15.3. MySQL, DRBD, Pacemaker, and Corosync Stack

The core Pacemaker process does not have built-in knowledge of the specific services to be managed; instead, it uses agents that provide a wrapper for the service-specific actions. For example, in this solution

we use agents for Virtual IP Addresses, MySQL and DRBD: these are all existing agents and come packaged with Pacemaker.

The essential services managed by Pacemaker in this configuration are DRBD, MySQL and the Virtual IP Address that applications use to connect to the active MySQL service.

DRBD synchronizes data at the block device (typically a spinning or solid state disk) – transparent to the application, database and even the file system. DRBD requires the use of a journaling file system such as `ext3` or `ext4`. For this solution, it acts in an active-standby mode: at any point in time, the directories being managed by DRBD are accessible for reads and writes on exactly one of the two hosts and inaccessible (even for reads) on the other. Any changes made on the active host are synchronously replicated to the standby host by DRBD.

Download the following guide for detailed instructions on installing, configuring, provisioning and testing the complete MySQL and DRBD stack, including:

- MySQL Database.
- DRBD kernel module and userland utilities.
- Pacemaker and Corosync cluster messaging and management processes.
- Oracle Linux operating system.

Download the guide at: http://www.mysql.com/why-mysql/white-papers/mysql_wp_drbd.php.

Support for DRBD

The complete DRBD stack for MySQL has been certified by Oracle, and commercial support is available as part of [MySQL Enterprise Edition](#) and [Oracle Linux Premier Support](#), providing a single point of contact for the entire stack, whether issues relate to the operating system, DRBD, clustering software or MySQL.

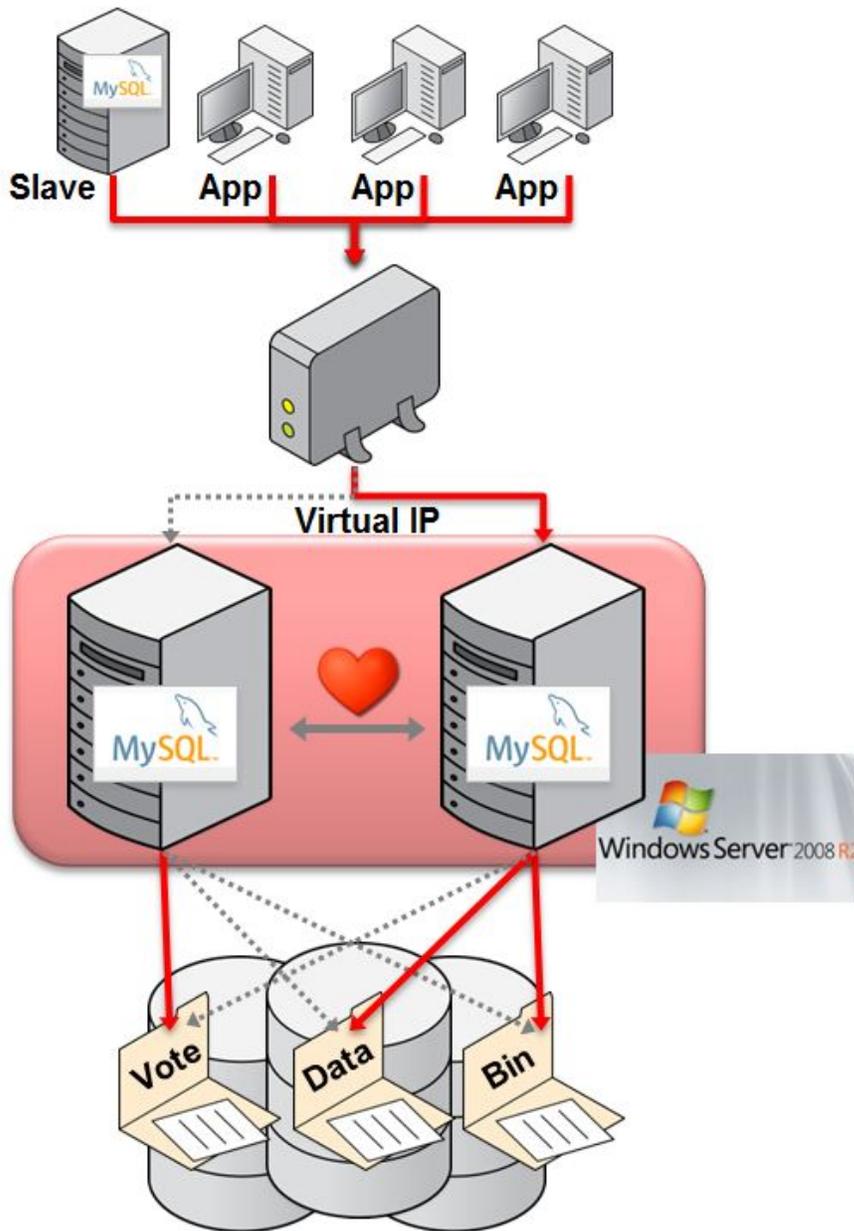
15.3. Overview of MySQL with Windows Failover Clustering

Microsoft Windows is consistently ranked as the top development platform for MySQL, based on surveys of the MySQL user community.

MySQL Enterprise Edition is certified and supported with Windows Server 2008 R2 Failover Clustering (WSFC), enabling organizations to safely deploy business-critical applications demanding high levels of availability using Microsoft's native Windows clustering services.

The following figure illustrates the integration of MySQL with Windows Server Failover Clustering to provide a highly available service:

Figure 15.4. Typical MySQL HA Configuration with Windows Server Failover Clustering



In this architecture, MySQL is deployed in an Active / Passive configuration. Failures of either MySQL or the underlying server are automatically detected and the MySQL instance is restarted on the Passive node. Applications accessing the database, as well as any MySQL replication slaves, can automatically reconnect to the new MySQL process using the same Virtual IP address once MySQL recovery has completed and it starts accepting connections.

MySQL with Windows Failover Clustering requires at least 2 servers within the cluster together with shared storage (for example, FC-AL SAN or iSCSI disks).

The MySQL binaries and data files are stored in the shared storage and Windows Failover Clustering ensures that only one of the cluster nodes will access those files at any point in time.

Clients connect to the MySQL service through a Virtual IP Address (VIP). In the event of failover they experience a brief loss of connection, but otherwise do not need to be aware that the failover has happened, other than to handle the failure of any transactions that were active when the failover occurred.

You can learn more about configuring MySQL with Windows Server Failover Clustering from the whitepaper posted here: http://www.mysql.com/why-mysql/white-papers/mysql_wp_windows_failover_clustering.php

For background and usage information about Windows Server Failover Clustering, see these pages on the Microsoft Technet site:

- [Failover Clustering](#)
- [Failover Clusters in Windows Server 2008 R2](#)

15.4. Using MySQL within an Amazon EC2 Instance

The Amazon Elastic Compute Cloud (EC2) service provides virtual servers that you can build and deploy to run a variety of different applications and services, including MySQL. The EC2 service is based around the Xen framework, supporting x86, Linux based, platforms with individual instances of a virtual machine referred to as an Amazon Machine Image (AMI). You have complete (root) access to the AMI instance that you create, enabling you to configure and install your AMI in any way you choose.

To use EC2, you create an AMI based on the configuration and applications that you intend to use, and upload the AMI to the Amazon Simple Storage Service (S3). From the S3 resource, you can deploy one or more copies of the AMI to run as an instance within the EC2 environment. The EC2 environment provides management and control of the instance and contextual information about the instance while it is running.

Because you can create and control the AMI, the configuration, and the applications, you can deploy and create any environment you choose. This includes a basic MySQL server in addition to more extensive replication, HA and scalability scenarios that enable you to take advantage of the EC2 environment, and the ability to deploy additional instances as the demand for your MySQL services and applications grow.

To aid the deployment and distribution of work, three different Amazon EC2 instances are available, small (identified as `m1.small`), large (`m1.large`) and extra large (`m1.xlarge`). The different types provide different levels of computing power measured in EC2 computer units (ECU). A summary of the different instance configurations is shown here.

	Small	Large	Extra Large
Platform	32-bit	64-bit	64-bit
CPU cores	1	2	4
ECUs	1	4	8
RAM	1.7GB	7.5GB	15GB
Storage	150GB	840GB	1680GB
I/O Performance	Medium	High	High

The typical model for deploying and using MySQL within the EC2 environment is to create a basic AMI that you can use to hold your database data and application. Once the basic environment for your database and application has been created you can then choose to deploy the AMI to a suitable instance. Here the flexibility of having an AMI that can be re-deployed from the small to the large or extra large EC2 instance makes it easy to upgrade the hardware environment without rebuilding your application or database stack.

To get started with MySQL on EC2, including information on how to set up and install MySQL within an EC2 installation and how to port and migrate your data to the running instance, see [Section 15.4.1, “Setting Up MySQL on an EC2 AMI”](#).

For tips and advice on how to create a scalable EC2 environment using MySQL, including guides on setting up replication, see [Section 15.4.3, “Deploying a MySQL Database Using EC2”](#).

15.4.1. Setting Up MySQL on an EC2 AMI

There are many different ways of setting up an EC2 AMI with MySQL, including using any of the pre-configured AMIs supplied by Amazon.

The default *Getting Started* AMI provided by Amazon uses Fedora Core 4, and you can install MySQL by using `yum`:

```
shell> yum install mysql
```

This installs both the MySQL server and the Perl DBD::mysql driver for the Perl DBI API.

Alternatively, you can use one of the AMIs that include MySQL within the standard installation.

Finally, you can also install a standard version of MySQL downloaded from the MySQL Web site. The installation process and instructions are identical to any other installation of MySQL on Linux. See [Chapter 2, *Installing and Upgrading MySQL*](#).

The standard configuration for MySQL places the data files in the default location, `/var/lib/mysql`. The default data directory on an EC2 instance is `/mnt` (although on the large and extra large instance you can alter this configuration). You must edit `/etc/my.cnf` to set the `datadir` [439] option to point to the larger storage area.



Important

The first time you use the main storage location within an EC2 instance it needs to be initialized. The initialization process starts automatically the first time you write to the device. You can start using the device right away, but the write performance of the new device is significantly lower on the initial writes until the initialization process has finished.

To avoid this problem when setting up a new instance, you should start the initialization process before populating your MySQL database. One way to do this is to use `dd` to write to the file system:

```
root-shell> dd if=/dev/zero of=initialize bs=1024M count=50
```

The preceding creates a 50GB on the file system and starts the initialization process. Delete the file once the process has finished.

The initialization process can be time-consuming. On the small instance, initialization takes between two and three hours. For the large and extra large drives, the initialization can be 10 or 20 hours, respectively.

In addition to configuring the correct storage location for your MySQL data files, also consider setting the following other settings in your instance before you save the instance configuration for deployment:

- Set the MySQL server ID, so that when you use it for replication, the ID information is set correctly.
- Enabling binary logging, so that replication can be initialized without starting and stopping the server.

- Set the caching and memory parameters for your storage engines. There are no limitations or restrictions on what storage engines you use in your EC2 environment. Choose a configuration, possibly using one of the standard configurations provided with MySQL appropriate for the instance on which you expect to deploy. The large and extra large instances have RAM that can be dedicated to caching. Be aware that if you choose to install [memcached](#) on the servers as part of your application stack you must ensure there is enough memory for both MySQL and [memcached](#).

Once you have configured your AMI with MySQL and the rest of your application stack, save the AMI so that you can deploy and reuse the instance.

Once you have your application stack configured in an AMI, populating your MySQL database with data should be performed by creating a dump of your database using [mysqldump](#), transferring the dump to the EC2 instance, and then reloading the information into the EC2 instance database.

Before using your instance with your application in a production situation, be aware of the limitations of the EC2 instance environment. See [Section 15.4.2, “EC2 Instance Limitations”](#). To begin using your MySQL AMI, consult the notes on deployment. See [Section 15.4.3, “Deploying a MySQL Database Using EC2”](#).

15.4.2. EC2 Instance Limitations

Be aware of the following limitations of the EC2 instances before deploying your applications. Although these shouldn't affect your ability to deploy within the Amazon EC2 environment, they may alter the way you setup and configure your environment to support your application.

- Data stored within instances is not persistent. If you create an instance and populate the instance with data, then the data only remains in place while the machine is running, and does not survive a reboot. If you shut down the instance, any data it contained is lost.

To ensure that you do not lose information, take regular backups using [mysqldump](#). If the data being stored is critical, consider using replication to keep a “live” backup of your data in the event of a failure. When creating a backup, write the data to the Amazon S3 service to avoid the transfer charges applied when copying data offsite.

- EC2 instances are not persistent. If the hardware on which an instance is running fails, the instance is shut down. This can lead to loss of data or service.
- To replicate your EC2 instances to a non-EC2 environment, be aware of the transfer costs to and from the EC2 service. Data transfer between different EC2 instances is free, so using replication within the EC2 environment does not incur additional charges.
- Certain HA features are either not directly supported, or have limiting factors or problems that could reduce their utility. For example, using DRBD or MySQL Cluster might not work. The default storage configuration is also not redundant. You can use software-based RAID to improve redundancy, but this implies a further performance hit.

15.4.3. Deploying a MySQL Database Using EC2

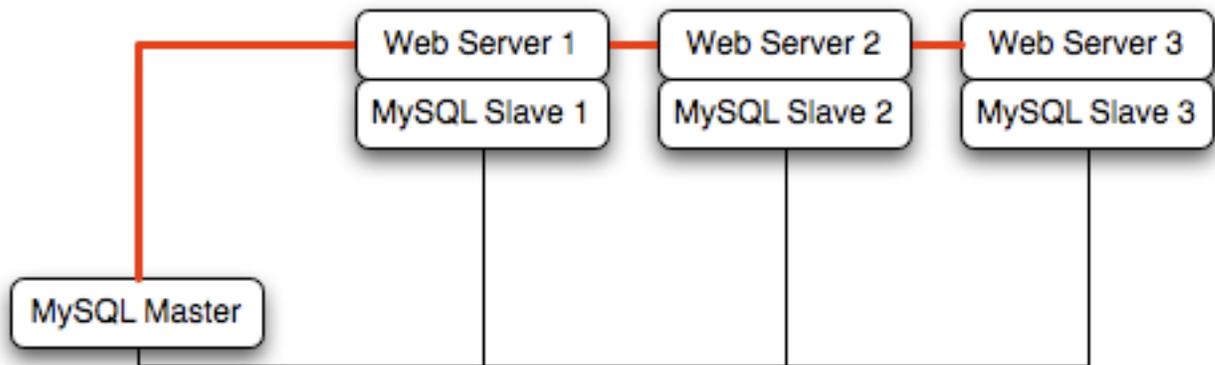
Because you cannot guarantee the uptime and availability of your EC2 instances, when deploying MySQL within the EC2 environment, use an approach that enables you to easily distribute work among your EC2 instances. There are a number of ways of doing this. Using sharding techniques, where you split the application across multiple servers dedicating specific blocks of your dataset and users to different servers is an effective way of doing this. As a general rule, it is easier to create more EC2 instances to support more users than to upgrade the instance to a larger machine.

The EC2 architecture works best when you treat the EC2 instances as temporary, cache-based solutions, rather than as a long-term, high availability solution. In addition to using multiple machines, take advantage

of other services, such as [memcached](#) to provide additional caching for your application to help reduce the load on the MySQL server so that it can concentrate on writes. On the large and extra large instances within EC2, the RAM available can provide a large memory cache for data.

Most types of scale-out topology that you would use with your own hardware can be used and applied within the EC2 environment. However, use the limitations and advice already given to ensure that any potential failures do not lose you any data. Also, because the relative power of each EC2 instance is so low, be prepared to alter your application to use sharding and add further EC2 instances to improve the performance of your application.

For example, take the typical scale-out environment shown following, where a single master replicates to one or more slaves (three in this example), with a web server running on each replication slave.



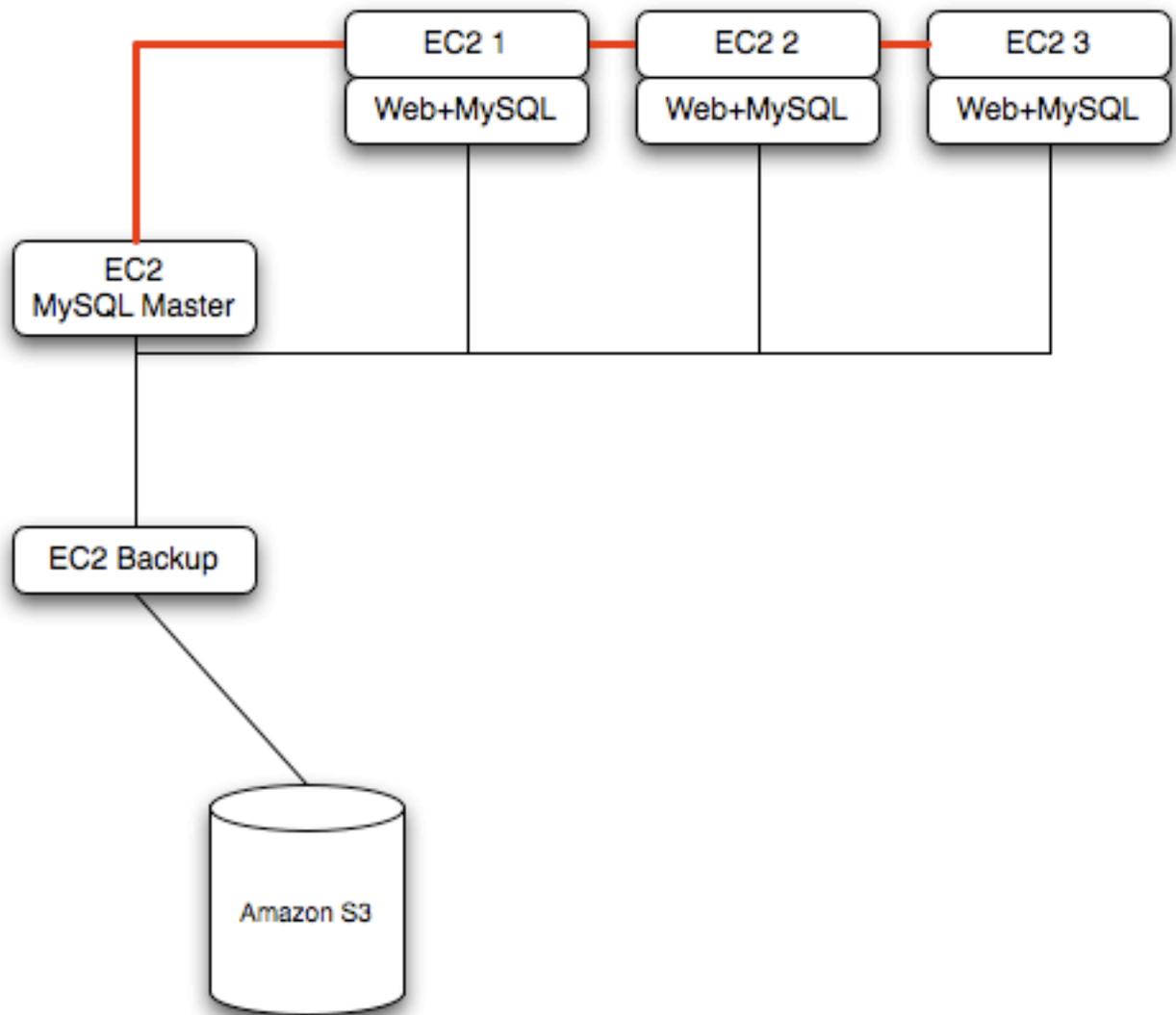
You can reproduce this structure completely within the EC2 environment, using an EC2 instance for the master, and one instance for each of the web and MySQL slave servers.



Note

Within the EC2 environment, internal (private) IP addresses used by the EC2 instances are constant. Always use these internal addresses and names when communicating between instances. Only use public IP addresses when communicating with the outside world - for example, when publicizing your application.

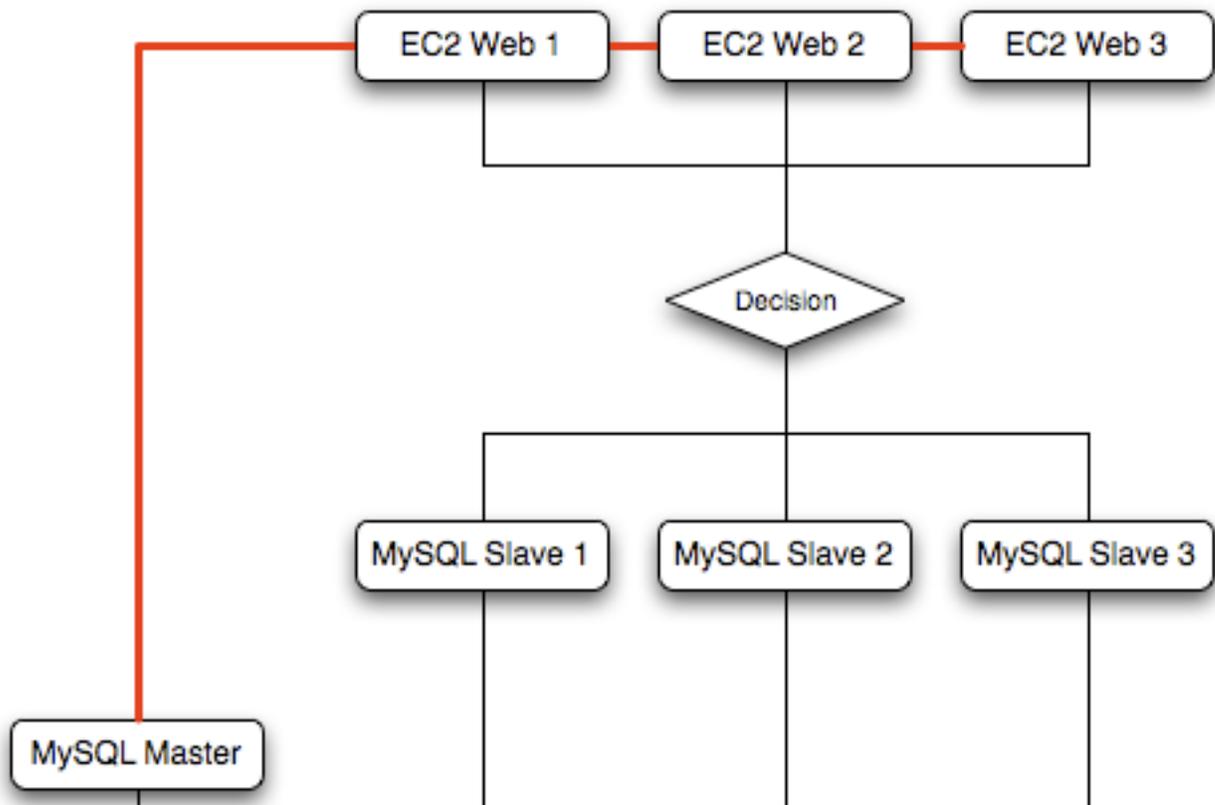
To ensure reliability of your database, add at least one replication slave dedicated to providing an active backup and storage to the Amazon S3 facility. You can see an example of this in the following topology.



Using [memcached](#) within your EC2 instances should provide better performance. The large and extra large instances have a significant amount of RAM. To use [memcached](#) in your application, when loading information from the database, first check whether the item exists in the cache. If the data you are looking for exists in the cache, use it. If not, reload the data from the database and populate the cache.

Sharding divides up data in your entire database by allocating individual machines or machine groups to provide a unique set of data according to an appropriate group. For example, you might put all users with a surname ending in the letters A-D onto a single server. When a user connects to the application and their surname is known, queries can be redirected to the appropriate MySQL server.

When using sharding with EC2, separate the web server and MySQL server into separate EC2 instances, and then apply the sharding decision logic into your application. Once you know which MySQL server you should be using for accessing the data you then distribute queries to the appropriate server. You can see a sample of this in the following illustration.



Warning

With sharding and EC2, be careful that the potential for failure of an instance does not affect your application. If the EC2 instance that provides the MySQL server for a particular shard fails, then all of the data on that shard becomes unavailable.

15.5. Using ZFS Replication

To support high availability environments, providing an instant copy of the information on both the currently active machine and the hot backup is a critical part of the HA solution. There are many solutions to this problem, including [Chapter 16, Replication](#) and [Section 15.2, “Overview of MySQL with DRBD/Pacemaker/Corosync/Oracle Linux”](#).

The ZFS file system provides functionality to create a snapshot of the file system contents, transfer the snapshot to another machine, and extract the snapshot to recreate the file system. You can create a snapshot at any time, and you can create as many snapshots as you like. By continually creating, transferring, and restoring snapshots, you can provide synchronization between one or more machines in a fashion similar to DRBD.

The following example shows a simple Solaris system running with a single ZFS pool, mounted at `/scratchpool`:

Filesystem	size	used	avail	capacity	Mounted on
/dev/dsk/c0d0s0	4.6G	3.7G	886M	82%	/
/devices	0K	0K	0K	0%	/devices
ctfs	0K	0K	0K	0%	/system/contract

Using ZFS Replication

```
proc                0K      0K      0K      0%    /proc
mnttab              0K      0K      0K      0%    /etc/mnttab
swap                1.4G    892K    1.4G    1%    /etc/svc/volatile
objfs               0K      0K      0K      0%    /system/object
/usr/lib/libc/libc_hwcap1.so.1
                    4.6G    3.7G    886M    82%   /lib/libc.so.1
fd                  0K      0K      0K      0%    /dev/fd
swap                1.4G    40K     1.4G    1%    /tmp
swap                1.4G    28K     1.4G    1%    /var/run
/dev/dsk/c0d0s7     26G     913M    25G     4%    /export/home
scratchpool         16G     24K     16G     1%    /scratchpool
```

The MySQL data is stored in a directory on `/scratchpool`. To help demonstrate some of the basic replication functionality, there are also other items stored in `/scratchpool` as well:

```
total 17
drwxr-xr-x  31 root    bin          50 Jul 21 07:32 DTT/
drwxr-xr-x   4 root    bin           5 Jul 21 07:32 SUNWmlib/
drwxr-xr-x  14 root    sys          16 Nov  5 09:56 SUNWspro/
drwxrwxrwx  19 1000   1000         40 Nov  6 19:16 emacs-22.1/
```

To create a snapshot of the file system, you use `zfs snapshot`, specifying the pool and the snapshot name:

```
root-shell> zfs snapshot scratchpool@snap1
```

To list the snapshots already taken:

```
root-shell> zfs list -t snapshot
NAME                USED  AVAIL  REFER  MOUNTPOINT
scratchpool@snap1   0     -    24.5K  -
scratchpool@snap2   0     -    24.5K  -
```

The snapshots themselves are stored within the file system metadata, and the space required to keep them varies as time goes on because of the way the snapshots are created. The initial creation of a snapshot is very quick, because instead of taking an entire copy of the data and metadata required to hold the entire snapshot, ZFS records only the point in time and metadata of when the snapshot was created.

As more changes to the original file system are made, the size of the snapshot increases because more space is required to keep the record of the old blocks. If you create lots of snapshots, say one per day, and then delete the snapshots from earlier in the week, the size of the newer snapshots might also increase, as the changes that make up the newer state have to be included in the more recent snapshots, rather than being spread over the seven snapshots that make up the week.

You cannot directly back up the snapshots because they exist within the file system metadata rather than as regular files. To get the snapshot into a format that you can copy to another file system, tape, and so on, you use the `zfs send` command to create a stream version of the snapshot.

For example, to write the snapshot out to a file:

```
root-shell> zfs send scratchpool@snap1 >/backup/scratchpool-snap1
```

Or tape:

```
root-shell> zfs send scratchpool@snap1 >/dev/rmt/0
```

You can also write out the incremental changes between two snapshots using `zfs send`:

```
root-shell> zfs send scratchpool@snap1 scratchpool@snap2 >/backup/scratchpool-changes
```

To recover a snapshot, you use `zfs recv`, which applies the snapshot information either to a new file system, or to an existing one.

15.5.1. Using ZFS for File System Replication

Because `zfs send` and `zfs recv` use streams to exchange data, you can use them to replicate information from one system to another by combining `zfs send`, `ssh`, and `zfs recv`.

For example, to copy a snapshot of the `scratchpool` file system to a new file system called `slavepool` on a new server, you would use the following command. This sequence combines the snapshot of `scratchpool`, the transmission to the slave machine (using `ssh` with login credentials), and the recovery of the snapshot on the slave using `zfs recv`:

```
root-shell> zfs send scratchpool@snap1 |ssh id@host pfexec zfs recv -F slavepool
```

The first part of the pipeline, `zfs send scratchpool@snap1`, streams the snapshot. The `ssh` command, and the command that it executes on the other server, `pfexec zfs recv -F slavepool`, receives the streamed snapshot data and writes it to `slavepool`. In this instance, I've specified the `-F` option which forces the snapshot data to be applied, and is therefore destructive. This is fine, as I'm creating the first version of my replicated file system.

On the slave machine, the replicated file system contains the exact same content:

```
root-shell> ls -al /slavepool/
total 23
drwxr-xr-x  6 root    root      7 Nov  8 09:13 ./
drwxr-xr-x 29 root    root      34 Nov  9 07:06 ../
drwxr-xr-x 31 root    bin       50 Jul 21 07:32 DTT/
drwxr-xr-x  4 root    bin       5 Jul 21 07:32 SUNWmlib/
drwxr-xr-x 14 root    sys      16 Nov  5 09:56 SUNWspro/
drwxrwxrwx 19 1000   1000    40 Nov  6 19:16 emacs-22.1/
```

Once a snapshot has been created, to synchronize the file system again, you create a new snapshot and then use the incremental snapshot feature of `zfs send` to send the changes between the two snapshots to the slave machine again:

```
root-shell> zfs send -i scratchpool@snapshot1 scratchpool@snapshot2 |ssh id@host pfexec zfs recv slavepool
```

This operation only succeeds if the file system on the slave machine has not been modified at all. You cannot apply the incremental changes to a destination file system that has changed. In the example above, the `ls` command would cause problems by changing the metadata, such as the last access time for files or directories.

To prevent changes on the slave file system, set the file system on the slave to be read-only:

```
root-shell> zfs set readonly=on slavepool
```

Setting `readonly` means that you cannot change the file system on the slave by normal means, including the file system metadata. Operations that would normally update metadata (like our `ls`) silently perform their function without attempting to update the file system state.

In essence, the slave file system is nothing but a static copy of the original file system. However, even when configured to be read-only, a file system can have snapshots applied to it. With the file system set to read only, re-run the initial copy:

```
root-shell> zfs send scratchpool@snap1 |ssh id@host pfexec zfs recv -F slavepool
```

Now you can make changes to the original file system and replicate them to the slave.

15.5.2. Configuring MySQL for ZFS Replication

Configuring MySQL on the source file system is a case of creating the data on the file system that you intend to replicate. The configuration file in the example below has been updated to use `/scratchpool/mysql-data` as the data directory, and now you can initialize the tables:

```
root-shell> mysql_install_db --defaults-file=/etc/mysql/5.5/my.cnf --user=mysql
```

To synchronize the initial information, perform a new snapshot and then send an incremental snapshot to the slave using `zfs send`:

```
root-shell> zfs snapshot scratchpool@snap2
root-shell> zfs send -i scratchpool@snap1 scratchpool@snap2|ssh id@host pfexec zfs recv slavepool
```

Doublecheck that the slave has the data by looking at the MySQL data directory on the `slavepool`:

```
root-shell> ls -al /slavepool/mysql-data/
```

Now you can start up MySQL, create some data, and then replicate the changes using `zfs send/ zfs recv` to the slave to synchronize the changes.

The rate at which you perform the synchronization depends on your application and environment. The limitation is the speed required to perform the snapshot and then to send the changes over the network.

To automate the process, create a script that performs the snapshot, send, and receive operation, and use `cron` to synchronize the changes at set times or intervals.

15.5.3. Handling MySQL Recovery with ZFS

When using ZFS replication to provide a constant copy of your data, ensure that you can recover your tables, either manually or automatically, in the event of a failure of the original system.

In the event of a failure, follow this sequence:

1. Stop the script on the master, if it is still up and running.
2. Set the slave file system to be read/write:

```
root-shell> zfs set readonly=off slavepool
```

3. Start up `mysqld` on the slave. If you are using `InnoDB`, you get auto-recovery, if it is needed, to make sure the table data is correct, as shown here when I started up from our mid-INSERT snapshot:

```
InnoDB: The log sequence number in ibdata files does not match
InnoDB: the log sequence number in the ib_logfiles!
081109 15:59:59 InnoDB: Database was not shut down normally!
InnoDB: Starting crash recovery.
InnoDB: Reading tablespace information from the .ibd files...
InnoDB: Restoring possible half-written data pages from the doublewrite
InnoDB: buffer...
081109 16:00:03 InnoDB: Started; log sequence number 0 1142807951
081109 16:00:03 [Note] /slavepool/mysql-5.0.67-solaris10-i386/bin/mysqld: ready for connections.
Version: '5.0.67' socket: '/tmp/mysql.sock' port: 3306 MySQL Community Server (GPL)
```

Use `InnoDB` tables and a regular synchronization schedule to reduce the risk for significant data loss. On `MyISAM` tables, you might need to run `REPAIR TABLE`, and you might even have lost some information.

15.6. Using MySQL with `memcached`

`memcached` is a simple, highly scalable key-based cache that stores data and objects wherever dedicated or spare RAM is available for quick access by applications, without going through layers of parsing or disk I/O. To use, you run the `memcached` command on one or more hosts and then use the shared cache to store objects. For more usage instructions, see [Section 15.6.2, “Using `memcached`”](#)

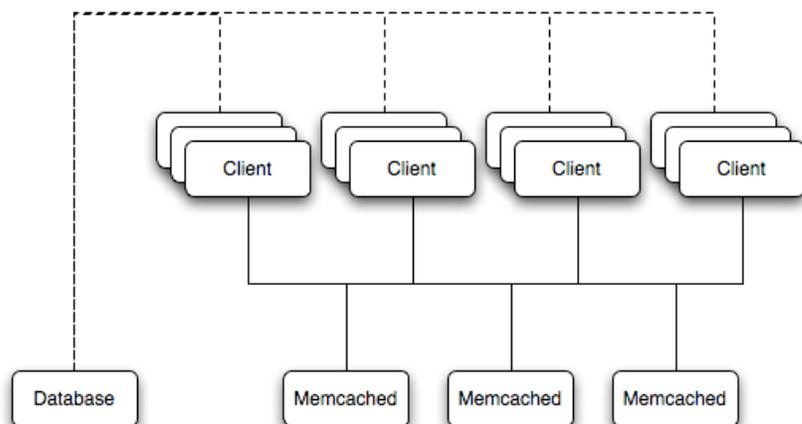
Benefits of using `memcached` include:

- Because all information is stored in RAM, the access speed is faster than loading the information each time from disk.
- Because the “value” portion of the key-value pair does not have any data type restrictions, you can cache data such as complex structures, documents, images, or a mixture of such things.
- If you use the in-memory cache to hold transient information, or as a read-only cache for information also stored in a database, the failure of any memcached server is not critical. For persistent data, you can fall back to an alternative lookup method using database queries, and reload the data into RAM on a different server.

The typical usage environment is to modify your application so that information is read from the cache provided by memcached. If the information is not in memcached, then the data is loaded from the MySQL database and written into the cache so that future requests for the same object benefit from the cached data.

For a typical deployment layout, see [Figure 15.5, “memcached Architecture Overview”](#).

Figure 15.5. memcached Architecture Overview



In the example structure, any of the clients can contact one of the memcached servers to request a given key. Each client is configured to talk to all of the servers shown in the illustration. Within the client, when the request is made to store the information, the key used to reference the data is hashed and this hash is then used to select one of the memcached servers. The selection of the memcached server takes place on the client before the server is contacted, keeping the process lightweight.

The same algorithm is used again when a client requests the same key. The same key generates the same hash, and the same memcached server is selected as the source for the data. Using this method, the cached data is spread among all of the memcached servers, and the cached information is accessible from any client. The result is a distributed, memory-based, cache that can return information, particularly complex data and structures, much faster than natively reading the information from the database.

The data held within a traditional memcached server is never stored on disk (only in RAM, which means there is no persistence of data), and the RAM cache is always populated from the backing store (a MySQL database). If a memcached server fails, the data can always be recovered from the MySQL database.

memcached Integration with MySQL Storage Engines

In April 2011, MySQL announced the preview of a new memcached interface for the InnoDB and MySQL Cluster storage engines.

Using the `memcached` API, web services can directly access the InnoDB and MySQL Cluster storage engines without transformations to SQL, ensuring low latency and high throughput for read/write queries. Operations such as SQL parsing are eliminated and more of the server's hardware resources (CPU, memory and I/O) are dedicated to servicing the query within the storage engine itself. The `memcached` data can be persisted to disk while still cached in memory for fast retrieval.

These are targeted to be incorporated into future MySQL 5.6 Milestone and MySQL Cluster Development Releases.

You can learn more about these interfaces from this Dev Zone article: <http://dev.mysql.com/tech-resources/articles/nosql-to-mysql-with-memcached.html>.

15.6.1. Installing `memcached`

You can build and install `memcached` from the source code directly, or you can use an existing operating system package or installation.

Installing `memcached` from a Binary Distribution

To install `memcached` on a Red Hat, or Fedora host, use `yum`:

```
root-shell> yum install memcached
```



Note

On CentOS, you may be able to obtain a suitable RPM from another source, or use the source tarball.

To install `memcached` on a Debian or Ubuntu host, use `apt-get`:

```
root-shell> apt-get install memcached
```

To install `memcached` on a Gentoo host, use `emerge`:

```
root-shell> emerge install memcached
```

Building `memcached` from Source

On other Unix-based platforms, including Solaris, AIX, HP-UX and Mac OS X, and Linux distributions not mentioned already, you must install from source. For Linux, make sure you have a 2.6-based kernel, which includes the improved `epoll` interface. For all platforms, ensure that you have `libevent` 1.1 or higher installed. You can obtain `libevent` from [libevent web page](#).

You can obtain the source for `memcached` from [memcached Web site](#).

To build `memcached`, follow these steps:

1. Extract the `memcached` source package:

```
shell> gunzip -c memcached-1.2.5.tar.gz | tar xf -
```

2. Change to the `memcached-1.2.5` directory:

```
shell> cd memcached-1.2.5
```

3. Run `configure`

```
shell> ./configure
```

Some additional options you might specify to the `configure`:

- `--prefix`

To specify a different installation directory, use the `--prefix` option:

```
shell> ./configure --prefix=/opt
```

The default is to use the `/usr/local` directory.

- `--with-libevent`

If you have installed `libevent` and `configure` cannot find the library, use the `--with-libevent` option to specify the location of the installed library.

- `--enable-64bit`

To build a 64-bit version of `memcached` (which enables you to use a single instance with a large RAM allocation), use `--enable-64bit`.

- `--enable-threads`

To enable multi-threading support in `memcached`, which improves the response times on servers with a heavy load, use `--enable-threads`. You must have support for the POSIX threads within your operating system to enable thread support. For more information on the threading support, see [Section 15.6.2.7, “memcached Thread Support”](#).

- `--enable-dtrace`

`memcached` includes a range of DTrace threads that can be used to monitor and benchmark a `memcached` instance. For more information, see [Section 15.6.2.5, “Using memcached and DTrace”](#).

4. Run `make` to build `memcached`:

```
shell> make
```

5. Run `make install` to install `memcached`:

```
shell> make install
```

15.6.2. Using memcached

To start using `memcached`, start the `memcached` service on one or more servers. Running `memcached` sets up the server, allocates the memory and starts listening for connections from clients.



Note

You do not need to be a privileged user (`root`) to run `memcached` except to listen on one of the privileged TCP/IP ports (below 1024). You must, however, use a user that has not had their memory limits restricted using `setrlimit` or similar.

To start the server, run `memcached` as a nonprivileged (that is, non-`root`) user:

```
shell> memcached
```

By default, `memcached` uses the following settings:

- Memory allocation of 64MB
- Listens for connections on all network interfaces, using port 11211

- Supports a maximum of 1024 simultaneous connections

Typically, you would specify the full combination of options that you want when starting `memcached`, and normally provide a startup script to handle the initialization of `memcached`. For example, the following line starts `memcached` with a maximum of 1024MB RAM for the cache, listening on port 11211 on the IP address 192.168.0.110, running as a background daemon:

```
shell> memcached -d -m 1024 -p 11211 -l 192.168.0.110
```

To ensure that `memcached` is started up on boot, check the init script and configuration parameters.

`memcached` supports the following options:

- `-u user`

If you start `memcached` as `root`, use the `-u` option to specify the user for executing `memcached`:

```
shell> memcached -u memcache
```

- `-m memory`

Set the amount of memory allocated to `memcached` for object storage. Default is 64MB.

To increase the amount of memory allocated for the cache, use the `-m` option to specify the amount of RAM to be allocated (in megabytes). The more RAM you allocate, the more data you can store and therefore the more effective your cache is.



Warning

Do not specify a memory allocation larger than your available RAM. If you specify too large a value, then some RAM allocated for `memcached` uses swap space, and not physical RAM. This may lead to delays when storing and retrieving values, because data is swapped to disk, instead of storing the data directly in RAM.

You can use the output of the `vmstat` command to get the free memory, as shown in `free` column:

```
shell> vmstat
kthr      memory          page        disk          faults        cpu
r  b  w   swap  free  re  mf  pi  po  fr  de  sr  s1  s2  --  --    in   sy   cs  us  sy  id
0  0  0 5170504 3450392 2   7   2   0   0  0  4   0   0   0   0   296  54  199  0   0 100
```

For example, to allocate 3GB of RAM:

```
shell> memcached -m 3072
```

On 32-bit x86 systems where you are using PAE to access memory above the 4GB limit, you cannot allocate RAM beyond the maximum process size. You can get around this by running multiple instances of `memcached`, each listening on a different port:

```
shell> memcached -m 1024 -p11211
shell> memcached -m 1024 -p11212
shell> memcached -m 1024 -p11213
```



Note

On all systems, particularly 32-bit, ensure that you leave enough room for both `memcached` application in addition to the memory setting. For example, if you

have a dedicated `memcached` host with 4GB of RAM, do not set the memory size above 3500MB. Failure to do this may cause either a crash or severe performance issues.

- `-l interface`

Specify a network interface/address to listen for connections. The default is to listen on all available address (`INADDR_ANY`).

```
shell> memcached -l 192.168.0.110
```

Support for IPv6 address support was added in `memcached` 1.2.5.

- `-p port`

Specify the TCP port to use for connections. Default is 18080.

```
shell> memcached -p 18080
```

- `-U port`

Specify the UDP port to use for connections. Default is 11211, 0 switches UDP off.

```
shell> memcached -U 18080
```

- `-s socket`

Specify a Unix socket to listen on.

If you are running `memcached` on the same server as the clients, you can disable the network interface and use a local UNIX socket using the `-s` option:

```
shell> memcached -s /tmp/memcached
```

Using a UNIX socket automatically disables network support, and saves network ports (allowing more ports to be used by your web server or other process).

- `-a mask`

Specify the access mask to be used for the Unix socket, in octal. Default is 0700.

- `-c connections`

Specify the maximum number of simultaneous connections to the `memcached` service. The default is 1024.

```
shell> memcached -c 2048
```

Use this option, either to reduce the number of connections (to prevent overloading `memcached` service) or to increase the number to make more effective use of the server running `memcached` server.

- `-t threads`

Specify the number of threads to use when processing incoming requests.

By default, `memcached` is configured to use 4 concurrent threads. The threading improves the performance of storing and retrieving data in the cache, using a locking system to prevent different threads overwriting or updating the same values. To increase or decrease the number of threads, use the `-t` option:

```
shell> memcached -t 8
```

- `-d`

Run `memcached` as a daemon (background) process:

```
shell> memcached -d
```

- `-r`

Maximize the size of the core file limit. In the event of a failure, this attempts to dump the entire memory space to disk as a core file, up to any limits imposed by `setrlimit`.

- `-M`

Return an error to the client when the memory has been exhausted. This replaces the normal behavior of removing older items from the cache to make way for new items.

- `-k`

Lock down all paged memory. This reserves the memory before use, instead of allocating new slabs of memory as new items are stored in the cache.



Note

There is a user-level limit on how much memory you can lock. Trying to allocate more than the available memory fails. You can set the limit for the user you started the daemon with (not for the `-u user` user) within the shell by using `ulimit -S -l NUM_KB`

- `-v`

Verbose mode. Prints errors and warnings while executing the main event loop.

- `-vv`

Very verbose mode. In addition to information printed by `-v`, also prints each client command and the response.

- `-vvv`

Extremely verbose mode. In addition to information printed by `-vv`, also show the internal state transitions.

- `-h`

Print the help message and exit.

- `-i`

Print the `memcached` and `libevent` license.

- `-I mem`

Specify the maximum size permitted for storing an object within the `memcached` instance. The size supports a unit postfix (`k` for kilobytes, `m` for megabytes). For example, to increase the maximum supported object size to 32MB:

```
shell> memcached -I 32m
```

The maximum object size you can specify is 128MB, the default remains at 1MB.

This option was added in 1.4.2.

- `-b`

Set the backlog queue limit. The backlog queue configures how many network connections can be waiting to be processed by memcached. Increasing this limit may reduce errors received by the client that it is not able to connect to the memcached instance, but does not improve the performance of the server. The default is 1024.

- `-P pidfile`

Save the process ID of the memcached instance into file.

- `-f`

Set the chunk size growth factor. When allocating new memory chunks, the allocated size of new chunks is determined by multiplying the default slab size by this factor.

To see the effects of this option without extensive testing, use the `-vv` command-line option to show the calculated slab sizes. For more information, see [Section 15.6.2.8, “memcached Logs”](#).

- `-n bytes`

The minimum space allocated for the key+value+flags information. The default is 48 bytes.

- `-L`

On systems that support large memory pages, enables large memory page use. Using large memory pages enables memcached to allocate the item cache in one large chunk, which can improve the performance by reducing the number misses when accessing memory.

- `-C`

Disable the use of compare and swap (CAS) operations.

This option was added in memcached 1.3.x.

- `-D char`

Set the default character to be used as a delimiter between the key prefixes and IDs. This is used for the per-prefix statistics reporting (see [Section 15.6.4, “Getting memcached Statistics”](#)). The default is the colon (:). If this option is used, statistics collection is turned on automatically. If not used, you can enable stats collection by sending the `stats detail on` command to the server.

This option was added in memcached 1.3.x.

- `-R num`

Sets the maximum number of requests per event process. The default is 20.

- `-B protocol`

Set the binding protocol, that is, the default memcached protocol support for client connections. Options are `ascii`, `binary` or `auto`. Automatic (`auto`) is the default.

This option was added in memcached 1.4.0.

15.6.2.1. `memcached` Deployment

When using `memcached` you can use a number of different potential deployment strategies and topologies. The exact strategy to use depends on your application and environment. When developing a system for deploying `memcached` within your system, keep in mind the following points:

- `memcached` is only a caching mechanism. It shouldn't be used to store information that you cannot otherwise afford to lose and then load from a different location.
- There is no security built into the `memcached` protocol. At a minimum, make sure that the servers running `memcached` are only accessible from inside your network, and that the network ports being used are blocked (using a firewall or similar). If the information on the `memcached` servers that is being stored is any sensitive, then encrypt the information before storing it in `memcached`.
- `memcached` does not provide any sort of failover. Because there is no communication between different `memcached` instances. If an instance fails, your application must be capable of removing it from the list, reloading the data and then writing data to another `memcached` instance.
- Latency between the clients and the `memcached` can be a problem if you are using different physical machines for these tasks. If you find that the latency is a problem, move the `memcached` instances to be on the clients.
- Key length is determined by the `memcached` server. The default maximum key size is 250 bytes.
- Try to use at least two `memcached` instances, especially for multiple clients, to avoid having a single point of failure. Ideally, create as many `memcached` nodes as possible. When adding and removing `memcached` instances from a pool, the hashing and distribution of key/value pairs may be affected. For information on how to avoid problems, see [Section 15.6.2.4, “`memcached` Hashing/Distribution Types”](#).

15.6.2.2. Using Namespaces

The `memcached` cache is a very simple massive key/value storage system, and as such there is no way of compartmentalizing data automatically into different sections. For example, if you are storing information by the unique ID returned from a MySQL database, then storing the data from two different tables could run into issues because the same ID might be valid in both tables.

Some interfaces provide an automated mechanism for creating *namespaces* when storing information into the cache. In practice, these namespaces are merely a prefix before a given ID that is applied every time a value is stored or retrieved from the cache.

You can implement the same basic principle by using keys that describe the object and the unique identifier within the key that you supply when the object is stored. For example, when storing user data, prefix the ID of the user with `user:` or `user-`.



Note

Using namespaces or prefixes only controls the keys stored/retrieved. There is no security within `memcached`, and therefore no way to enforce that a particular client only accesses keys with a particular namespace. Namespaces are only useful as a method of identifying data and preventing corruption of key/value pairs.

15.6.2.3. Data Expiry

There are two types of data expiry within a `memcached` instance. The first type is applied at the point when you store a new key/value pair into the `memcached` instance. If there is not enough space within a suitable slab to store the value, then an existing least recently used (LRU) object is removed (evicted) from the cache to make room for the new item.

The LRU algorithm ensures that the object that is removed is one that is either no longer in active use or that was used so long ago that its data is potentially out of date or of little value. However, in a system where the memory allocated to `memcached` is smaller than the number of regularly used objects required in the cache, a lot of expired items could be removed from the cache even though they are in active use. You use the statistics mechanism to get a better idea of the level of evictions (expired objects). For more information, see [Section 15.6.4, “Getting memcached Statistics”](#).

You can change this eviction behavior by setting the `-M` command-line option when starting `memcached`. This option forces an error to be returned when the memory has been exhausted, instead of automatically evicting older data.

The second type of expiry system is an explicit mechanism that you can set when a key/value pair is inserted into the cache, or when deleting an item from the cache. Using an expiration time can be a useful way of ensuring that the data in the cache is up to date and in line with your application needs and requirements.

A typical scenario for explicitly setting the expiry time might include caching session data for a user when accessing a Web site. `memcached` uses a lazy expiry mechanism where the explicit expiry time that has been set is compared with the current time when the object is requested. Only objects that have not expired are returned.

You can also set the expiry time when explicitly deleting an object from the cache. In this case, the expiry time is really a timeout and indicates the period when any attempts to set the value for a given key are rejected.

15.6.2.4. memcached Hashing/Distribution Types

The `memcached` client interface supports a number of different distribution algorithms that are used in multi-server configurations to determine which host should be used when setting or getting data from a given `memcached` instance. When you get or set a value, a hash is constructed from the supplied key and then used to select a host from the list of configured servers. Because the hashing mechanism uses the supplied key as the basis for the hash, the same server is selected during both set and get operations.

You can think of this process as follows. Given an array of servers (a, b, and c), the client uses a hashing algorithm that returns an integer based on the key being stored or retrieved. The resulting value is then used to select a server from the list of servers configured in the client. Most standard client hashing within `memcache` clients uses a simple modulus calculation on the value against the number of configured `memcached` servers. You can summarize the process in pseudocode as:

```
@memcservers = ['a.memc', 'b.memc', 'c.memc'];
$value = hash($key);
$chosen = $value % length(@memcservers);
```

Replacing the above with values:

```
@memcservers = ['a.memc', 'b.memc', 'c.memc'];
$value = hash('myid');
$chosen = 7009 % 3;
```

In the above example, the client hashing algorithm chooses the server at index 1 ($7009 \% 3 = 1$), and store or retrieve the key and value with that server.

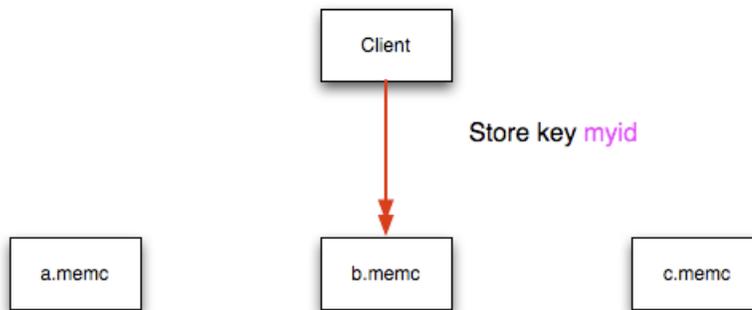


Note

This selection and hashing process is handled automatically by the `memcached` client you are using; you need only provide the list of `memcached` servers to use.

You can see a graphical representation of this below in [Figure 15.6, “memcached Hash Selection”](#).

Figure 15.6. memcached Hash Selection



The same hashing and selection process takes place during any operation on the specified key within the `memcached` client.

Using this method provides a number of advantages:

- The hashing and selection of the server to contact is handled entirely within the client. This eliminates the need to perform network communication to determine the right machine to contact.
- Because the determination of the `memcached` server occurs entirely within the client, the server can be selected automatically regardless of the operation being executed (set, get, increment, etc.).
- Because the determination is handled within the client, the hashing algorithm returns the same value for a given key; values are not affected or reset by differences in the server environment.
- Selection is very fast. The hashing algorithm on the key value is quick and the resulting selection of the server is from a simple array of available machines.
- Using client-side hashing simplifies the distribution of data over each `memcached` server. Natural distribution of the values returned by the hashing algorithm means that keys are automatically spread over the available servers.

Providing that the list of servers configured within the client remains the same, the same stored key returns the same value, and therefore selects the same server.

However, if you do not use the same hashing mechanism then the same data may be recorded on different servers by different interfaces, both wasting space on your `memcached` and leading to potential differences in the information.



Note

One way to use a multi-interface compatible hashing mechanism is to use the `libmemcached` library and the associated interfaces. Because the interfaces for the different languages (including C, Ruby, Perl and Python) use the same client library interface, they always generate the same hash code from the ID.

The problem with client-side selection of the server is that the list of the servers (including their sequential order) *must* remain consistent on each client using the `memcached` servers, and the servers must be available. If you try to perform an operation on a key when:

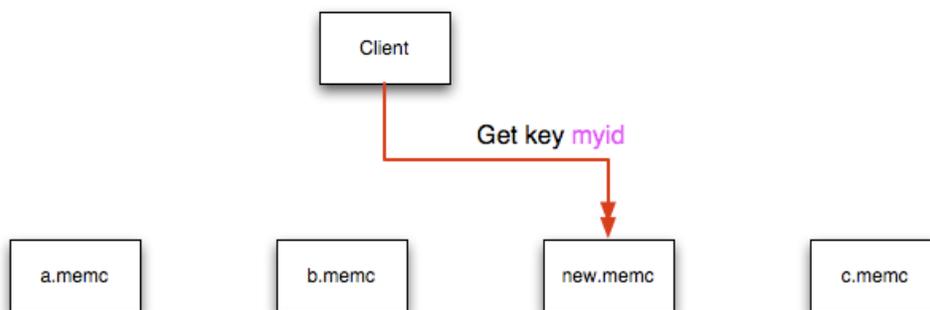
- A new `memcached` instance has been added to the list of available instances

- A `memcached` instance has been removed from the list of available instances
- The order of the `memcached` instances has changed

When the hashing algorithm is used on the given key, but with a different list of servers, the hash calculation may choose a different server from the list.

If a new `memcached` instance is added into the list of servers, as `new.memc` is in the example below, then a GET operation using the same key, `myid`, can result in a cache-miss. This is because the same value is computed from the key, which selects the same index from the array of servers, but index 2 now points to the new server, not the server `c.memc` where the data was originally stored. This would result in a cache miss, even though the key exists within the cache on another `memcached` instance.

Figure 15.7. memcached Hash Selection with New memcached instance



This means that servers `c.memc` and `new.memc` both contain the information for key `myid`, but the information stored against the key in each server may be different in each instance. A more significant problem is a much higher number of cache-misses when retrieving data, as the addition of a new server changes the distribution of keys, and this in turn requires rebuilding the cached data on the `memcached` instances, causing an increase in database reads.

The same effect can occur if you actively manage the list of servers configured in your clients, adding and removing the configured `memcached` instances as each instance is identified as being available. For example, removing a `memcached` instance when the client notices that the instance can no longer be contacted can cause the server selection to fail as described here.

To prevent this causing significant problems and invalidating your cache, you can select the hashing algorithm used to select the server. There are two common types of hashing algorithm, *consistent* and *modula*.

With *consistent* hashing algorithms, the same key when applied to a list of servers always uses the same server to store or retrieve the keys, even if the list of configured servers changes. This means that you can add and remove servers from the configure list and always use the same server for a given key. There are two types of consistent hashing algorithms available, Ketama and Wheel. Both types are supported by `libmemcached`, and implementations are available for PHP and Java.

Any consistent hashing algorithm has some limitations. When you add servers to an existing list of configured servers, keys are distributed to the new servers as part of the normal distribution. When you remove servers from the list, the keys are re-allocated to another server within the list, meaning that the cache needs to be re-populated with the information. Also, a consistent hashing algorithm does not resolve the issue where you want consistent selection of a server across multiple clients, but where each client contains a different list of servers. The consistency is enforced only within a single client.

With a *modula* hashing algorithm, the client selects a server by first computing the hash and then choosing a server from the list of configured servers. As the list of servers changes, so the server selected when using a modula hashing algorithm also changes. The result is the behavior described above; changes to the list of servers mean that different servers are selected when retrieving data, leading to cache misses and increase in database load as the cache is re-seeded with information.

If you use only a single `memcached` instance for each client, or your list of `memcached` servers configured for a client never changes, then the selection of a hashing algorithm is irrelevant, as it has no noticeable effect.

If you change your servers regularly, or you use a common set of servers that are shared among a large number of clients, then using a consistent hashing algorithm should help to ensure that your cache data is not duplicated and the data is evenly distributed.

15.6.2.5. Using `memcached` and DTrace

`memcached` includes a number of different DTrace probes that can be used to monitor the operation of the server. The probes included can monitor individual connections, slab allocations, and modifications to the hash table when a key/value pair is added, updated, or removed.

For more information on DTrace and writing DTrace scripts, read the [DTrace User Guide](#).

Support for DTrace probes was added to `memcached` 1.2.6 includes a number of DTrace probes that can be used to help monitor your application. DTrace is supported on Solaris 10, OpenSolaris, Mac OS X 10.5 and FreeBSD. To enable the DTrace probes in `memcached`, build from source and use the `--enable-dtrace` option. For more information, see [Section 15.6.1, "Installing `memcached`"](#).

The probes supported by `memcached` are:

- `conn-allocate(connid)`

Fired when a connection object is allocated from the connection pool.

- `connid`: The connection ID.

- `conn-release(connid)`

Fired when a connection object is released back to the connection pool.

Arguments:

- `connid`: The connection ID.

- `conn-create(ptr)`

Fired when a new connection object is being created (that is, there are no free connection objects in the connection pool).

Arguments:

- `ptr`: A pointer to the connection. object

- `conn-destroy(ptr)`

Fired when a connection object is being destroyed.

Arguments:

- `ptr`: A pointer to the connection object.

- `conn-dispatch(connid, threadid)`

Fired when a connection is dispatched from the main or connection-management thread to a worker thread.

Arguments:

- `connid`: The connection ID.
- `threadid`: The thread ID.

- `slabs-allocate(size, slabclass, slabsize, ptr)`

Allocate memory from the slab allocator.

Arguments:

- `size`: The requested size.
- `slabclass`: The allocation is fulfilled in this class.
- `slabsize`: The size of each item in this class.
- `ptr`: A pointer to allocated memory.

- `slabs-allocate-failed(size, slabclass)`

Failed to allocate memory (out of memory).

Arguments:

- `size`: The requested size.
- `slabclass`: The class that failed to fulfill the request.

- `slabs-slabclass-allocate(slabclass)`

Fired when a slab class needs more space.

Arguments:

- `slabclass`: The class that needs more memory.

- `slabs-slabclass-allocate-failed(slabclass)`

Failed to allocate memory (out of memory).

Arguments:

- `slabclass`: The class that failed to grab more memory.

- `slabs-free(size, slabclass, ptr)`

Release memory.

Arguments:

- `size`: The amount of memory to release, in bytes.
- `slabclass`: The class the memory belongs to.
- `ptr`: A pointer to the memory to release.
- `assoc-find(key, depth)`

Fired when we have searched the hash table for a named key. These two elements provide an insight into how well the hash function operates. Traversals are a sign of a less optimal function, wasting CPU capacity.

Arguments:

 - `key`: The key searched for.
 - `depth`: The depth in the list of hash table.
- `assoc-insert(key, nokeys)`

Fired when a new item has been inserted.

Arguments:

 - `key`: The key just inserted.
 - `nokeys`: The total number of keys currently being stored, including the key for which insert was called.
- `assoc-delete(key, nokeys)`

Fired when a new item has been removed.

Arguments:

 - `key`: The key just deleted.
 - `nokeys`: The total number of keys currently being stored, excluding the key for which delete was called.
- `item-link(key, size)`

Fired when an item is being linked in the cache.

Arguments:

 - `key`: The items key.
 - `size`: The size of the data.
- `item-unlink(key, size)`

Fired when an item is being deleted.

Arguments:

 - `key`: The items key.
 - `size`: The size of the data.

- `item-remove(key, size)`

Fired when the refcount for an item is reduced.

Arguments:

 - `key`: The item's key.
 - `size`: The size of the data.
- `item-update(key, size)`

Fired when the "last referenced" time is updated.

Arguments:

 - `key`: The item's key.
 - `size`: The size of the data.
- `item-replace(oldkey, oldsize, newkey, newsize)`

Fired when an item is being replaced with another item.

Arguments:

 - `oldkey`: The key of the item to replace.
 - `oldsize`: The size of the old item.
 - `newkey`: The key of the new item.
 - `newsize`: The size of the new item.
- `process-command-start(connid, request, size)`

Fired when the processing of a command starts.

Arguments:

 - `connid`: The connection ID.
 - `request`: The incoming request.
 - `size`: The size of the request.
- `process-command-end(connid, response, size)`

Fired when the processing of a command is done.

Arguments:

 - `connid`: The connection ID.
 - `response`: The response to send back to the client.
 - `size`: The size of the response.
- `command-get(connid, key, size)`

Fired for a `get` command.

Arguments:

- `connid`: The connection ID.
 - `key`: The requested key.
 - `size`: The size of the key's data (or -1 if not found).
- `command-gets(connid, key, size, casid)`

Fired for a `gets` command.

Arguments:

- `connid`: The connection ID.
 - `key`: The requested key.
 - `size`: The size of the key's data (or -1 if not found).
 - `casid`: The casid for the item.
- `command-add(connid, key, size)`

Fired for a `add` command.

Arguments:

- `connid`: The connection ID.
 - `key`: The requested key.
 - `size`: The new size of the key's data (or -1 if not found).
- `command-set(connid, key, size)`

Fired for a `set` command.

Arguments:

- `connid`: The connection ID.
 - `key`: The requested key.
 - `size`: The new size of the key's data (or -1 if not found).
- `command-replace(connid, key, size)`

Fired for a `replace` command.

Arguments:

- `connid`: The connection ID.
- `key`: The requested key.

- `size`: The new size of the key's data (or -1 if not found).

- `command-prepend(connid, key, size)`

Fired for a `prepend` command.

Arguments:

- `connid`: The connection ID.
- `key`: The requested key.
- `size`: The new size of the key's data (or -1 if not found).

- `command-append(connid, key, size)`

Fired for a `append` command.

Arguments:

- `connid`: The connection ID.
- `key`: The requested key.
- `size`: The new size of the key's data (or -1 if not found).

- `command-cas(connid, key, size, casid)`

Fired for a `cas` command.

Arguments:

- `connid`: The connection ID.
- `key`: The requested key.
- `size`: The size of the key's data (or -1 if not found).
- `casid`: The cas ID requested.

- `command-incr(connid, key, val)`

Fired for `incr` command.

Arguments:

- `connid`: The connection ID.
- `key`: The requested key.
- `val`: The new value.

- `command-decr(connid, key, val)`

Fired for `decr` command.

Arguments:

- `connid`: The connection ID.
- `key`: The requested key.
- `val`: The new value.
- `command-delete(connid, key, exptime)`

Fired for a `delete` command.

Arguments:

- `connid`: The connection ID.
- `key`: The requested key.
- `exptime`: The expiry time.

15.6.2.6. Memory Allocation within `memcached`

When you first start `memcached`, the memory that you have configured is not automatically allocated. Instead, `memcached` only starts allocating and reserving physical memory once you start saving information into the cache.

When you start to store data into the cache, `memcached` does not allocate the memory for the data on an item by item basis. Instead, a slab allocation is used to optimize memory usage and prevent memory fragmentation when information expires from the cache.

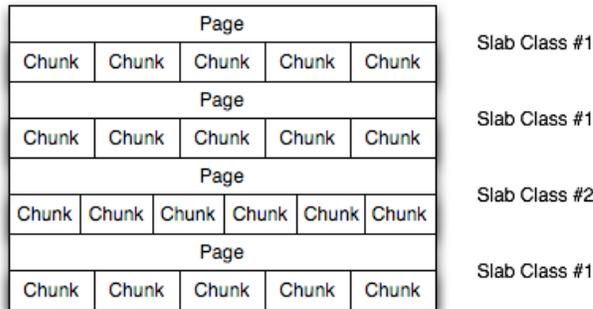
With slab allocation, memory is reserved in blocks of 1MB. The slab is divided up into a number of blocks of equal size. When you try to store a value into the cache, `memcached` checks the size of the value that you are adding to the cache and determines which slab contains the right size allocation for the item. If a slab with the item size already exists, the item is written to the block within the slab.

If the new item is bigger than the size of any existing blocks, then a new slab is created, divided up into blocks of a suitable size. If an existing slab with the right block size already exists, but there are no free blocks, a new slab is created. If you update an existing item with data that is larger than the existing block allocation for that key, then the key is re-allocated into a suitable slab.

For example, the default size for the smallest block is 88 bytes (40 bytes of value, and the default 48 bytes for the key and flag data). If the size of the first item you store into the cache is less than 40 bytes, then a slab with a block size of 88 bytes is created and the value stored.

If the size of the data that you intend to store is larger than this value, then the block size is increased by the chunk size factor until a block size large enough to hold the value is determined. The block size is always a function of the scale factor, rounded up to a block size which is exactly divisible into the chunk size.

For a sample of the structure, see [Figure 15.8, “Memory Allocation in `memcached`”](#).

Figure 15.8. Memory Allocation in memcached

The result is that you have multiple pages allocated within the range of memory allocated to `memcached`. Each page is 1MB in size (by default), and is split into a different number of chunks, according to the chunk size required to store the key/value pairs. Each instance has multiple pages allocated, and a page is always created when a new item needs to be created requiring a chunk of a particular size. A slab may consist of multiple pages, and each page within a slab contains an equal number of chunks.

The chunk size of a new slab is determined by the base chunk size combined with the chunk size growth factor. For example, if the initial chunks are 104 bytes in size, and the default chunk size growth factor is used (1.25), then the next chunk size allocated would be the best power of 2 fit for $104 * 1.25$, or 136 bytes.

Allocating the pages in this way ensures that memory does not get fragmented. However, depending on the distribution of the objects that you store, it may lead to an inefficient distribution of the slabs and chunks if you have significantly different sized items. For example, having a relatively small number of items within each chunk size may waste a lot of memory with just few chunks in each allocated page.

You can tune the growth factor to reduce this effect by using the `-f` command line option, which adapts the growth factor applied to make more effective use of the chunks and slabs allocated. For information on how to determine the current slab allocation statistics, see [Section 15.6.4.2, “memcached Slabs Statistics”](#).

If your operating system supports it, you can also start `memcached` with the `-L` command line option. This option preallocates all the memory during startup using large memory pages. This can improve performance by reducing the number of misses in the CPU memory cache.

15.6.2.7. memcached Thread Support

If you enable the thread implementation within when building `memcached` from source, then `memcached` uses multiple threads in addition to the `libevent` system to handle requests.

When enabled, the threading implementation operates as follows:

- Threading is handled by wrapping functions within the code to provide basic protection from updating the same global structures at the same time.
- Each thread uses its own instance of the `libevent` to help improve performance.
- TCP/IP connections are handled with a single thread listening on the TCP/IP socket. Each connection is then distributed to one of the active threads on a simple round-robin basis. Each connection then operates solely within this thread while the connection remains open.
- For UDP connections, all the threads listen to a single UDP socket for incoming requests. Threads that are not currently dealing with another request ignore the incoming packet. One of the remaining, nonbusy, threads reads the request and sends the response. This implementation can lead to increased CPU load as threads wake from sleep to potentially process the request.

Using threads can increase the performance on servers that have multiple CPU cores available, as the requests to update the hash table can be spread between the individual threads. To minimize overhead from the locking mechanism employed, experiment with different thread values to achieve the best performance based on the number and type of requests within your given workload.

15.6.2.8. memcached Logs

If you enable verbose mode, using the `-v`, `-vv`, or `-vvv` options, then the information output by memcached includes details of the operations being performed.

Without the verbose options, memcached normally produces no output during normal operating.

- **Output when using `-v`**

The lowest verbosity level shows you:

- Errors and warnings
- Transient errors
- Protocol and socket errors, including exhausting available connections
- Each registered client connection, including the socket descriptor number and the protocol used.

For example:

```
32: Client using the ascii protocol
33: Client using the ascii protocol
```

Note that the socket descriptor is only valid while the client remains connected. Non-persistent connections may not be effectively represented.

Examples of the error messages output at this level include:

```
<%d send buffer was %d, now %d
Can't listen for events on fd %d
Can't read from libevent pipe
Catastrophic: event fd doesn't match conn fd!
Couldn't build response
Couldn't realloc input buffer
Couldn't update event
Failed to build UDP headers
Failed to read, and not due to blocking
Too many open connections
Unexpected state %d
```

- **Output when using `-vv`**

When using the second level of verbosity, you get more detailed information about protocol operations, keys updated, chunk and network operations and details.

During the initial start-up of memcached with this level of verbosity, you are shown the sizes of the individual slab classes, the chunk sizes, and the number of entries per slab. These do not show the allocation of the slabs, just the slabs that would be created when data is added. You are also given information about the listen queues and buffers used to send information. A sample of the output generated for a TCP/IP based system with the default memory and growth factors is given below:

```
shell> memcached -vv
slab class 1: chunk size      80 perslab 13107
slab class 2: chunk size     104 perslab 10082
slab class 3: chunk size     136 perslab  7710
```

```

slab class 4: chunk size 176 perslab 5957
slab class 5: chunk size 224 perslab 4681
slab class 6: chunk size 280 perslab 3744
slab class 7: chunk size 352 perslab 2978
slab class 8: chunk size 440 perslab 2383
slab class 9: chunk size 552 perslab 1899
slab class 10: chunk size 696 perslab 1506
slab class 11: chunk size 872 perslab 1202
slab class 12: chunk size 1096 perslab 956
slab class 13: chunk size 1376 perslab 762
slab class 14: chunk size 1720 perslab 609
slab class 15: chunk size 2152 perslab 487
slab class 16: chunk size 2696 perslab 388
slab class 17: chunk size 3376 perslab 310
slab class 18: chunk size 4224 perslab 248
slab class 19: chunk size 5280 perslab 198
slab class 20: chunk size 6600 perslab 158
slab class 21: chunk size 8256 perslab 127
slab class 22: chunk size 10320 perslab 101
slab class 23: chunk size 12904 perslab 81
slab class 24: chunk size 16136 perslab 64
slab class 25: chunk size 20176 perslab 51
slab class 26: chunk size 25224 perslab 41
slab class 27: chunk size 31536 perslab 33
slab class 28: chunk size 39424 perslab 26
slab class 29: chunk size 49280 perslab 21
slab class 30: chunk size 61600 perslab 17
slab class 31: chunk size 77000 perslab 13
slab class 32: chunk size 96256 perslab 10
slab class 33: chunk size 120320 perslab 8
slab class 34: chunk size 150400 perslab 6
slab class 35: chunk size 188000 perslab 5
slab class 36: chunk size 235000 perslab 4
slab class 37: chunk size 293752 perslab 3
slab class 38: chunk size 367192 perslab 2
slab class 39: chunk size 458992 perslab 2
<26 server listening (auto-negotiate)
<29 server listening (auto-negotiate)
<30 send buffer was 57344, now 2097152
<31 send buffer was 57344, now 2097152
<30 server listening (udp)
<30 server listening (udp)
<31 server listening (udp)
<30 server listening (udp)
<30 server listening (udp)
<31 server listening (udp)
<31 server listening (udp)
<31 server listening (udp)

```

Using this verbosity level can be a useful way to check the effects of the growth factor used on slabs with different memory allocations, which in turn can be used to better tune the growth factor to suit the data you are storing in the cache. For example, if you set the growth factor to 4 (quadrupling the size of each slab):

```

shell> memcached -f 4 -m 1g -vv
slab class 1: chunk size 80 perslab 13107
slab class 2: chunk size 320 perslab 3276
slab class 3: chunk size 1280 perslab 819
slab class 4: chunk size 5120 perslab 204
slab class 5: chunk size 20480 perslab 51
slab class 6: chunk size 81920 perslab 12
slab class 7: chunk size 327680 perslab 3
...

```

During use of the cache, this verbosity level also prints out detailed information on the storage and recovery of keys and other information. An example of the output during a typical set/get and increment/decrement operation is shown below.

```

32: Client using the ascii protocol
<32 set my_key 0 0 10
>32 STORED
<32 set object_key 1 0 36
>32 STORED
<32 get my_key
>32 sending key my_key
>32 END
<32 get object_key
>32 sending key object_key
>32 END
<32 set key 0 0 6
>32 STORED
<32 incr key 1
>32 789544
<32 decr key 1
>32 789543
<32 incr key 2
>32 789545
<32 set my_key 0 0 10
>32 STORED
<32 set object_key 1 0 36
>32 STORED
<32 get my_key
>32 sending key my_key
>32 END
<32 get object_key
>32 sending key object_key1 1 36

>32 END
<32 set key 0 0 6
>32 STORED
<32 incr key 1
>32 789544
<32 decr key 1
>32 789543
<32 incr key 2
>32 789545

```

During client communication, for each line, the initial character shows the direction of flow of the information. The < for communication from the client to the memcached server and > for communication back to the client. The number is the numeric socket descriptor for the connection.

- **Output when using `-vvv`**

This level of verbosity includes the transitions of connections between different states in the event library while reading and writing content to/from the clients. It should be used to diagnose and identify issues in client communication. For example, you can use this information to determine if memcached is taking a long time to return information to the client, during the read of the client operation or before returning and completing the operation. An example of the typical sequence for a set operation is provided below:

```

<32 new auto-negotiating client connection
32: going from conn_new_cmd to conn_waiting
32: going from conn_waiting to conn_read
32: going from conn_read to conn_parse_cmd
32: Client using the ascii protocol
<32 set my_key 0 0 10
32: going from conn_parse_cmd to conn_nread
> NOT FOUND my_key

```

```
>32 STORED
32: going from conn_nread to conn_write
32: going from conn_write to conn_new_cmd
32: going from conn_new_cmd to conn_waiting
32: going from conn_waiting to conn_read
32: going from conn_read to conn_closing
<32 connection closed.
```

All of the verbosity levels in `memcached` are designed to be used during debugging or examination of issues. The quantity of information generated, particularly when using `-vvv`, is significant, particularly on a busy server. Also be aware that writing the error information out, especially to disk, may negate some of the performance gains you achieve by using `memcached`. Therefore, use in production or deployment environments is not recommended.

15.6.3. Developing a `memcached` Application

A number of language interfaces let applications store and retrieve information with `memcached` servers. You can write `memcached` applications in popular languages such as Perl, PHP, Python, Ruby, C, and Java.

Data stored into a `memcached` server is referred to by a single string (the key), with storage into the cache and retrieval from the cache using the key as the reference. The cache therefore operates like a large associative array or hash table. It is not possible to structure or otherwise organize the information stored in the cache. To emulate database notions such as multiple tables or composite key values, you must use encode the extra information into the strings used as keys. For example, to store or look up the address corresponding to a specific latitude and longitude, you might turn those two numeric values into a single comma-separated string to use as a key.

15.6.3.1. Basic `memcached` Operations

The interface to `memcached` supports the following methods for storing and retrieving information in the cache, and these are consistent across all the different APIs, although the language specific mechanics might be different:

- `get(key)`: Retrieves information from the cache. Returns the value associated with the key if the specified key exists. Returns `NULL`, `nil`, `undefined`, or the closest equivalent in the corresponding language, if the specified key does not exist.
- `set(key, , value, [, expiry])`: Sets the item associated with a key in the cache to the specified value. Note that this either updates an existing item if the key already exists, or adds a new key/value pair if the key doesn't exist. If the expiry time is specified, then the item expires (and is deleted) when the expiry time is reached. The time is specified in seconds, and is taken as a relative time if the value is less than 30 days ($30 \times 24 \times 60 \times 60$), or an absolute time (epoch) if larger than this value.
- `add(key, , value, [, expiry])`: Adds the key and associated value to the cache, if the specified key does not already exist.
- `replace(key, , value, [, expiry])`: Replaces the item associated with the specified `key`, only if the key already exists. The new value is given by the `value` parameter.
- `delete(key, [, time])`: Deletes the `key` and its associated item from the cache. If you supply a `time`, then adding another item with the specified `key` is blocked for the specified period.
- `incr(key, [, value])`: Increments the item associated with the `key` by one or the optional `value`.
- `decr(key, [, value])`: Decrements the item associated with the `key` by one or the optional `value`.

- `flush_all`: Invalidates (or expires) all the current items in the cache. Technically they still exist (they are not deleted), but they are silently destroyed the next time you try to access them.

In all implementations, most or all of these functions are duplicated through the corresponding native language interface.

When practical, use `memcached` to store full items, rather than caching a single column value from the database. For example, when displaying a record about an object (invoice, user history, or blog post), load all the data for the associated entry from the database, and compile it into the internal structure that would normally be required by the application. Save the complete object in the cache.

Complex data structures cannot be stored directly. Most interfaces serialize the data for you, that is, put it in a textual form that can reconstruct the original pointers and nesting. Perl uses `Storable`, PHP uses `serialize`, Python uses `cPickle` (or `Pickle`) and Java uses the `Serializable` interface. In most cases, the serialization interface used is customizable. To share data stored in `memcached` instances between different language interfaces, consider using a common serialization solution such as JSON (Javascript Object Notation).

15.6.3.2. Using `memcached` as a MySQL Caching Layer

When using `memcached` to cache MySQL data, your application must retrieve data from the database and load the appropriate key-value pairs into the cache. Then, subsequent lookups can be done directly from the cache.

Because MySQL has its own in-memory caching mechanisms for queried data, such as the `InnoDB buffer pool` and the MySQL query cache, look for opportunities beyond loading individual column values or rows into the cache. Prefer to cache composite values, such as those retrieved from multiple tables through a join query, or result sets assembled from multiple rows.



Caution

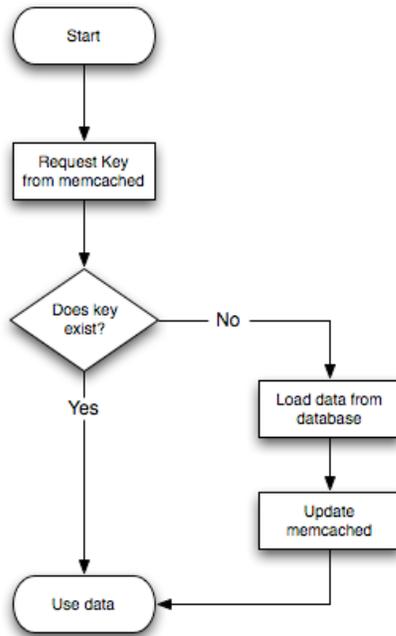
Limit the information in the cache to non-sensitive data, because there is no security required to access or update the information within a `memcached` instance. Anybody with access to the machine has the ability to read, view and potentially update the information. To keep the data secure, encrypt the information before caching it. To restrict the users capable of connecting to the server, either disable network access, or use `IPTables` or similar techniques to restrict access to the `memcached` ports to a select set of hosts.

You can introduce `memcached` to an existing application, even if caching was not part of the original design. In many languages and environments the changes to the application will be just a few lines, first to attempt to read from the cache when loading data, fall back to the old method if the information is not cached, and to update the cache with information once the data has been read.

The general sequence for using `memcached` in any language as a caching solution for MySQL is as follows:

1. Request the item from the cache.
2. If the item exists, use the item data.
3. If the item does not exist, load the data from MySQL, and store the value into the cache. This means the value is available to the next client that requests it from the cache.

For a flow diagram of this sequence, see [Figure 15.9, “Typical `memcached` Application Flowchart”](#).

Figure 15.9. Typical `memcached` Application Flowchart

Adapting Database Best Practices to `memcached` Applications

The most direct way to cache MySQL data is to use a 2-column table, where the first column is a [primary key](#). Because of the uniqueness requirements for `memcached` keys, make sure your database schema makes appropriate use of primary keys and [unique constraints](#).

If you combine multiple column values into a single `memcached` item value, choose data types to make it easy to parse the value back into its components, for example by using a separator character between numeric values.

The queries that map most easily to `memcached` lookups are those with a single `WHERE` clause, using an `=` or `IN` operator. For complicated `WHERE` clauses, or those using operators such as `<`, `>`, `BETWEEN`, or `LIKE`, `memcached` does not provide a simple or efficient way to scan through or filter the keys or associated values, so typically you perform those operations as SQL queries on the underlying database.

15.6.3.3. Using `libmemcached` with C and C++

The `libmemcached` library provides both C and C++ interfaces to `memcached` and is also the basis for a number of different additional API implementations, including Perl, Python and Ruby. Understanding the core `libmemcached` functions can help when using these other interfaces.

The C library is the most comprehensive interface library for `memcached` and provides functions and operational systems not always exposed in interfaces not based on the `libmemcached` library.

The different functions can be divided up according to their basic operation. In addition to functions that interface to the core API, a number of utility functions provide extended functionality, such as appending and prepending data.

To build and install `libmemcached`, download the `libmemcached` package, run `configure`, and then build and install:

```
shell> tar xjf libmemcached-0.21.tar.gz
shell> cd libmemcached-0.21
shell> ./configure
shell> make
shell> make install
```

On many Linux operating systems, you can install the corresponding `libmemcached` package through the usual `yum`, `apt-get`, or similar commands.

To build an application that uses the library, first set the list of servers. Either directly manipulate the servers configured within the main `memcached_st` structure, or separately populate a list of servers, and then add this list to the `memcached_st` structure. The latter method is used in the following example. Once the server list has been set, you can call the functions to store or retrieve data. A simple application for setting a preset value to `localhost` is provided here:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
    memcached_server_st *servers = NULL;
    memcached_st *memc;
    memcached_return rc;
    char *key= "keystring";
    char *value= "keyvalue";

    memcached_server_st *memcached_servers_parse (char *server_strings);
    memc= memcached_create(NULL);

    servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
    rc= memcached_server_push(memc, servers);

    if (rc == MEMCACHED_SUCCESS)
        fprintf(stderr, "Added server successfully\n");
    else
        fprintf(stderr, "Couldn't add server: %s\n", memcached_strerror(memc, rc));

    rc= memcached_set(memc, key, strlen(key), value, strlen(value), (time_t)0, (uint32_t)0);

    if (rc == MEMCACHED_SUCCESS)
        fprintf(stderr, "Key stored successfully\n");
    else
        fprintf(stderr, "Couldn't store key: %s\n", memcached_strerror(memc, rc));

    return 0;
}
```

To test the success of an operation, use the return value, or populated result code, for a given function. The value is always set to `MEMCACHED_SUCCESS` if the operation succeeded. In the event of a failure, use the `memcached_strerror()` function to translate the result code into a printable string.

To build the application, specify the `memcached` library:

```
shell> gcc -o memc_basic memc_basic.c -lmemcached
```

Running the above sample application, after starting a `memcached` server, should return a success message:

```
shell> memc_basic
Added server successfully
Key stored successfully
```

15.6.3.3.1. `libmemcached` Base Functions

The base `libmemcached` functions let you create, destroy and clone the main `memcached_st` structure that is used to interface with the `memcached` servers. The main functions are defined below:

```
memcached_st *memcached_create (memcached_st *ptr);
```

Creates a new `memcached_st` structure for use with the other `libmemcached` API functions. You can supply an existing, static, `memcached_st` structure, or `NULL` to have a new structured allocated. Returns a pointer to the created structure, or `NULL` on failure.

```
void memcached_free (memcached_st *ptr);
```

Frees the structure and memory allocated to a previously created `memcached_st` structure.

```
memcached_st *memcached_clone(memcached_st *clone, memcached_st *source);
```

Clones an existing `memcached` structure from the specified `source`, copying the defaults and list of servers defined in the structure.

15.6.3.3.2. `libmemcached` Server Functions

The `libmemcached` API uses a list of servers, stored within the `memcached_server_st` structure, to act as the list of servers used by the rest of the functions. To use `memcached`, you first create the server list, and then apply the list of servers to a valid `libmemcached` object.

Because the list of servers, and the list of servers within an active `libmemcached` object can be manipulated separately, you can update and manage server lists while an active `libmemcached` interface is running.

The functions for manipulating the list of servers within a `memcached_st` structure are:

```
memcached_return
memcached_server_add (memcached_st *ptr,
                    char *hostname,
                    unsigned int port);
```

Adds a server, using the given `hostname` and `port` into the `memcached_st` structure given in `ptr`.

```
memcached_return
memcached_server_add_unix_socket (memcached_st *ptr,
                                char *socket);
```

Adds a Unix socket to the list of servers configured in the `memcached_st` structure.

```
unsigned int memcached_server_count (memcached_st *ptr);
```

Returns a count of the number of configured servers within the `memcached_st` structure.

```
memcached_server_st *
memcached_server_list (memcached_st *ptr);
```

Returns an array of all the defined hosts within a `memcached_st` structure.

```
memcached_return
memcached_server_push (memcached_st *ptr,
                    memcached_server_st *list);
```

Pushes an existing list of servers onto list of servers configured for a current `memcached_st` structure. This adds servers to the end of the existing list, and duplicates are not checked.

The `memcached_server_st` structure can be used to create a list of `memcached` servers which can then be applied individually to `memcached_st` structures.

```
memcached_server_st *
memcached_server_list_append (memcached_server_st *ptr,
                             char *hostname,
                             unsigned int port,
                             memcached_return *error);
```

Adds a server, with `hostname` and `port`, to the server list in `ptr`. The result code is handled by the `error` argument, which should point to an existing `memcached_return` variable. The function returns a pointer to the returned list.

```
unsigned int memcached_server_list_count (memcached_server_st *ptr);
```

Returns the number of the servers in the server list.

```
void memcached_server_list_free (memcached_server_st *ptr);
```

Frees the memory associated with a server list.

```
memcached_server_st *memcached_servers_parse (char *server_strings);
```

Parses a string containing a list of servers, where individual servers are separated by a comma, space, or both, and where individual servers are of the form `server[:port]`. The return value is a server list structure.

15.6.3.3.3. libmemcached Set Functions

The set-related functions within `libmemcached` provide the same functionality as the core functions supported by the `memcached` protocol. The full definition for the different functions is the same for all the base functions (`add`, `replace`, `prepend`, `append`). For example, the function definition for `memcached_set()` is:

```
memcached_return
memcached_set (memcached_st *ptr,
              const char *key,
              size_t key_length,
              const char *value,
              size_t value_length,
              time_t expiration,
              uint32_t flags);
```

The `ptr` is the `memcached_st` structure. The `key` and `key_length` define the key name and length, and `value` and `value_length` the corresponding value and length. You can also set the expiration and optional flags. For more information, see [Section 15.6.3.3.5, "Controlling libmemcached Behaviors"](#).

The following table outlines the remainder of the set-related functions.

libmemcached Function	Equivalent to
<code>memcached_set(memc, key, key_length, value, value_length, expiration, flags)</code>	Generic <code>set()</code> operation.
<code>memcached_add(memc, key, key_length, value, value_length, expiration, flags)</code>	Generic <code>add()</code> function.
<code>memcached_replace(memc, key, key_length, value, value_length, expiration, flags)</code>	Generic <code>replace()</code> .
<code>memcached_prepend(memc, key, key_length, value, value_length, expiration, flags)</code>	Prepends the specified <code>value</code> before the current value of the specified <code>key</code> .

Libmemcached Function	Equivalent to
<code>memcached_append(memc, key, key_length, value, value_length, expiration, flags)</code>	Appends the specified <code>value</code> after the current value of the specified <code>key</code> .
<code>memcached_cas(memc, key, key_length, value, value_length, expiration, flags, cas)</code>	Overwrites the data for a given key as long as the corresponding <code>cas</code> value is still the same within the server.
<code>memcached_set_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)</code>	Similar to the generic <code>set()</code> , but has the option of an additional master key that can be used to identify an individual server.
<code>memcached_add_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)</code>	Similar to the generic <code>add()</code> , but has the option of an additional master key that can be used to identify an individual server.
<code>memcached_replace_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)</code>	Similar to the generic <code>replace()</code> , but has the option of an additional master key that can be used to identify an individual server.
<code>memcached_prepend_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)</code>	Similar to the <code>memcached_prepend()</code> , but has the option of an additional master key that can be used to identify an individual server.
<code>memcached_append_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)</code>	Similar to the <code>memcached_append()</code> , but has the option of an additional master key that can be used to identify an individual server.
<code>memcached_cas_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)</code>	Similar to the <code>memcached_cas()</code> , but has the option of an additional master key that can be used to identify an individual server.

The `by_key` methods add two further arguments that define the master key, to be used and applied during the hashing stage for selecting the servers. You can see this in the following definition:

```
memcached_return
memcached_set_by_key(memcached_st *ptr,
                    const char *master_key,
                    size_t master_key_length,
                    const char *key,
                    size_t key_length,
                    const char *value,
                    size_t value_length,
                    time_t expiration,
                    uint32_t flags);
```

All the functions return a value of type `memcached_return`, which you can compare against the `MEMCACHED_SUCCESS` constant.

15.6.3.3.4. libmemcached Get Functions

The `libmemcached` functions provide both direct access to a single item, and a multiple-key request mechanism that provides much faster responses when fetching a large number of keys simultaneously.

The main get-style function, which is equivalent to the generic `get()` is `memcached_get()`. This function returns a string pointer, pointing to the value associated with the specified key.

```
char *memcached_get (memcached_st *ptr,
                    const char *key, size_t key_length,
                    size_t *value_length,
                    uint32_t *flags,
                    memcached_return *error);
```

A multi-key get, `memcached_mget()`, is also available. Using a multiple key get operation is much quicker to do in one block than retrieving the key values with individual calls to `memcached_get()`. To start the multi-key get, call `memcached_mget()`:

```
memcached_return
memcached_mget (memcached_st *ptr,
               char **keys, size_t *key_length,
               unsigned int number_of_keys);
```

The return value is the success of the operation. The `keys` parameter should be an array of strings containing the keys, and `key_length` an array containing the length of each corresponding key. `number_of_keys` is the number of keys supplied in the array.

To fetch the individual values, use `memcached_fetch()` to get each corresponding value.

```
char *memcached_fetch (memcached_st *ptr,
                      const char *key, size_t *key_length,
                      size_t *value_length,
                      uint32_t *flags,
                      memcached_return *error);
```

The function returns the key value, with the `key`, `key_length` and `value_length` parameters being populated with the corresponding key and length information. The function returns `NULL` when there are no more values to be returned. A full example, including the populating of the key data and the return of the information is provided here.

```
#include <stdio.h>
#include <sstring.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
    memcached_server_st *servers = NULL;
    memcached_st *memc;
    memcached_return rc;
    char *keys[]= {"huey", "dewey", "louie"};
    size_t key_length[3];
    char *values[]= {"red", "blue", "green"};
    size_t value_length[3];
    unsigned int x;
    uint32_t flags;

    char return_key[MEMCACHED_MAX_KEY];
    size_t return_key_length;
    char *return_value;
    size_t return_value_length;

    memc= memcached_create(NULL);

    servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
    rc= memcached_server_push(memc, servers);

    if (rc == MEMCACHED_SUCCESS)
        fprintf(stderr, "Added server successfully\n");
    else
```

```

    fprintf(stderr, "Couldn't add server: %s\n", memcached_strerror(memc, rc));

    for(x= 0; x < 3; x++)
    {
        key_length[x] = strlen(keys[x]);
        value_length[x] = strlen(values[x]);

        rc= memcached_set(memc, keys[x], key_length[x], values[x],
                          value_length[x], (time_t)0, (uint32_t)0);
        if (rc == MEMCACHED_SUCCESS)
            fprintf(stderr, "Key %s stored successfully\n", keys[x]);
        else
            fprintf(stderr, "Couldn't store key: %s\n", memcached_strerror(memc, rc));
    }

    rc= memcached_mget(memc, keys, key_length, 3);

    if (rc == MEMCACHED_SUCCESS)
    {
        while ((return_value= memcached_fetch(memc, return_key, &return_key_length,
                                              &return_value_length, &flags, &rc)) != NULL)
        {
            if (rc == MEMCACHED_SUCCESS)
            {
                fprintf(stderr, "Key %s returned %s\n", return_key, return_value);
            }
        }
    }

    return 0;
}

```

Running the above application produces the following output:

```

shell> memc_multi_fetch
Added server successfully
Key huey stored successfully
Key dewey stored successfully
Key louie stored successfully
Key huey returned red
Key dewey returned blue
Key louie returned green

```

15.6.3.3.5. Controlling `libmemcached` Behaviors

The behavior of `libmemcached` can be modified by setting one or more behavior flags. These can either be set globally, or they can be applied during the call to individual functions. Some behaviors also accept an additional setting, such as the hashing mechanism used when selecting servers.

To set global behaviors:

```

memcached_return
    memcached_behavior_set (memcached_st *ptr,
                            memcached_behavior flag,
                            uint64_t data);

```

To get the current behavior setting:

```

uint64_t
    memcached_behavior_get (memcached_st *ptr,
                            memcached_behavior flag);

```

Behavior	Description
<code>MEMCACHED_BEHAVIOR_NO_BLOCK</code>	Caused <code>libmemcached</code> to use asynchronous I/O.
<code>MEMCACHED_BEHAVIOR_TCP_NODELAY</code>	Turns on no-delay for network sockets.

Behavior	Description
<code>MEMCACHED_BEHAVIOR_HASH</code>	Without a value, sets the default hashing algorithm for keys to use MD5. Other valid values include <code>MEMCACHED_HASH_DEFAULT</code> , <code>MEMCACHED_HASH_MD5</code> , <code>MEMCACHED_HASH_CRC</code> , <code>MEMCACHED_HASH_FNV1_64</code> , <code>MEMCACHED_HASH_FNV1A_64</code> , <code>MEMCACHED_HASH_FNV1_32</code> , and <code>MEMCACHED_HASH_FNV1A_32</code> .
<code>MEMCACHED_BEHAVIOR_DISTRIBUTION</code>	Changes the method of selecting the server used to store a given value. The default method is <code>MEMCACHED_DISTRIBUTION_MODULA</code> . You can enable consistent hashing by setting <code>MEMCACHED_DISTRIBUTION_CONSISTENT</code> . <code>MEMCACHED_DISTRIBUTION_CONSISTENT</code> is an alias for the value <code>MEMCACHED_DISTRIBUTION_CONSISTENT_KETAMA</code> .
<code>MEMCACHED_BEHAVIOR_CACHE_LOOKUPS</code>	Cache the lookups made to the DNS service. This can improve the performance if you are using names instead of IP addresses for individual hosts.
<code>MEMCACHED_BEHAVIOR_SUPPORT_CAS</code>	Support CAS operations. By default, this is disabled because it imposes a performance penalty.
<code>MEMCACHED_BEHAVIOR_KETAMA</code>	Sets the default distribution to <code>MEMCACHED_DISTRIBUTION_CONSISTENT_KETAMA</code> and the hash to <code>MEMCACHED_HASH_MD5</code> .
<code>MEMCACHED_BEHAVIOR_POLL_TIMEOUT</code>	Modify the timeout value used by <code>poll()</code> . Supply a <code>signed int</code> pointer for the timeout value.
<code>MEMCACHED_BEHAVIOR_BUFFER_REQUESTS</code>	Buffers IO requests instead of them being sent. A get operation, or closing the connection causes the data to be flushed.
<code>MEMCACHED_BEHAVIOR_VERIFY_KEY</code>	Forces <code>libmemcached</code> to verify that a specified key is valid.
<code>MEMCACHED_BEHAVIOR_SORT_HOSTS</code>	If set, hosts added to the list of configured hosts for a <code>memcached_st</code> structure are placed into the host list in sorted order. This breaks consistent hashing if that behavior has been enabled.
<code>MEMCACHED_BEHAVIOR_CONNECT_TIMEOUT</code>	In nonblocking mode this changes the value of the timeout during socket connection.

15.6.3.3.6. libmemcached Command-Line Utilities

In addition to the main C library interface, `libmemcached` also includes a number of command-line utilities that can be useful when working with and debugging `memcached` applications.

All of the command-line tools accept a number of arguments, the most critical of which is `servers`, which specifies the list of servers to connect to when returning information.

The main tools are:

- `memcat`: Display the value for each ID given on the command line:

```
shell> memcat --servers=localhost hwkey
Hello world
```

- `memcp`: Copy the contents of a file into the cache, using the file name as the key:

```
shell> echo "Hello World" > hwkey
shell> memcp --servers=localhost hwkey
shell> memcat --servers=localhost hwkey
Hello world
```

- `memrm`: Remove an item from the cache:

```
shell> memcat --servers=localhost hwkey
Hello world
shell> memrm --servers=localhost hwkey
shell> memcat --servers=localhost hwkey
```

- `memslap`: Test the load on one or more `memcached` servers, simulating get/set and multiple client operations. For example, you can simulate the load of 100 clients performing get operations:

```
shell> memslap --servers=localhost --concurrency=100 --flush --test=get
memslap --servers=localhost --concurrency=100 --flush --test=get Threads connecting to servers 100
Took 13.571 seconds to read data
```

- `memflush`: Flush (empty) the contents of the `memcached` cache.

```
shell> memflush --servers=localhost
```

15.6.3.4. Using MySQL and `memcached` with Perl

The `Cache::Memcached` module provides a native interface to the Memcache protocol, and provides support for the core functions offered by `memcached`. Install the module using your operating system's package management system, or using [CPAN](#):

```
root-shell> perl -MCPAN -e 'install Cache::Memcached'
```

To use `memcached` from Perl through the `Cache::Memcached` module, first create a new `Cache::Memcached` object that defines the list of servers and other parameters for the connection. The only argument is a hash containing the options for the cache interface. For example, to create a new instance that uses three `memcached` servers:

```
use Cache::Memcached;

my $cache = new Cache::Memcached {
    'servers' => [
        '192.168.0.100:11211',
        '192.168.0.101:11211',
        '192.168.0.102:11211',
    ],
};
```



Note

When using the `Cache::Memcached` interface with multiple servers, the API automatically performs certain operations across all the servers in the group. For example, getting statistical information through `Cache::Memcached` returns a hash that contains data on a host-by-host basis, as well as generalized statistics for all the servers in the group.

You can set additional properties on the cache object instance when it is created by specifying the option as part of the option hash. Alternatively, you can use a corresponding method on the instance:

- `servers` or method `set_servers()`: Specifies the list of the servers to be used. The servers list should be a reference to an array of servers, with each element as the address and port number combination (separated by a colon). You can also specify a local connection through a UNIX socket

(for example `/tmp/sock/memcached`). To specify the server with a weight (indicating how much more frequently the server should be used during hashing), specify an array reference with the `memcached` server instance and a weight number. Higher numbers give higher priority.

- `compress_threshold` or method `set_compress_threshold()`: Specifies the threshold when values are compressed. Values larger than the specified number are automatically compressed (using `zlib`) during storage and retrieval.
- `no_rehash` or method `set_norehash()`: Disables finding a new server if the original choice is unavailable.
- `readonly` or method `set_readonly()`: Disables writes to the `memcached` servers.

Once the `Cache::Memcached` object instance has been configured, you can use the `set()` and `get()` methods to store and retrieve information from the `memcached` servers. Objects stored in the cache are automatically serialized and deserialized using the `Storable` module.

The `Cache::Memcached` interface supports the following methods for storing/retrieving data, and relate to the generic methods as shown in the table.

<code>Cache::Memcached</code> Function	Equivalent to
<code>get()</code>	Generic <code>get()</code> .
<code>get_multi(keys)</code>	Gets multiple <code>keys</code> from memcache using just one query. Returns a hash reference of key/value pairs.
<code>set()</code>	Generic <code>set()</code> .
<code>add()</code>	Generic <code>add()</code> .
<code>replace()</code>	Generic <code>replace()</code> .
<code>delete()</code>	Generic <code>delete()</code> .
<code>incr()</code>	Generic <code>incr()</code> .
<code>decr()</code>	Generic <code>decr()</code> .

Below is a complete example for using `memcached` with Perl and the `Cache::Memcached` module:

```
#!/usr/bin/perl

use Cache::Memcached;
use DBI;
use Data::Dumper;

# Configure the memcached server

my $cache = new Cache::Memcached {
    'servers' => [
        'localhost:11211',
    ],
};

# Get the film name from the command line
# memcached keys must not contain spaces, so create
# a key name by replacing spaces with underscores

my $filename = shift or die "Must specify the film name\n";
my $filmkey = $filename;
$filmkey =~ s/ /_/;

# Load the data from the cache
```

```

my $filmdata = $cache->get($filmkey);

# If the data wasn't in the cache, then we load it from the database
if (!defined($filmdata))
{
    $filmdata = load_filmdata($filmname);

    if (defined($filmdata))
    {
# Set the data into the cache, using the key

        if ($cache->set($filmkey,$filmdata))
            {
                print STDERR "Film data loaded from database and cached\n";
            }
            else
            {
                print STDERR "Couldn't store to cache\n";
            }
        }
        else
        {
            die "Couldn't find $filmname\n";
        }
    }
}
else
{
    print STDERR "Film data loaded from Memcached\n";
}

sub load_filmdata
{
    my ($filmname) = @_ ;

    my $dsn = "DBI:mysql:database=sakila;host=localhost;port=3306";

    $dbh = DBI->connect($dsn, 'sakila','password');

    my ($filmbase) = $dbh->selectrow_hashref(sprintf('select * from film where title = %s',
                                                    $dbh->quote($filmname)));

    if (!defined($filmname))
    {
        return (undef);
    }

    $filmbase->{stars} =
    $dbh->selectall_arrayref(sprintf('select concat(first_name," ",last_name) ' .
                                    'from film_actor left join (actor) ' .
                                    'on (film_actor.actor_id = actor.actor_id) ' .
                                    ' where film_id=%s',
                                    $dbh->quote($filmbase->{film_id})));

    return($filmbase);
}

```

The example uses the Sakila database, obtaining film data from the database and writing a composite record of the film and actors to `memcached`. When calling it for a film does not exist, you get this result:

```

shell> memcached-sakila.pl "ROCK INSTINCT"
Film data loaded from database and cached

```

When accessing a film that has already been added to the cache:

```

shell> memcached-sakila.pl "ROCK INSTINCT"

```

Film data loaded from Memcached

15.6.3.5. Using MySQL and `memcached` with Python

The Python `memcache` module interfaces to `memcached` servers, and is written in pure Python (that is, without using one of the C APIs). You can download and install a copy from [Python Memcached](#).

To install, download the package and then run the Python installer:

```
python setup.py install
running install
running bdist_egg
running egg_info
creating python_memcached.egg-info
...
removing 'build/bdist.linux-x86_64/egg' (and everything under it)
Processing python_memcached-1.43-py2.4.egg
creating /usr/lib64/python2.4/site-packages/python_memcached-1.43-py2.4.egg
Extracting python_memcached-1.43-py2.4.egg to /usr/lib64/python2.4/site-packages
Adding python-memcached 1.43 to easy-install.pth file

Installed /usr/lib64/python2.4/site-packages/python_memcached-1.43-py2.4.egg
Processing dependencies for python-memcached==1.43
Finished processing dependencies for python-memcached==1.43
```

Once installed, the `memcache` module provides a class-based interface to your `memcached` servers. When you store Python data structures as `memcached` items, they are automatically serialized (turned into string values) using the Python `cPickle` or `pickle` modules.

To create a new `memcache` interface, import the `memcache` module and create a new instance of the `memcache.Client` class. For example, if the `memcached` daemon is running on localhost using the default port:

```
import memcache
memc = memcache.Client(['127.0.0.1:11211'])
```

The first argument is an array of strings containing the server and port number for each `memcached` instance to use. To enable debugging, set the optional `debug` parameter to 1.

By default, the hashing mechanism used to divide the items among multiple servers is `crc32`. To change the function used, set the value of `memcache.serverHashFunction` to the alternate function to use. For example:

```
from zlib import Adler32
memcache.serverHashFunction = Adler32
```

Once you have defined the servers to use within the `memcache` instance, the core functions provide the same functionality as in the generic interface specification. The following table provides a summary of the supported functions:

Python <code>memcache</code> Function	Equivalent to
<code>get()</code>	Generic <code>get()</code> .
<code>get_multi(keys)</code>	Gets multiple values from the supplied array of <code>keys</code> . Returns a hash reference of key/value pairs.
<code>set()</code>	Generic <code>set()</code> .
<code>set_multi(dict [, expiry [, key_prefix]])</code>	Sets multiple key/value pairs from the supplied <code>dict</code> .

Python <code>memcache</code> Function	Equivalent to
<code>add()</code>	Generic <code>add()</code> .
<code>replace()</code>	Generic <code>replace()</code> .
<code>prepend(key, value [, expiry])</code>	Prepends the supplied <code>value</code> to the value of the existing <code>key</code> .
<code>append(key, value [, expiry])</code>	Appends the supplied <code>value</code> to the value of the existing <code>key</code> .
<code>delete()</code>	Generic <code>delete()</code> .
<code>delete_multi(keys [, expiry [, key_prefix]])</code>	Deletes all the keys from the hash matching each string in the array <code>keys</code> .
<code>incr()</code>	Generic <code>incr()</code> .
<code>decr()</code>	Generic <code>decr()</code> .



Note

Within the Python `memcache` module, all the `*_multi()` functions support an optional `key_prefix` parameter. If supplied, then the string is used as a prefix to all key lookups. For example, if you call:

```
memc.get_multi(['a','b'], key_prefix='users:')
```

The function retrieves the keys `users:a` and `users:b` from the servers.

Here is an example showing the storage and retrieval of information to a `memcache` instance, loading the raw data from MySQL:

```
import sys
import MySQLdb
import memcache

memc = memcache.Client(['127.0.0.1:11211'], debug=1);

try:
    conn = MySQLdb.connect (host = "localhost",
                            user = "sakila",
                            passwd = "password",
                            db = "sakila")
except MySQLdb.Error, e:
    print "Error %d: %s" % (e.args[0], e.args[1])
    sys.exit (1)

popularfilms = memc.get('top5films')

if not popularfilms:
    cursor = conn.cursor()
    cursor.execute('select film_id,title from film order by rental_rate desc limit 5')
    rows = cursor.fetchall()
    memc.set('top5films',rows,60)
    print "Updated memcached with MySQL data"
else:
    print "Loaded data from memcached"
    for row in popularfilms:
        print "%s, %s" % (row[0], row[1])
```

When executed for the first time, the data is loaded from the MySQL database and stored to the `memcached` server.

```
shell> python memc_python.py
```

```
Updated memcached with MySQL data
```

Because the data is automatically serialized using `cPickle/pickle`, when you load the data back from `memcached`, you can use the object directly. In the example above, the information stored to `memcached` is in the form of rows from a Python DB cursor. When accessing the information (within the 60 second expiry time), the data is loaded from `memcached` and dumped:

```
shell> python memc_python.py
Loaded data from memcached
2, ACE GOLDFINGER
7, AIRPLANE SIERRA
8, AIRPORT POLLOCK
10, ALADDIN CALENDAR
13, ALI FOREVER
```

The serialization and deserialization happens automatically. Because serialization of Python data may be incompatible with other interfaces and languages, you can change the serialization module used during initialization. For example, you might use JSON format when you store complex data structures using a script written in one language, and access them in a script written in a different language.

15.6.3.6. Using MySQL and memcached with PHP

PHP provides support for the Memcache functions through a PECL extension. To enable the PHP `memcache` extensions, build PHP using the `--enable-memcache` option to `configure` when building from source.

If you are installing on a Red Hat-based server, you can install the `php-pecl-memcache` RPM:

```
root-shell> yum --install php-pecl-memcache
```

On Debian-based distributions, use the `php-memcache` package.

To set global runtime configuration options, specify the values in the following table within your `php.ini` file:

Configuration option	Default	Description
<code>memcache.allow_failover</code>	1	Specifies whether another server in the list should be queried if the first server selected fails.
<code>memcache.max_failover_attempts</code>	20	Specifies the number of servers to try before returning a failure.
<code>memcache.chunk_size</code>	8192	Defines the size of network chunks used to exchange data with the <code>memcached</code> server.
<code>memcache.default_port</code>	11211	Defines the default port to use when communicating with the <code>memcached</code> servers.
<code>memcache.hash_strategy</code>	standard	Specifies which hash strategy to use. Set to <code>consistent</code> to enable servers to be added or removed from the pool without causing the keys to be remapped to other servers. When set to <code>standard</code> , an older (modula) strategy is used that potentially uses different servers for storage.
<code>memcache.hash_function</code>	crc32	Specifies which function to use when mapping keys to servers. <code>crc32</code> uses the standard

Configuration option	Default	Description
		CRC32 hash. <code>fnv</code> uses the FNV-1a hashing algorithm.

To create a connection to a `memcached` server, create a new `Memcache` object and then specify the connection options. For example:

```
<?php
$cache = new Memcache;
$cache->connect('localhost',11211);
?>
```

This opens an immediate connection to the specified server.

To use multiple `memcached` servers, you need to add servers to the memcache object using `addServer()`:

```
bool Memcache::addServer ( string $host [, int $port [, bool $persistent
                        [, int $weight [, int $timeout [, int $retry_interval
                        [, bool $status [, callback $failure_callback
                        ]]]]] ] )
```

The server management mechanism within the `php-memcache` module is a critical part of the interface as it controls the main interface to the `memcached` instances and how the different instances are selected through the hashing mechanism.

To create a simple connection to two `memcached` instances:

```
<?php
$cache = new Memcache;
$cache->addServer('192.168.0.100',11211);
$cache->addServer('192.168.0.101',11211);
?>
```

In this scenario, the instance connection is not explicitly opened, but only opened when you try to store or retrieve a value. To enable persistent connections to `memcached` instances, set the `$persistent` argument to true. This is the default setting, and causes the connections to remain open.

To help control the distribution of keys to different instances, use the global `memcache.hash_strategy` setting. This sets the hashing mechanism used to select. You can also add another weight to each server, which effectively increases the number of times the instance entry appears in the instance list, therefore increasing the likelihood of the instance being chosen over other instances. To set the weight, set the value of the `$weight` argument to more than one.

The functions for setting and retrieving information are identical to the generic functional interface offered by `memcached`, as shown in this table:

PECL <code>memcache</code> Function	Equivalent to
<code>get()</code>	Generic <code>get()</code> .
<code>set()</code>	Generic <code>set()</code> .
<code>add()</code>	Generic <code>add()</code> .
<code>replace()</code>	Generic <code>replace()</code> .

PECL <code>memcache</code> Function	Equivalent to
<code>delete()</code>	Generic <code>delete()</code> .
<code>increment()</code>	Generic <code>incr()</code> .
<code>decrement()</code>	Generic <code>decr()</code> .

A full example of the PECL `memcache` interface is provided below. The code loads film data from the Sakila database when the user provides a film name. The data stored into the `memcached` instance is recorded as a `mysqli` result row, and the API automatically serializes the information for you.

```
<?php

$memc = new Memcache;
$memc->addServer('localhost', '11211');

if(empty($_POST['film'])) {
?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Simple Memcache Lookup</title>
</head>
<body>
<form method="post">
<p><b>Film</b>: <input type="text" size="20" name="film"></p>
<input type="submit">
</form>
<hr/>
<?php
} else {

    echo "Loading data...\n";

    $film = htmlspecialchars($_POST['film'], ENT_QUOTES, 'UTF-8');
    $mfilms = $memc->get($film);

    if ($mfilms) {

        printf("<p>Film data for %s loaded from memcache</p>", $mfilms['title']);

        foreach (array_keys($mfilms) as $key) {
            printf("<p><b>%s</b>: %s</p>", $key, $mfilms[$key]);
        }

    } else {

        $mysqli = mysqli('localhost', 'sakila', 'password', 'sakila');

        if (mysqli_connect_error()) {
            printf("Database error: (%d) %s", mysqli_connect_errno(), mysqli_connect_error());
            exit;
        }

        $sql = sprintf('SELECT * FROM film WHERE title="%s"', $mysqli->real_escape_string($film));

        $result = $mysqli->query($sql);

        if (!$result) {
            printf("Database error: (%d) %s", $mysqli->errno, $mysqli->error);
            exit;
        }

        $row = $result->fetch_assoc();
```

```

        $memc->set($row['title'], $row);

        printf("<p>Loaded (%s) from MySQL</p>", htmlspecialchars($row['title'], ENT_QUOTES, 'UTF-8'));
    }
}
?>
</body>
</html>

```

With PHP, the connections to the `memcached` instances are kept open as long as the PHP and associated Apache instance remain running. When adding or removing servers from the list in a running instance (for example, when starting another script that mentions additional servers), the connections are shared, but the script only selects among the instances explicitly configured within the script.

To ensure that changes to the server list within a script do not cause problems, make sure to use the consistent hashing mechanism.

15.6.3.7. Using MySQL and `memcached` with Ruby

There are a number of different modules for interfacing to `memcached` within Ruby. The `Ruby-MemCache` client library provides a native interface to `memcached` that does not require any external libraries, such as `libmemcached`. You can obtain the installer package from <http://www.deveiate.org/projects/RMemCache>.

To install, extract the package and then run `install.rb`:

```
shell> install.rb
```

If you have RubyGems, you can install the `Ruby-MemCache` gem:

```

shell> gem install Ruby-MemCache
Bulk updating Gem source index for: http://gems.rubyforge.org
Install required dependency io-reactor? [Yn] y
Successfully installed Ruby-MemCache-0.0.1
Successfully installed io-reactor-0.05
Installing ri documentation for io-reactor-0.05...
Installing RDoc documentation for io-reactor-0.05...

```

To use a `memcached` instance from within Ruby, create a new instance of the `MemCache` object.

```

require 'memcache'
memc = MemCache::new '192.168.0.100:11211'

```

You can add a weight to each server to increase the likelihood of the server being selected during hashing by appending the weight count to the server host name/port string:

```

require 'memcache'
memc = MemCache::new '192.168.0.100:11211:3'

```

To add servers to an existing list, you can append them directly to the `MemCache` object:

```
memc += ["192.168.0.101:11211"]
```

To set data into the cache, you can just assign a value to a key within the new cache object, which works just like a standard Ruby hash object:

```
memc["key"] = "value"
```

Or to retrieve the value:

```
print memc["key"]
```

For more explicit actions, you can use the method interface, which mimics the main `memcached` API functions, as summarized in the following table:

Ruby <code>MemCache</code> Method	Equivalent to
<code>get()</code>	Generic <code>get()</code> .
<code>get_hash(keys)</code>	Get the values of multiple <code>keys</code> , returning the information as a hash of the keys and their values.
<code>set()</code>	Generic <code>set()</code> .
<code>set_many(pairs)</code>	Set the values of the keys and values in the hash <code>pairs</code> .
<code>add()</code>	Generic <code>add()</code> .
<code>replace()</code>	Generic <code>replace()</code> .
<code>delete()</code>	Generic <code>delete()</code> .
<code>incr()</code>	Generic <code>incr()</code> .
<code>decr()</code>	Generic <code>decr()</code> .

15.6.3.8. Using MySQL and `memcached` with Java

The `com.danga.MemCached` class within Java provides a native interface to `memcached` instances. You can obtain the client from <http://whalin.com/memcached/>. The Java class uses hashes that are compatible with `libmemcached`, so you can mix and match Java and `libmemcached` applications accessing the same `memcached` instances. The serialization between Java and other interfaces are not compatible. If this is a problem, use JSON or a similar nonbinary serialization format.

On most systems, you can download the package and use the `jar` directly.

To use the `com.danga.MemCached` interface, you create a `MemCachedClient` instance and then configure the list of servers by configuring the `SockIOPool`. Through the pool specification you set up the server list, weighting, and the connection parameters to optimized the connections between your client and the `memcached` instances that you configure.

Generally, you can configure the `memcached` interface once within a single class, then use this interface throughout the rest of your application.

For example, to create a basic interface, first configure the `MemCachedClient` and base `SockIOPool` settings:

```
public class MyClass {
    protected static MemCachedClient mcc = new MemCachedClient();

    static {
        String[] servers =
            {
                "localhost:11211",
            };

        Integer[] weights = { 1 };

        SockIOPool pool = SockIOPool.getInstance();
    }
}
```

```
pool.setServers( servers );
pool.setWeights( weights );
```

In the above sample, the list of servers is configured by creating an array of the `memcached` instances to use. You can then configure individual weights for each server.

The remainder of the properties for the connection are optional, but you can set the connection numbers (initial connections, minimum connections, maximum connections, and the idle timeout) by setting the pool parameters:

```
pool.setInitConn( 5 );
pool.setMinConn( 5 );
pool.setMaxConn( 250 );
pool.setMaxIdle( 1000 * 60 * 60 * 6
```

Once the parameters have been configured, initialize the connection pool:

```
pool.initialize();
```

The pool, and the connection to your `memcached` instances should now be ready to use.

To set the hashing algorithm used to select the server used when storing a given key, use `pool.setHashingAlg()`:

```
pool.setHashingAlg( SockIOPool.NEW_COMPAT_HASH );
```

Valid values are `NEW_COMPAT_HASH`, `OLD_COMPAT_HASH` and `NATIVE_HASH` are also basic modular hashing algorithms. For a consistent hashing algorithm, use `CONSISTENT_HASH`. These constants are equivalent to the corresponding hash settings within `libmemcached`.

Java <code>com.danga.MemCached</code> Method	Equivalent to
<code>get()</code>	Generic <code>get()</code> .
<code>getMulti(keys)</code>	Get the values of multiple <code>keys</code> , returning the information as Hash map using <code>java.lang.String</code> for the keys and <code>java.lang.Object</code> for the corresponding values.
<code>set()</code>	Generic <code>set()</code> .
<code>add()</code>	Generic <code>add()</code> .
<code>replace()</code>	Generic <code>replace()</code> .
<code>delete()</code>	Generic <code>delete()</code> .
<code>incr()</code>	Generic <code>incr()</code> .
<code>decr()</code>	Generic <code>decr()</code> .

15.6.3.9. Using the MySQL `memcached` User-Defined Functions

The `memcached` MySQL User-Defined Functions (UDFs) enable you to set and retrieve objects through a SQL interface, in MySQL 5.0 or greater. This technique has the following benefits:

- You can update and retrieve cached items directly from within SQL scripts, stored procedures, and triggers. For example, you might already have triggers in place to increment counters and set status flags based on general database activity.
- You can pass information back and forth with other applications that use `memcached`, without adding database connection and query code to them.

- You can avoid installing and configuring a `memcached` client on every combination of language and server that you use for your MySQL applications. The applications can relay `memcached` requests through the database server they connect to.
- You can access `memcached` servers from languages that do not have a `memcached` client.

To install the MySQL `memcached` UDFs, download the UDF package from <https://launchpad.net/memcached-udfs>. Unpack the package and run `configure` to configure the build process. When running `configure`, use the `--with-mysql` option and specify the location of the `mysql_config` command.

```
shell> tar zxf memcached_functions_mysql-1.1.tar.gz
shell> cd memcached_functions_mysql-1.1
shell> # If memcached library is not found, set LDFLAGS=-Llibrary_directory before next command.
shell> ./configure --with-mysql=/usr/local/mysql/bin/mysql_config
```

Now build and install the functions:

```
shell> make
shell> make install
```

Copy the MySQL `memcached` UDFs into your MySQL plugins directory:

```
shell> cp /usr/local/lib/libmemcached_functions_mysql* /usr/local/mysql/lib/mysql/plugins/
```

The plugin directory is given by the value of the `plugin_dir` [537] system variable. For more information, see [Section 23.3.2.5, “Compiling and Installing User-Defined Functions”](#).

Once installed, you must initialize the function within MySQL using `CREATE` and specifying the return value and library. For example, to add the `memc_get()` function:

```
mysql> CREATE FUNCTION memc_get RETURNS STRING SONAME "libmemcached_functions_mysql.so";
```

Repeat this process for each function to provide access to within MySQL. Once you have created the association, the information is retained, even over restarts of the MySQL server. To simplify the process, use the SQL script provided in the `memcached` UDFs package:

```
shell> mysql <sql/install_functions.sql
```

Alternatively, if you have Perl installed, then you can use the supplied Perl script, which checks for the existence of each function and creates the function/library association if it is not already defined:

```
shell> utils/install.pl --silent
```

The `--silent` option installs everything automatically. Without this option, the script asks whether to install each of the available functions.

The interface remains consistent with the other APIs and interfaces. To set up a list of servers, use the `memc_servers_set()` function, which accepts a single string containing and comma-separated list of servers:

```
mysql> SELECT memc_servers_set('192.168.0.1:11211,192.168.0.2:11211');
```



Note

The list of servers used by the `memcached` UDFs is not persistent over restarts of the MySQL server. If the MySQL server fails, then you must re-set the list of `memcached` servers.

To set a value, use `memc_set`:

```
mysql> SELECT memc_set('myid', 'myvalue');
```

To retrieve a stored value:

```
mysql> SELECT memc_get('myid');
```

The list of functions supported by the UDFs, in relation to the standard protocol functions, is shown in the following table:

MySQL <code>memcached</code> UDF Function	Equivalent to
<code>memc_get()</code>	Generic <code>get()</code> .
<code>memc_get_by_key(master_key, key, value)</code>	Like the generic <code>get()</code> , but uses the supplied master key to select the server to use.
<code>memc_set()</code>	Generic <code>set()</code> .
<code>memc_set_by_key(master_key, key, value)</code>	Like the generic <code>set()</code> , but uses the supplied master key to select the server to use.
<code>memc_add()</code>	Generic <code>add()</code> .
<code>memc_add_by_key(master_key, key, value)</code>	Like the generic <code>add()</code> , but uses the supplied master key to select the server to use.
<code>memc_replace()</code>	Generic <code>replace()</code> .
<code>memc_replace_by_key(master_key, key, value)</code>	Like the generic <code>replace()</code> , but uses the supplied master key to select the server to use.
<code>memc_prepend(key, value)</code>	Prepend the specified <code>value</code> to the current value of the specified <code>key</code> .
<code>memc_prepend_by_key(master_key, key, value)</code>	Prepend the specified <code>value</code> to the current value of the specified <code>key</code> , but uses the supplied master key to select the server to use.
<code>memc_append(key, value)</code>	Append the specified <code>value</code> to the current value of the specified <code>key</code> .
<code>memc_append_by_key(master_key, key, value)</code>	Append the specified <code>value</code> to the current value of the specified <code>key</code> , but uses the supplied master key to select the server to use.
<code>memc_delete()</code>	Generic <code>delete()</code> .
<code>memc_delete_by_key(master_key, key, value)</code>	Like the generic <code>delete()</code> , but uses the supplied master key to select the server to use.
<code>memc_increment()</code>	Generic <code>incr()</code> .
<code>memc_decrement()</code>	Generic <code>decr()</code> .

The respective `*_by_key()` functions are useful to store a specific value into a specific `memcached` server, possibly based on a differently calculated or constructed key.

The `memcached` UDFs include some additional functions:

- `memc_server_count()`

Returns a count of the number of servers in the list of registered servers.

- `memc_servers_set_behavior(behavior_type, value)`,
`memc_set_behavior(behavior_type, value)`

Sets behaviors for the list of servers. These behaviors are identical to those provided by the `libmemcached` library. For more information on `libmemcached` behaviors, see [Section 15.6.3.3, “Using `libmemcached` with C and C++”](#).

You can use the behavior name as the `behavior_type`:

```
mysql> SELECT memc_servers_behavior_set("MEMCACHED_BEHAVIOR_KETAMA",1);
```

- `memc_servers_behavior_get(behavior_type), memc_get_behavior(behavior_type, value)`

Returns the value for a given behavior.

- `memc_list_behaviors()`

Returns a list of the known behaviors.

- `memc_list_hash_types()`

Returns a list of the supported key-hashing algorithms.

- `memc_list_distribution_types()`

Returns a list of the supported distribution types to be used when selecting a server to use when storing a particular key.

- `memc_libmemcached_version()`

Returns the version of the `libmemcached` library.

- `memc_stats()`

Returns the general statistics information from the server.

15.6.3.10. Using the `memcached` TCP Text Protocol

Communicating with a `memcached` server can be achieved through either the TCP or UDP protocols. When using the TCP protocol, you can use a simple text based interface for the exchange of information.

When communicating with `memcached`, you can connect to the server using the port configured for the server. You can open a connection with the server without requiring authorization or login. As soon as you have connected, you can start to send commands to the server. When you have finished, you can terminate the connection without sending any specific disconnection command. Clients are encouraged to keep their connections open to decrease latency and improve performance.

Data is sent to the `memcached` server in two forms:

- Text lines, which are used to send commands to the server, and receive responses from the server.
- Unstructured data, which is used to receive or send the value information for a given key. Data is returned to the client in exactly the format it was provided.

Both text lines (commands and responses) and unstructured data are always terminated with the string `\r\n`. Because the data being stored may contain this sequence, the length of the data (returned by the client before the unstructured data is transmitted should be used to determine the end of the data.

Commands to the server are structured according to their operation:

- **Storage commands:** `set`, `add`, `replace`, `append`, `prepend`, `cas`

Storage commands to the server take the form:

```
command key [flags] [exptime] length [noreply]
```

Or when using compare and swap (`cas`):

```
cas key [flags] [exptime] length [casunique] [noreply]
```

Where:

- `command`: The command name.
 - `set`: Store value against key
 - `add`: Store this value against key if the key does not already exist
 - `replace`: Store this value against key if the key already exists
 - `append`: Append the supplied value to the end of the value for the specified key. The `flags` and `exptime` arguments should not be used.
 - `prepend`: Append value currently in the cache to the end of the supplied value for the specified key. The `flags` and `exptime` arguments should not be used.
 - `cas`: Set the specified key to the supplied value, only if the supplied `casunique` matches. This is effectively the equivalent of change the information if nobody has updated it since I last fetched it.
- `key`: The key. All data is stored using a the specific key. The key cannot contain control characters or whitespace, and can be up to 250 characters in size.
- `flags`: The flags for the operation (as an integer). Flags in `memcached` are transparent. The `memcached` server ignores the contents of the flags. They can be used by the client to indicate any type of information. In `memcached` 1.2.0 and lower the value is a 16-bit integer value. In `memcached` 1.2.1 and higher the value is a 32-bit integer.
- `exptime`: The expiry time, or zero for no expiry.
- `length`: The length of the supplied value block in bytes, excluding the terminating `\r\n` characters.
- `casunique`: A unique 64-bit value of an existing entry. This is used to compare against the existing value. Use the value returned by the `gets` command when issuing `cas` updates.
- `noreply`: Tells the server not to reply to the command.

For example, to store the value `abcdef` into the key `xyzkey`, you would use:

```
set xyzkey 0 0 6\r\nabcdef\r\n
```

The return value from the server is one line, specifying the status or error information. For more information, see [Table 15.3, “memcached Protocol Responses”](#).

- **Retrieval commands:** `get`, `gets`

Retrieval commands take the form:

```
get key1 [key2 ... keyn]
gets key1 [key2 ... keyn]
```

You can supply multiple keys to the commands, with each requested key separated by whitespace.

The server responds with an information line of the form:

```
VALUE key flags bytes [casunique]
```

Where:

- `key`: The key name.
- `flags`: The value of the flag integer supplied to the `memcached` server when the value was stored.
- `bytes`: The size (excluding the terminating `\r\n` character sequence) of the stored value.
- `casunique`: The unique 64-bit integer that identifies the item.

The information line is immediately followed by the value data block. For example:

```
get xyzkey\r\n
VALUE xyzkey 0 6\r\n
abcdef\r\n
```

If you have requested multiple keys, an information line and data block is returned for each key found. If a requested key does not exist in the cache, no information is returned.

- **Delete commands:** `delete`

Deletion commands take the form:

```
delete key [time] [noreply]
```

Where:

- `key`: The key name.
- `time`: The time in seconds (or a specific Unix time) for which the client wishes the server to refuse `add` or `replace` commands on this key. All `add`, `replace`, `get`, and `gets` commands fail during this period. `set` operations succeed. After this period, the key is deleted permanently and all commands are accepted.

If not supplied, the value is assumed to be zero (delete immediately).

- `noreply`: Tells the server not to reply to the command.

Responses to the command are either `DELETED` to indicate that the key was successfully removed, or `NOT_FOUND` to indicate that the specified key could not be found.

- **Increment/Decrement:** `incr`, `decr`

The increment and decrement commands change the value of a key within the server without performing a separate `get/set` sequence. The operations assume that the currently stored value is a 64-bit integer. If the stored value is not a 64-bit integer, then the value is assumed to be zero before the increment or decrement operation is applied.

Increment and decrement commands take the form:

```
incr key value [noreply]
decr key value [noreply]
```

Where:

- `key`: The key name.
- `value`: An integer to be used as the increment or decrement value.
- `noreply`: Tells the server not to reply to the command.

The response is:

- `NOT_FOUND`: The specified key could not be located.
- `value`: The new value associated with the specified key.

Values are assumed to be unsigned. For `decr` operations, the value is never decremented below 0. For `incr` operations, the value wraps around the 64-bit maximum.

- **Statistics commands:** `stats`

The `stats` command provides detailed statistical information about the current status of the `memcached` instance and the data it is storing.

Statistics commands take the form:

```
STAT [name] [value]
```

Where:

- `name`: The optional name of the statistics to return. If not specified, the general statistics are returned.
- `value`: A specific value to be used when performing certain statistics operations.

The return value is a list of statistics data, formatted as follows:

```
STAT name value
```

The statistics are terminated with a single line, `END`.

For more information, see [Section 15.6.4, "Getting memcached Statistics"](#).

For reference, a list of the different commands supported and their formats is provided below.

Table 15.2. `memcached` Command Reference

Command	Command Formats
<code>set</code>	<code>set key flags exptime length, set key flags exptime length noreply</code>
<code>add</code>	<code>add key flags exptime length, add key flags exptime length noreply</code>
<code>replace</code>	<code>replace key flags exptime length, replace key flags exptime length noreply</code>
<code>append</code>	<code>append key length, append key length noreply</code>
<code>prepend</code>	<code>prepend key length, prepend key length noreply</code>
<code>cas</code>	<code>cas key flags exptime length casunique, cas key flags exptime length casunique noreply</code>

Command	Command Formats
<code>get</code>	<code>get key1 [key2 ... keyn]</code>
<code>gets</code>	
<code>delete</code>	<code>delete key, delete key noreply, delete key expiry, delete key expiry noreply</code>
<code>incr</code>	<code>incr key, incr key noreply, incr key value, incr key value noreply</code>
<code>decr</code>	<code>decr key, decr key noreply, decr key value, decr key value noreply</code>
<code>stat</code>	<code>stat, stat name, stat name value</code>

When sending a command to the server, the response from the server is one of the settings in the following table. All response values from the server are terminated by `\r\n`:

Table 15.3. memcached Protocol Responses

String	Description
<code>STORED</code>	Value has successfully been stored.
<code>NOT_STORED</code>	The value was not stored, but not because of an error. For commands where you are adding a or updating a value if it exists (such as <code>add</code> and <code>replace</code>), or where the item has already been set to be deleted.
<code>EXISTS</code>	When using a <code>cas</code> command, the item you are trying to store already exists and has been modified since you last checked it.
<code>NOT_FOUND</code>	The item you are trying to store, update or delete does not exist or has already been deleted.
<code>ERROR</code>	You submitted a nonexistent command name.
<code>CLIENT_ERROR</code> <code>errorstring</code>	There was an error in the input line, the detail is contained in <code>errorstring</code> .
<code>SERVER_ERROR</code> <code>errorstring</code>	There was an error in the server that prevents it from returning the information. In extreme conditions, the server may disconnect the client after this error occurs.
<code>VALUE</code> <code>keys</code> <code>flags</code> <code>length</code>	The requested key has been found, and the stored <code>key</code> , <code>flags</code> and data block are returned, of the specified <code>length</code> .
<code>DELETED</code>	The requested key was deleted from the server.
<code>STAT</code> <code>name</code> <code>value</code>	A line of statistics data.
<code>END</code>	The end of the statistics data.

15.6.4. Getting memcached Statistics

The `memcached` system has a built-in statistics system that collects information about the data being stored into the cache, cache hit ratios, and detailed information on the memory usage and distribution of information through the slab allocation used to store individual items. Statistics are provided at both a basic level that provide the core statistics, and more specific statistics for specific areas of the `memcached` server.

This information can be useful to ensure that you are getting the correct level of cache and memory usage, and that your slab allocation and configuration properties are set at an optimal level.

The stats interface is available through the standard `memcached` protocol, so the reports can be accessed by using `telnet` to connect to the `memcached`. The supplied `memcached-tool` includes support for obtaining the [Section 15.6.4.2, “memcached Slabs Statistics”](#) and [Section 15.6.4.1, “memcached General Statistics”](#) information. For more information, see [Section 15.6.4.6, “Using memcached-tool”](#).

Alternatively, most of the language API interfaces provide a function for obtaining the statistics from the server.

For example, to get the basic stats using `telnet`:

```
shell> telnet localhost 11211
Trying ::1...
Connected to localhost.
Escape character is '^]'.
stats
STAT pid 23599
STAT uptime 675
STAT time 1211439587
STAT version 1.2.5
STAT pointer_size 32
STAT rusage_user 1.404992
STAT rusage_system 4.694685
STAT curr_items 32
STAT total_items 56361
STAT bytes 2642
STAT curr_connections 53
STAT total_connections 438
STAT connection_structures 55
STAT cmd_get 113482
STAT cmd_set 80519
STAT get_hits 78926
STAT get_misses 34556
STAT evictions 0
STAT bytes_read 6379783
STAT bytes_written 4860179
STAT limit_maxbytes 67108864
STAT threads 1
END
```

When using Perl and the `Cache::Memcached` module, the `stats()` function returns information about all the servers currently configured in the connection object, and total statistics for all the `memcached` servers as a whole.

For example, the following Perl script obtains the stats and dumps the hash reference that is returned:

```
use Cache::Memcached;
use Data::Dumper;

my $memc = new Cache::Memcached;
$memc->set_servers(\@ARGV);

print Dumper($memc->stats());
```

When executed on the same `memcached` as used in the `Telnet` example above we get a hash reference with the host by host and total statistics:

```
$VAR1 = {
  'hosts' => {
    'localhost:11211' => {
      'misc' => {
        'bytes' => '2421',
        'curr_connections' => '3',
        'connection_structures' => '56',
        'pointer_size' => '32',
        'time' => '1211440166',
```

```

        'total_items' => '410956',
        'cmd_set' => '588167',
        'bytes_written' => '35715151',
        'evictions' => '0',
        'curr_items' => '31',
        'pid' => '23599',
        'limit_maxbytes' => '67108864',
        'uptime' => '1254',
        'rusage_user' => '9.857805',
        'cmd_get' => '838451',
        'rusage_system' => '34.096988',
        'version' => '1.2.5',
        'get_hits' => '581511',
        'bytes_read' => '46665716',
        'threads' => '1',
        'total_connections' => '3104',
        'get_misses' => '256940'
    },
    'sizes' => {
        '128' => '16',
        '64' => '15'
    }
},
'self' => {},
'total' => {
    'cmd_get' => 838451,
    'bytes' => 2421,
    'get_hits' => 581511,
    'connection_structures' => 56,
    'bytes_read' => 46665716,
    'total_items' => 410956,
    'total_connections' => 3104,
    'cmd_set' => 588167,
    'bytes_written' => 35715151,
    'curr_items' => 31,
    'get_misses' => 256940
}
};

```

The statistics are divided up into a number of distinct sections, and then can be requested by adding the type to the `stats` command. Each statistics output is covered in more detail in the following sections.

- General statistics, see [Section 15.6.4.1, “memcached General Statistics”](#).
- Slab statistics (`slabs`), see [Section 15.6.4.2, “memcached Slabs Statistics”](#).
- Item statistics (`items`), see [Section 15.6.4.3, “memcached Item Statistics”](#).
- Size statistics (`sizes`), see [Section 15.6.4.4, “memcached Size Statistics”](#).
- Detailed status (`detail`), see [Section 15.6.4.5, “memcached Detail Statistics”](#).

15.6.4.1. memcached General Statistics

The output of the general statistics provides an overview of the performance and use of the `memcached` instance. The statistics returned by the command and their meaning is shown in the following table.

The following terms are used to define the value type for each statistics value:

- `32u`: 32-bit unsigned integer
- `64u`: 64-bit unsigned integer
- `32u:32u`: Two 32-bit unsigned integers separated by a colon

- `String`: Character string

Statistic	Data type	Description	Version
<code>pid</code>	32u	Process ID of the <code>memcached</code> instance.	
<code>uptime</code>	32u	Uptime (in seconds) for this <code>memcached</code> instance.	
<code>time</code>	32u	Current time (as epoch).	
<code>version</code>	string	Version string of this instance.	
<code>pointer_size</code>	string	Size of pointers for this host specified in bits (32 or 64).	
<code>rusage_user</code>	32u:32u	Total user time for this instance (seconds:microseconds).	
<code>rusage_system</code>	32u:32u	Total system time for this instance (seconds:microseconds).	
<code>curr_items</code>	32u	Current number of items stored by this instance.	
<code>total_items</code>	32u	Total number of items stored during the life of this instance.	
<code>bytes</code>	64u	Current number of bytes used by this server to store items.	
<code>curr_connections</code>	32u	Current number of open connections.	
<code>total_connections</code>	32u	Total number of connections opened since the server started running.	
<code>connection_structures</code>	32u	Number of connection structures allocated by the server.	
<code>cmd_get</code>	64u	Total number of retrieval requests (<code>get</code> operations).	
<code>cmd_set</code>	64u	Total number of storage requests (<code>set</code> operations).	
<code>get_hits</code>	64u	Number of keys that have been requested and found present.	
<code>get_misses</code>	64u	Number of items that have been requested and not found.	
<code>delete_hits</code>	64u	Number of keys that have been deleted and found present.	1.3.x
<code>delete_misses</code>	64u	Number of items that have been delete and not found.	1.3.x
<code>incr_hits</code>	64u	Number of keys that have been incremented and found present.	1.3.x
<code>incr_misses</code>	64u	Number of items that have been incremented and not found.	1.3.x
<code>decr_hits</code>	64u	Number of keys that have been decremented and found present.	1.3.x
<code>decr_misses</code>	64u	Number of items that have been decremented and not found.	1.3.x
<code>cas_hits</code>	64u	Number of keys that have been compared and swapped and found present.	1.3.x
<code>cas_misses</code>	64u	Number of items that have been compared and swapped and not found.	1.3.x
<code>cas_badvalue</code>	64u	Number of keys that have been compared and swapped, but the comparison (original) value did not match the supplied value.	1.3.x

Statistic	Data type	Description	Version
<code>evictions</code>	64u	Number of valid items removed from cache to free memory for new items.	
<code>bytes_read</code>	64u	Total number of bytes read by this server from network.	
<code>bytes_written</code>	64u	Total number of bytes sent by this server to network.	
<code>limit_maxbytes</code>	32u	Number of bytes this server is permitted to use for storage.	
<code>threads</code>	32u	Number of worker threads requested.	
<code>conn_yields</code>	64u	Number of yields for connections (related to the <code>-R</code> option).	1.4.0

The most useful statistics from those given here are the number of cache hits, misses, and evictions.

A large number of `get_misses` may just be an indication that the cache is still being populated with information. The number should, over time, decrease in comparison to the number of cache `get_hits`. If, however, you have a large number of cache misses compared to cache hits after an extended period of execution, it may be an indication that the size of the cache is too small and you either need to increase the total memory size, or increase the number of the `memcached` instances to improve the hit ratio.

A large number of `evictions` from the cache, particularly in comparison to the number of items stored is a sign that your cache is too small to hold the amount of information that you regularly want to keep cached. Instead of items being retained in the cache, items are being evicted to make way for new items keeping the turnover of items in the cache high, reducing the efficiency of the cache.

15.6.4.2. memcached Slabs Statistics

To get the `slabs` statistics, use the `stats slabs` command, or the API equivalent.

The slab statistics provide you with information about the slabs that have created and allocated for storing information within the cache. You get information both on each individual slab-class and total statistics for the whole slab.

```
STAT 1:chunk_size 104
STAT 1:chunks_per_page 10082
STAT 1:total_pages 1
STAT 1:total_chunks 10082
STAT 1:used_chunks 10081
STAT 1:free_chunks 1
STAT 1:free_chunks_end 10079
STAT 9:chunk_size 696
STAT 9:chunks_per_page 1506
STAT 9:total_pages 63
STAT 9:total_chunks 94878
STAT 9:used_chunks 94878
STAT 9:free_chunks 0
STAT 9:free_chunks_end 0
STAT active_slabs 2
STAT total_malloced 67083616
END
```

Individual stats for each slab class are prefixed with the slab ID. A unique ID is given to each allocated slab from the smallest size up to the largest. The prefix number indicates the slab class number in relation to the calculated chunk from the specified growth factor. Hence in the example, 1 is the first chunk size and 9 is the 9th chunk allocated size.

The different parameters returned for each chunk size and the totals are shown in the following table.

Statistic	Description	Version
<code>chunk_size</code>	Space allocated to each chunk within this slab class.	
<code>chunks_per_page</code>	Number of chunks within a single page for this slab class.	
<code>total_pages</code>	Number of pages allocated to this slab class.	
<code>total_chunks</code>	Number of chunks allocated to the slab class.	
<code>used_chunks</code>	Number of chunks allocated to an item..	
<code>free_chunks</code>	Number of chunks not yet allocated to items.	
<code>free_chunks_end</code>	Number of free chunks at the end of the last allocated page.	
<code>get_hits</code>	Number of get hits to this chunk	1.3.x
<code>cmd_set</code>	Number of set commands on this chunk	1.3.x
<code>delete_hits</code>	Number of delete hits to this chunk	1.3.x
<code>incr_hits</code>	Number of increment hits to this chunk	1.3.x
<code>decr_hits</code>	Number of decrement hits to this chunk	1.3.x
<code>cas_hits</code>	Number of CAS hits to this chunk	1.3.x
<code>cas_badval</code>	Number of CAS hits on this chunk where the existing value did not match	1.3.x
<code>mem_requested</code>	The true amount of memory of memory requested within this chunk	1.4.1

The following additional statistics cover the information for the entire server, rather than on a chunk by chunk basis:

Statistic	Description	Version
<code>active_slabs</code>	Total number of slab classes allocated.	
<code>total_malloced</code>	Total amount of memory allocated to slab pages.	

The key values in the slab statistics are the `chunk_size`, and the corresponding `total_chunks` and `used_chunks` parameters. These given an indication of the size usage of the chunks within the system. Remember that one key/value pair is placed into a chunk of a suitable size.

From these stats, you can get an idea of your size and chunk allocation and distribution. If you store many items with a number of largely different sizes, consider adjusting the chunk size growth factor to increase in larger steps to prevent chunk and memory wastage. A good indication of a bad growth factor is a high number of different slab classes, but with relatively few chunks actually in use within each slab. Increasing the growth factor creates fewer slab classes and therefore makes better use of the allocated pages.

15.6.4.3. `memcached` Item Statistics

To get the `items` statistics, use the `stats items` command, or the API equivalent.

The `items` statistics give information about the individual items allocated within a given slab class.

```
STAT items:2:number 1
STAT items:2:age 452
STAT items:2:evicted 0
STAT items:2:evicted_nonzero 0
STAT items:2:evicted_time 2
STAT items:2:outofmemory 0
STAT items:2:tailrepairs 0
...
STAT items:27:number 1
STAT items:27:age 452
```

```

STAT items:27:evicted 0
STAT items:27:evicted_nonzero 0
STAT items:27:evicted_time 2
STAT items:27:outofmemory 0
STAT items:27:tailrepairs 0

```

The prefix number against each statistics relates to the corresponding chunk size, as returned by the `stats slabs` statistics. The result is a display of the number of items stored within each chunk within each slab size, and specific statistics about their age, eviction counts, and out of memory counts. A summary of the statistics is given in the following table.

Statistic	Description	
<code>number</code>	The number of items currently stored in this slab class.	
<code>age</code>	The age of the oldest item within the slab class, in seconds.	
<code>evicted</code>	The number of items evicted to make way for new entries.	
<code>evicted_time</code>	The time of the last evicted entry	
<code>evicted_nonzero</code>	The time of the last evicted non-zero entry	1.4.0
<code>outofmemory</code>	The number of items for this slab class that have triggered an out of memory error (only value when the <code>-M</code> command line option is in effect).	
<code>tailrepairs</code>	Number of times the entries for a particular ID need repairing	

Item level statistics can be used to determine how many items are stored within a given slab and their freshness and recycle rate. You can use this to help identify whether there are certain slab classes that are triggering a much larger number of evictions than others.

15.6.4.4. memcached Size Statistics

To get size statistics, use the `stats sizes` command, or the API equivalent.

The size statistics provide information about the sizes and number of items of each size within the cache. The information is returned as two columns, the first column is the size of the item (rounded up to the nearest 32 byte boundary), and the second column is the count of the number of items of that size within the cache:

```

96 35
128 38
160 807
192 804
224 410
256 222
288 83
320 39
352 53
384 33
416 64
448 51
480 30
512 54
544 39
576 10065

```



Caution

Running this statistic locks up your cache as each item is read from the cache and its size calculated. On a large cache, this may take some time and prevent any set or get operations until the process completes.

The item size statistics are useful only to determine the sizes of the objects you are storing. Since the actual memory allocation is relevant only in terms of the chunk size and page size, the information is only useful during a careful debugging or diagnostic session.

15.6.4.5. `memcached` Detail Statistics

For `memcached` 1.3.x and higher, you can enable and obtain detailed statistics about the `get`, `set`, and `del` operations on the individual keys stored in the cache, and determine whether the attempts hit (found) a particular key. These operations are only recorded while the detailed stats analysis is turned on.

To enable detailed statistics, you must send the `stats detail on` command to the `memcached` server:

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to tiger.
Escape character is '^]'.stats detail on
OK
```

Individual statistics are recorded for every `get`, `set` and `del` operation on a key, including keys that are not currently stored in the server. For example, if an attempt is made to obtain the value of key `abckey` and it does not exist, the `get` operating on the specified key are recorded while detailed statistics are in effect, even if the key is not currently stored. The `hits`, that is, the number of `get` or `del` operations for a key that exists in the server are also counted.

To turn detailed statistics off, send the `stats detail off` command to the `memcached` server:

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to tiger.
Escape character is '^]'.stats detail on
OK
```

To obtain the detailed statistics recorded during the process, send the `stats detail dump` command to the `memcached` server:

```
stats detail dump
PREFIX hykkey get 0 hit 0 set 1 del 0
PREFIX xyzkey get 0 hit 0 set 1 del 0
PREFIX yukkey get 1 hit 0 set 0 del 0
PREFIX abckey get 3 hit 3 set 1 del 0
END
```

You can use the detailed statistics information to determine whether your `memcached` clients are using a large number of keys that do not exist in the server by comparing the `hit` and `get` or `del` counts. Because the information is recorded by key, you can also determine whether the failures or operations are clustered around specific keys.

15.6.4.6. Using `memcached-tool`

The `memcached-tool`, located within the `scripts` directory within the `memcached` source directory. The tool provides convenient access to some reports and statistics from any `memcached` instance.

The basic format of the command is:

```
shell> ./memcached-tool hostname:port [command]
```

The default output produces a list of the slab allocations and usage. For example:

```
shell> memcached-tool localhost:11211 display
# Item_Size Max_age Pages Count Full? Evicted Evict_Time OOM
```

Getting memcached Statistics

1	80B	93s	1	20	no	0	0	0
2	104B	93s	1	16	no	0	0	0
3	136B	1335s	1	28	no	0	0	0
4	176B	1335s	1	24	no	0	0	0
5	224B	1335s	1	32	no	0	0	0
6	280B	1335s	1	34	no	0	0	0
7	352B	1335s	1	36	no	0	0	0
8	440B	1335s	1	46	no	0	0	0
9	552B	1335s	1	58	no	0	0	0
10	696B	1335s	1	66	no	0	0	0
11	872B	1335s	1	89	no	0	0	0
12	1.1K	1335s	1	112	no	0	0	0
13	1.3K	1335s	1	145	no	0	0	0
14	1.7K	1335s	1	123	no	0	0	0
15	2.1K	1335s	1	198	no	0	0	0
16	2.6K	1335s	1	199	no	0	0	0
17	3.3K	1335s	1	229	no	0	0	0
18	4.1K	1335s	1	248	yes	36	2	0
19	5.2K	1335s	2	328	no	0	0	0
20	6.4K	1335s	2	316	yes	387	1	0
21	8.1K	1335s	3	381	yes	492	1	0
22	10.1K	1335s	3	303	yes	598	2	0
23	12.6K	1335s	5	405	yes	605	1	0
24	15.8K	1335s	6	384	yes	766	2	0
25	19.7K	1335s	7	357	yes	908	170	0
26	24.6K	1336s	7	287	yes	1012	1	0
27	30.8K	1336s	7	231	yes	1193	169	0
28	38.5K	1336s	4	104	yes	1323	169	0
29	48.1K	1336s	1	21	yes	1287	1	0
30	60.2K	1336s	1	17	yes	1093	169	0
31	75.2K	1337s	1	13	yes	713	168	0
32	94.0K	1337s	1	10	yes	278	168	0
33	117.5K	1336s	1	3	no	0	0	0

This output is the same if you specify the `command` as `display`:

```
shell> memcached-tool localhost:11211 display
# Item_Size Max_age Pages Count Full? Evicted Evict_Time OOM
1 80B 93s 1 20 no 0 0 0
2 104B 93s 1 16 no 0 0 0
...
```

The output shows a summarized version of the output from the `slabs` statistics. The columns provided in the output are shown below:

- `#`: The slab number
- `Item_Size`: The size of the slab
- `Max_age`: The age of the oldest item in the slab
- `Pages`: The number of pages allocated to the slab
- `Count`: The number of items in this slab
- `Full?`: Whether the slab is fully populated
- `Evicted`: The number of objects evicted from this slab
- `Evict_Time`: The time (in seconds) since the last eviction
- `OOM`: The number of items that have triggered an out of memory error

You can also obtain a dump of the general statistics for the server using the `stats` command:

```
shell> memcached-tool localhost:11211 stats
#localhost:11211  Field      Value
  accepting_conns      1
    bytes              162
  bytes_read           485
  bytes_written        6820
  cas_badval           0
  cas_hits              0
  cas_misses           0
  cmd_flush            0
  cmd_get              4
  cmd_set              2
  conn_yields          0
connection_structures 11
  curr_connections     10
    curr_items         2
  decr_hits            0
  decr_misses         1
  delete_hits          0
  delete_misses       0
  evictions            0
  get_hits             4
  get_misses           0
  incr_hits            0
  incr_misses         2
  limit_maxbytes       67108864
listen_disabled_num   0
  pid                 12981
  pointer_size         32
  rusage_system        0.013911
  rusage_user          0.011876
  threads              4
  time                1255518565
total_connections     20
  total_items          2
  uptime              880
  version              1.4.2
```

15.6.5. memcached FAQ

Questions

- [16.6.5.1: \[1797\]](#) Can MySQL actually trigger/store the changed data to memcached?
- [16.6.5.2: \[1797\]](#) Can memcached be run on a Windows environment?
- [16.6.5.3: \[1797\]](#) What is the maximum size of an object you can store in memcached? Is that configurable?
- [16.6.5.4: \[1797\]](#) Is it true `memcached` will be much more effective with db-read-intensive applications than with db-write-intensive applications?
- [16.6.5.5: \[1797\]](#) Is there any overhead in not using persistent connections? If persistent is always recommended, what are the downsides (for example, locking up)?
- [16.6.5.6: \[1798\]](#) How is an event such as a crash of one of the `memcached` servers handled by the `memcached` client?
- [16.6.5.7: \[1798\]](#) What is a recommended hardware configuration for a memcached server?
- [16.6.5.8: \[1798\]](#) Is memcached more effective for video and audio as opposed to textual read/writes?
- [16.6.5.9: \[1798\]](#) Can `memcached` work with ASPX?

- [16.6.5.10: \[1798\]](#) How expensive is it to establish a memcache connection? Should those connections be pooled?
- [16.6.5.11: \[1798\]](#) How is the data handled when the `memcached` server is down?
- [16.6.5.12: \[1799\]](#) How are auto-increment columns in the MySQL database coordinated across multiple instances of memcached?
- [16.6.5.13: \[1799\]](#) Is compression available?
- [16.6.5.14: \[1799\]](#) Can we implement different types of `memcached` as different nodes in the same server, so can there be deterministic and non-deterministic in the same server?
- [16.6.5.15: \[1799\]](#) What are best practices for testing an implementation, to ensure that it improves performance, and to measure the impact of `memcached` configuration changes? And would you recommend keeping the configuration very simple to start?

Questions and Answers

16.6.5.1: Can MySQL actually trigger/store the changed data to memcached?

Yes. You can use the MySQL UDFs for `memcached` and either write statements that directly set the values in the `memcached` server, or use triggers or stored procedures to do it for you. For more information, see [Section 15.6.3.9, “Using the MySQL `memcached` User-Defined Functions”](#)

16.6.5.2: Can memcached be run on a Windows environment?

No. Currently `memcached` is available only on the Unix/Linux platform. There is an unofficial port available, see <http://www.codeplex.com/memcachedproviders>.

16.6.5.3: What is the maximum size of an object you can store in memcached? Is that configurable?

The default maximum object size is 1MB. In `memcached` 1.4.2 and later, you can change the maximum size of an object using the `-I` command line option.

For versions before this, to increase this size, you have to re-compile `memcached`. You can modify the value of the `POWER_BLOCK` within the `slabs.c` file within the source.

In `memcached` 1.4.2 and higher, you can configure the maximum supported object size by using the `-I` command-line option. For example, to increase the maximum object size to 5MB:

```
$ memcached -I 5m
```

If an object is larger than the maximum object size, you must manually split it. `memcached` is very simple: you give it a key and some data, it tries to cache it in RAM. If you try to store more than the default maximum size, the value is just truncated for speed reasons.

16.6.5.4: Is it true `memcached` will be much more effective with db-read-intensive applications than with db-write-intensive applications?

Yes. `memcached` plays no role in database writes, it is a method of caching data already read from the database in RAM.

16.6.5.5: Is there any overhead in not using persistent connections? If persistent is always recommended, what are the downsides (for example, locking up)?

If you don't use persistent connections when communicating with `memcached`, there will be a small increase in the latency of opening the connection each time. The effect is comparable to use nonpersistent connections with MySQL.

In general, the chance of locking or other issues with persistent connections is minimal, because there is very little locking within `memcached`. If there is a problem, eventually your request will time out and return no result, so your application will need to load from MySQL again.

16.6.5.6: How is an event such as a crash of one of the `memcached` servers handled by the `memcached` client?

There is no automatic handling of this. If your client fails to get a response from a server, code a fallback mechanism to load the data from the MySQL database.

The client APIs all provide the ability to add and remove `memcached` instances on the fly. If within your application you notice that `memcached` server is no longer responding, you can remove the server from the list of servers, and keys will automatically be redistributed to another `memcached` server in the list. If retaining the cache content on all your servers is important, make sure you use an API that supports a consistent hashing algorithm. For more information, see [Section 15.6.2.4, “memcached Hashing/Distribution Types”](#).

16.6.5.7: What is a recommended hardware configuration for a `memcached` server?

`memcached` has a very low processing overhead. All that is required is spare physical RAM capacity. A `memcached` server does not require a dedicated machine. If you have web, application, or database servers that have spare RAM capacity, then use them with `memcached`.

To build and deploy a dedicated `memcached` server, use a relatively low-power CPU, lots of RAM, and one or more Gigabit Ethernet interfaces.

16.6.5.8: Is `memcached` more effective for video and audio as opposed to textual read/writes?

`memcached` works equally well for all kinds of data. To `memcached`, any value you store is just a stream of data. Remember, though, that the maximum size of an object you can store in `memcached` is 1MB, but can be configured to be larger by using the `-I` option in `memcached` 1.4.2 and later, or by modifying the source in versions before 1.4.2. If you plan on using `memcached` with audio and video content, you will probably want to increase the maximum object size. Also remember that `memcached` is a solution for caching information for reading. It shouldn't be used for writes, except when updating the information in the cache.

16.6.5.9: Can `memcached` work with ASPX?

There are ports and interfaces for many languages and environments. ASPX relies on an underlying language such as C# or VisualBasic, and if you are using ASP.NET then there is a C# `memcached` library. For more information, see <https://sourceforge.net/projects/memcacheddotnet/>.

16.6.5.10: How expensive is it to establish a memcache connection? Should those connections be pooled?

Opening the connection is relatively inexpensive, because there is no security, authentication or other handshake taking place before you can start sending requests and getting results. Most APIs support a persistent connection to a `memcached` instance to reduce the latency. Connection pooling would depend on the API you are using, but if you are communicating directly over TCP/IP, then connection pooling would provide some small performance benefit.

16.6.5.11: How is the data handled when the `memcached` server is down?

The behavior is entirely application dependent. Most applications fall back to loading the data from the database (just as if they were updating the `memcached` information). If you are using multiple `memcached` servers, you might also remove a downed server from the list to prevent it from affecting performance. Otherwise, the client will still attempt to communicate with the `memcached` server that corresponds to the key you are trying to load.

16.6.5.12: How are auto-increment columns in the MySQL database coordinated across multiple instances of memcached?

They aren't. There is no relationship between MySQL and `memcached` unless your application (or, if you are using the MySQL UDFs for `memcached`, your database definition) creates one.

If you are storing information based on an auto-increment key into multiple instances of `memcached`, the information is only stored on one of the `memcached` instances anyway. The client uses the key value to determine which `memcached` instance to store the information. It doesn't store the same information across all the instances, as that would be a waste of cache memory.

16.6.5.13: Is compression available?

Yes. Most of the client APIs support some sort of compression, and some even allow you to specify the threshold at which a value is deemed appropriate for compression during storage.

16.6.5.14: Can we implement different types of memcached as different nodes in the same server, so can there be deterministic and non-deterministic in the same server?

Yes. You can run multiple instances of `memcached` on a single server, and in your client configuration you choose the list of servers you want to use.

16.6.5.15: What are best practices for testing an implementation, to ensure that it improves performance, and to measure the impact of memcached configuration changes? And would you recommend keeping the configuration very simple to start?

The best way to test the performance is to start up a `memcached` instance. First, modify your application so that it stores the data just before the data is about to be used or displayed into `memcached`. Since the APIs handle the serialization of the data, it should just be a one-line modification to your code. Then, modify the start of the process that would normally load that information from MySQL with the code that requests the data from `memcached`. If the data cannot be loaded from `memcached`, default to the MySQL process.

All of the changes required will probably amount to just a few lines of code. To get the best benefit, make sure you cache entire objects (for example, all the components of a web page, blog post, discussion thread, and so on), rather than using `memcached` as a simple cache of individual rows of MySQL tables.

Keeping the configuration simple at the start, or even over the long term, is easy with `memcached`. Once you have the basic structure up and running, often the only ongoing change is to add more servers into the list of servers used by your applications. You don't need to manage the `memcached` servers, and there is no complex configuration; just add more servers to the list and let the client API and the `memcached` servers make the decisions.

15.7. MySQL Proxy

The MySQL Proxy is an application that communicates over the network using the MySQL network protocol and provides communication between one or more MySQL servers and one or more MySQL clients. Because MySQL Proxy uses the MySQL network protocol, it can be used without modification with any MySQL-compatible client that uses the protocol. This includes the `mysql` command-line client, any clients that uses the MySQL client libraries, and any connector that supports the MySQL network protocol.

In the most basic configuration, MySQL Proxy simply interposes itself between the server and clients, passing queries from the clients to the MySQL Server and returning the responses from the MySQL Server to the appropriate client. In more advanced configurations, the MySQL Proxy can also monitor and alter the communication between the client and the server. Query interception enables you to add profiling, and interception of the exchanges is scriptable using the Lua scripting language.

By intercepting the queries from the client, the proxy can insert additional queries into the list of queries sent to the server, and remove the additional results when they are returned by the server. Using this functionality you can return the results from the original query to the client while adding informational statements to each query, for example, to monitor their execution time or progress, and separately log the results.

The proxy enables you to perform additional monitoring, filtering, or manipulation of queries without requiring you to make any modifications to the client and without the client even being aware that it is communicating with anything but a genuine MySQL server.

This documentation covers MySQL Proxy 0.8.2. And MySQL Proxy contains third-party code. For license information on third-party code, see [Appendix A, Licenses for Third-Party Components](#).



Warning

MySQL Proxy is currently an Alpha release and should not be used within production environments.



Important

MySQL Proxy is compatible with MySQL 5.0 or later. Testing has not been performed with Version 4.1. Please provide feedback on your experiences using the [MySQL Proxy Forum](#).

For release notes detailing the changes in each release of MySQL Proxy, see [MySQL Proxy Release Notes](#).

15.7.1. MySQL Proxy Supported Platforms

MySQL Proxy is currently available as a precompiled binary for the following platforms:

- Linux (including Red Hat, Fedora, Debian, SuSE) and derivatives
- Mac OS X
- FreeBSD
- IBM AIX
- Sun Solaris
- Microsoft Windows (including Microsoft Windows XP, Microsoft Windows Vista, Microsoft Windows Server 2003, Microsoft Windows Server 2008)



Note

You must have the .NET Framework 1.1 or higher installed.

Other Unix/Linux platforms not listed should be compatible by using the source package and building MySQL Proxy locally.

System requirements for the MySQL Proxy application are the same as the main MySQL server. Currently MySQL Proxy is compatible only with MySQL 5.0.1 and later. MySQL Proxy is provided as a standalone, statically linked binary. You need not have MySQL or Lua installed.

15.7.2. Installing MySQL Proxy

You have three choices for installing MySQL Proxy:

- Precompiled binaries are available for a number of different platforms. See [Section 15.7.2.1, “Installing MySQL Proxy from a Binary Distribution”](#).
- You can install from the source code to build on an environment not supported by the binary distributions. See [Section 15.7.2.2, “Installing MySQL Proxy from a Source Distribution”](#).
- The latest version of the MySQL Proxy source code is available through a development repository is the best way to stay up to date with the latest fixes and revisions. See [Section 15.7.2.3, “Installing MySQL Proxy from the Bazaar Repository”](#).

15.7.2.1. Installing MySQL Proxy from a Binary Distribution

If you download a binary package, you must extract and copy the package contents to your desired installation directory. The package contains files required by MySQL Proxy, including additional Lua scripts and other components required for execution.

To install, unpack the archive into the desired directory, then modify your `PATH` environment variable so that you can use the `mysql-proxy` command directly:

```
shell> cd /usr/local
shell> tar xzf mysql-proxy-0.8.2-platform.tar.gz
shell> PATH=$PATH:/usr/local/mysql-proxy-0.8.2-platform/sbin
```

To update the path globally on a system, you might need administrator privileges to modify the appropriate `/etc/profile`, `/etc/bashrc`, or other system configuration file.

On Windows, you can update the `PATH` environment variable using this procedure:

1. On the Windows desktop, right-click the My Computer icon, and select Properties.
2. Next select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.
3. Under **System Variables**, select Path, then click the Edit button. The Edit System Variable dialogue should appear.

The Microsoft Visual C++ runtime libraries are a requirement for running MySQL Proxy as of version 0.8.2. Users that do not have these libraries must download and install the Microsoft Visual C++ 2008 Service Pack 1 Redistributable Package MFC Security Update. Use the following link to obtain the package:

```
http://www.microsoft.com/download/en/details.aspx?id=26368
```

15.7.2.2. Installing MySQL Proxy from a Source Distribution

You can download a source package and compile the MySQL Proxy yourself. To build from source, you must have the following prerequisite components installed:

- `libevent` 1.x or higher (1.3b or later is preferred).
- `lua` 5.1.x or higher.
- `glib2` 2.6.0 or higher.
- `pkg-config`.
- `libtool` 1.5 or higher.
- MySQL 5.0.x or higher developer files.

**Note**

On some operating systems, you might need to manually build the required components to get the latest version. If you have trouble compiling MySQL Proxy, consider using a binary distributions instead.

After verifying that the prerequisite components are installed, configure and build MySQL Proxy:

```
shell> tar zxf mysql-proxy-0.8.2.tar.gz
shell> cd mysql-proxy-0.8.2
shell> ./configure
shell> make
```

To test the build, use the `check` target to `make`:

```
shell> make check
```

The tests try to connect to `localhost` using the `root` user. To provide a password, set the `MYSQL_PASSWORD` environment variable:

```
shell> MYSQL_PASSWORD=root_pwd make check
```

You can install using the `install` target:

```
shell> make install
```

By default, `mysql-proxy` is installed into `/usr/local/sbin/mysql-proxy`. The Lua example scripts are installed into `/usr/local/share`.

15.7.2.3. Installing MySQL Proxy from the Bazaar Repository

The MySQL Proxy source is available through a public Bazaar repository and is the quickest way to get the latest releases and fixes.

A build from the Bazaar repository requires that the following prerequisite components be installed:

- Bazaar 1.10.0 or later.
- `libtool` 1.5 or higher.
- `autoconf` 2.56 or higher.
- `automake` 1.10 or higher.
- `libevent` 1.x or higher (1.3b or later is preferred).
- `lua` 5.1.x or higher.
- `glib2` 2.4.0 or higher.
- `pkg-config`.
- MySQL 5.0.x or higher developer files.

The `mysql-proxy` source is hosted on Launchpad. To check out a local copy of the Bazaar repository, use `bzr`:

```
shell> bzr branch lp:mysql-proxy
```

The preceding command downloads a complete version of the Bazaar repository for `mysql-proxy`. The main source files are located within the `trunk` subdirectory. The configuration scripts must be generated before you can configure and build `mysql-proxy`. The `autogen.sh` script generates the required configuration scripts for you:

```
shell> sh ./autogen.sh
```

The `autogen.sh` script creates the standard `configure` script, which you then use to configure and build with `make`:

```
shell> ./configure
shell> make
shell> make install
```

To create a standalone source distribution, identical to the source distribution available for download, use this command:

```
shell> make distcheck
```

The preceding command creates the file `mysql-proxy-0.8.2.tar.gz` (with the corresponding current version) within the current directory.

15.7.2.4. Setting Up MySQL Proxy as a Windows Service

The MySQL distribution on Windows includes the `mysql-proxy-svc.exe` command that enables a MySQL Proxy instance to be managed by the Windows service control manager. You can control the service, including automatically starting and stopping it during boot, reboot and shutdown, without separately running the MySQL Proxy application.

To set up a MySQL Proxy service, use the `sc` command to create a new service using the MySQL Proxy service command. Specify the MySQL Proxy options on the `sc` command line, and identify the service with a unique name. For example, to configure a new MySQL Proxy instance that will automatically start when your system boots, redirecting queries to the local MySQL server:

```
C:\> sc create "Proxy" DisplayName= "MySQL Proxy" start= "auto" >
binPath= "C:\Program Files\MySQL\mysql-proxy-0.8.2\bin\mysql-proxy-svc.exe >
--proxy-backend-addresses=127.0.0.1:3306"
```



Note

The space following the equal sign after each property is required; failure to include it results in an error.

The preceding command creates a new service called `Proxy`. You can start and stop the service using the `net start|stop` command with the service name. The service is not automatically started after it is created. To start the service:

```
C:\> net start proxy
The MySQL Proxy service is starting.
The MySQL Proxy service was started successfully.
```

You can specify additional command-line options to the `sc` command. You can also set up multiple MySQL Proxy services on the same machine (providing they are configured to listen on different ports and/or IP addresses).

You can delete a service that you have created:

```
C:\> sc delete proxy
```

For more information on creating services using `sc`, see [How to create a Windows service by using Sc.exe](#).

15.7.3. MySQL Proxy Command Options

To start MySQL Proxy, you can run it directly from the command line:

```
shell> mysql-proxy
```

For most situations, you specify at least the host name or address and the port number of the backend MySQL server to which the MySQL Proxy should pass queries.

You can specify options to `mysql-proxy` either on the command line, or by using a configuration file and the `--defaults-file [1808]` command-line option to specify the file location.

If you use a configuration file, format it as follows:

- Specify the options within a `[mysql-proxy]` configuration group. For example:

```
[mysql-proxy]
admin-address = host:port
```

- Specify all configuration options in the form of a configuration name and the value to set.
- For options that are a simple toggle on the command line (for example, `--proxy-skip-profiling [1812]`), use `true` or `false`. For example, the following is invalid:

```
[mysql-proxy]
proxy-skip-profiling
```

But this is valid:

```
[mysql-proxy]
proxy-skip-profiling = true
```

- Give the configuration file Unix permissions of `0660` (readable and writable by user and group, no access for others).

Failure to adhere to any of these requirements causes `mysql-proxy` to generate an error during startup.

The following tables list the supported configuration file and command-line options.

Table 15.4. `mysql-proxy` Help Options

Format	Option File	Description
<code>--help [1806]</code>		Show help options
<code>--help-admin [1806]</code>		Show admin module options
<code>--help-all [1806]</code>		Show all help options
<code>--help-proxy [1806]</code>		Show proxy module options

Table 15.5. `mysql-proxy` Admin Options

Format	Option File	Description
<code>--admin-address=host:port [1806]</code>	<code>admin-address=host:port [1806]</code>	The admin module listening host and port
<code>--admin-lua-script=file_name [1806]</code>	<code>admin-lua-script=file_name [1806]</code>	Script to execute by the admin module
<code>--admin-password=password [1807]</code>	<code>admin-password=password [1807]</code>	Authentication password for admin module
<code>--admin-username=user_name [1807]</code>	<code>admin-username=user_name [1807]</code>	Authentication user name for admin module
<code>--proxy-address=host:port [1810]</code>	<code>proxy-address=host:port [1810]</code>	The listening proxy server host and port

Table 15.6. `mysql-proxy` Proxy Options

Format	Option File	Description	Removed
<code>--no-proxy</code> [1809]	<code>no-proxy</code> [1809]	Do not start the proxy module	
<code>--proxy-backend-addresses=host:port</code> [1811]	<code>proxy-backend-addresses=host:port</code> [1811]	The MySQL server host and port	
<code>--proxy-fix-bug-25371</code> [1812]	<code>proxy-fix-bug-25371</code> [1812]	Enable the fix for Bug #25371 for older libmysql versions	0.8.1
<code>--proxy-lua-script=file_name</code> [1812]	<code>proxy-lua-script=file_name</code> [1812]	Filename for Lua script for proxy operations	
<code>--proxy-pool-no-change-user</code> [1811]	<code>proxy-pool-no-change-user</code> [1811]	Do not use the protocol <code>CHANGE_USER</code> command to reset the connection when coming from the connection pool	
<code>--proxy-read-only-backend-addresses=host:port</code> [1810]	<code>proxy-read-only-backend-addresses=host:port</code> [1810]	The MySQL server host and port (read only)	
<code>--proxy-skip-profiling</code> [1812]	<code>proxy-skip-profiling</code> [1812]	Disable query profiling	

Table 15.7. `mysql-proxy` Applications Options

Format	Option File	Description
<code>--basedir=dir_name</code> [1807]	<code>basedir=dir_name</code> [1807]	The base directory prefix for paths in the configuration
<code>--daemon</code> [1807]	<code>daemon</code> [1807]	Start in daemon mode
<code>--defaults-file=file_name</code> [1808]		The configuration file to use
<code>--event-threads=count</code> [1808]	<code>event-threads=count</code> [1808]	The number of event-handling threads
<code>--keepalive</code> [1808]	<code>keepalive</code> [1808]	Try to restart the proxy if a crash occurs
<code>--log-backtrace-on-crash</code> [1808]	<code>log-backtrace-on-crash</code> [1808]	Try to invoke the debugger and generate a backtrace on crash
<code>--log-file=file_name</code> [1808]	<code>log-file=file_name</code> [1808]	The file where error messages are logged
<code>--log-level=level</code> [1808]	<code>log-level=level</code> [1808]	The logging level
<code>--log-use-syslog</code> [1809]	<code>log-use-syslog</code> [1809]	Log errors to syslog
<code>--lua-cpath=dir_name</code> [1809]	<code>lua-cpath=dir_name</code> [1809]	Set the <code>LUA_CPATH</code>
<code>--lua-path=dir_name</code> [1809]	<code>lua-path=dir_name</code> [1809]	Set the <code>LUA_PATH</code>
<code>--max-open-files=count</code> [1809]	<code>max-open-files=count</code> [1809]	The maximum number of open files to support
<code>--pid-file=file_name</code> [1812]	<code>pid-file=file_name</code> [1812]	File in which to store the process ID

MySQL Proxy Command Options

Format	Option File	Description
<code>--plugin-dir=dir_name [1810]</code>	<code>plugin-dir=dir_name [1810]</code>	Directory containing plugin files
<code>--plugins=plugin,... [1810]</code>	<code>plugins=plugin,... [1810]</code>	List of plugins to load
<code>--user=user_name [1812]</code>	<code>user=user_name [1812]</code>	The user to use when running mysql-proxy
<code>--version [1813]</code>		Show version information

Except as noted in the following details, all of the options can be used within the configuration file by supplying the option and the corresponding value. For example:

```
[mysql-proxy]
log-file = /var/log/mysql-proxy.log
log-level = message
```

- `--help [1806], -h`

Command-Line Format	<code>--help</code>
	<code>-h</code>

Show available help options.

- `--help-admin [1806]`

Command-Line Format	<code>--help-admin</code>
----------------------------	---------------------------

Show options for the admin module.

- `--help-all [1806]`

Command-Line Format	<code>--help-all</code>
----------------------------	-------------------------

Show all help options.

- `--help-proxy [1806]`

Command-Line Format	<code>--help-proxy</code>
----------------------------	---------------------------

Show options for the proxy module.

- `--admin-address=host:port [1806]`

Command-Line Format	<code>--admin-address=host:port</code>
Option-File Format	<code>admin-address=host:port</code>
	Permitted Values
	Type <code>string</code>
	Default <code>:4041</code>

The host name (or IP address) and port for the administration port. The default is `localhost:4041`.

- `--admin-lua-script=file_name [1806]`

Command-Line Format	<code>--admin-lua-script=file_name</code>	
Option-File Format	<code>admin-lua-script=file_name</code>	
	Permitted Values	
	Type	<code>file name</code>
	Default	

The script to use for the proxy administration module.

- `--admin-password=password` [1807]

Command-Line Format	<code>--admin-password=password</code>	
Option-File Format	<code>admin-password=password</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

The password to use to authenticate users wanting to connect to the MySQL Proxy administration module. This module uses the MySQL protocol to request a user name and password for connections.

- `--admin-username=user_name` [1807]

Command-Line Format	<code>--admin-username=user_name</code>	
Option-File Format	<code>admin-username=user_name</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>root</code>

The user name to use to authenticate users wanting to connect to the MySQL Proxy administration module. This module uses the MySQL protocol to request a user name and password for connections. The default user name is `root`.

- `--basedir=dir_name` [1807]

Command-Line Format	<code>--basedir=dir_name</code>	
Option-File Format	<code>basedir=dir_name</code>	
	Permitted Values	
	Type	<code>directory name</code>

The base directory to use as a prefix for all other file name configuration options. The base name should be an absolute (not relative) directory. If you specify a relative directory, `mysql-proxy` generates an error during startup.

- `--daemon` [1807]

Command-Line Format	<code>--daemon</code>	
Option-File Format	<code>daemon</code>	

Starts the proxy in daemon mode.

- `--defaults-file=file_name` [1808]

Command-Line Format	<code>--defaults-file=file_name</code>
----------------------------	--

The file to read for configuration options. If not specified, MySQL Proxy takes options only from the command line.

- `--event-threads=count` [1808]

Command-Line Format	<code>--event-threads=count</code>	
Option-File Format	<code>event-threads=count</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>1</code>

The number of event threads to reserve to handle incoming requests.

- `--keepalive` [1808]

Command-Line Format	<code>--keepalive</code>
Option-File Format	<code>keepalive</code>

Create a process surrounding the main `mysql-proxy` process that attempts to restart the main `mysql-proxy` process in the event of a crash or other failure.



Note

The `--keepalive` [1808] option is not available on Microsoft Windows. When running as a service, `mysql-proxy` automatically restarts.

- `--log-backtrace-on-crash` [1808]

Command-Line Format	<code>--log-backtrace-on-crash</code>
Option-File Format	<code>log-backtrace-on-crash</code>

Log a backtrace to the error log and try to initialize the debugger in the event of a failure.

- `--log-file=file_name` [1808]

Command-Line Format	<code>--log-file=file_name</code>	
Option-File Format	<code>log-file=file_name</code>	
	Permitted Values	
	Type	<code>file name</code>

The file to use to record log information. If this option is not given, `mysql-proxy` logs to the standard error output.

- `--log-level=level` [1808]

Command-Line Format	<code>--log-level=level</code>
Option-File Format	<code>log-level=level</code>

	Permitted Values	
	Type	enumeration
	Valid Values	error warning info message debug

The log level to use when outputting error messages. Messages with that level (or lower) are output. For example, `message` level also outputs message with `info`, `warning`, and `error` levels.

- `--log-use-syslog` [1809]

Command-Line Format	<code>--log-use-syslog</code>
Option-File Format	<code>log-use-syslog</code>

Log errors to the syslog (Unix/Linux only).

- `--lua-cpath=dir_name` [1809]

Command-Line Format	<code>--lua-cpath=dir_name</code>
Option-File Format	<code>lua-cpath=dir_name</code>
	Permitted Values
	Type directory name

The `LUA_CPATH` to use when loading compiled modules or libraries for Lua scripts.

- `--lua-path=dir_name` [1809]

Command-Line Format	<code>--lua-path=dir_name</code>
Option-File Format	<code>lua-path=dir_name</code>
	Permitted Values
	Type directory name

The `LUA_CPATH` to use when loading modules for Lua.

- `--max-open-files=count` [1809]

Command-Line Format	<code>--max-open-files=count</code>
Option-File Format	<code>max-open-files=count</code>
	Permitted Values
	Type numeric

The maximum number of open files and sockets supported by the `mysql-proxy` process. Certain scripts might require a higher value.

- `--no-proxy` [1809]

Command-Line Format	<code>--no-proxy</code>
Option-File Format	<code>no-proxy</code>

Disable the proxy module.

- `--plugin-dir=dir_name` [1810]

Command-Line Format	<code>--plugin-dir=dir_name</code>	
Option-File Format	<code>plugin-dir=dir_name</code>	
	Permitted Values	
	Type	directory name

The directory to use when loading plugins for `mysql-proxy`.

- `--plugins=plugin` [1810]

Command-Line Format	<code>--plugins=plugin,...</code>	
Option-File Format	<code>plugins=plugin,...</code>	
	Permitted Values	
	Type	string

Loads a plugin.

When using this option on the command line, you can specify the option multiple times to specify multiple plugins. For example:

```
shell> mysql-proxy --plugins=proxy --plugins=admin
```

When using the option within the configuration file, you should separate multiple plugins by commas. The equivalent of the preceding example would be:

```
...
plugins=proxy,admin
```

- `--proxy-address=host:port` [1810], `-P host:port`

Command-Line Format	<code>--proxy-address=host:port</code>	
	<code>-P host:port</code>	
Option-File Format	<code>proxy-address=host:port</code>	
	Permitted Values	
	Type	string
	Default	:4040

The listening host name (or IP address) and port of the proxy server. The default is `:4040` (all IPs on port 4040).

- `--proxy-read-only-backend-addresses=host:port` [1810], `-r host:port`

Command-Line Format	<code>--proxy-read-only-backend-addresses=host:port</code>	
	<code>-r host:port</code>	

Option-File Format	<code>proxy-read-only-backend-addresses=host:port</code>	
	Permitted Values	
	Type	<code>string</code>

The listening host name (or IP address) and port of the proxy server for read-only connections. The default is for this information not to be set.



Note

Setting this value only configures the servers within the corresponding internal structure (see `proxy.global.backends` [1818]). You can determine the backend type by checking the `type` field for each connection.

You should therefore only use this option in combination with a script designed to make use of the different backend types.

When using this option on the command line, you can specify the option and the server multiple times to specify multiple backends. For example:

```
shell> mysql-proxy --proxy-read-only-backend-addresses=192.168.0.1:3306 --proxy-read-only-backend-addresses=192.168.0.2:3306
```

When using the option within the configuration file, you should separate multiple servers by commas. The equivalent of the preceding example would be:

```
...
proxy-read-only-backend-addresses = 192.168.0.1:3306,192.168.0.2:3306
```

- `--proxy-backend-addresses=host:port` [1811], `-b host:port`

Command-Line Format	<code>--proxy-backend-addresses=host:port</code>	
	<code>-b host:port</code>	
Option-File Format	<code>proxy-backend-addresses=host:port</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>127.0.0.1:3306</code>

The host name (or IP address) and port of the MySQL server to connect to. You can specify multiple backend servers by supplying multiple options. Clients are connected to each backend server in round-robin fashion. For example, if you specify two servers A and B, the first client connection will go to server A; the second client connection to server B and the third client connection to server A.

When using this option on the command line, you can specify the option and the server multiple times to specify multiple backends. For example:

```
shell> mysql-proxy --proxy-backend-addresses 192.168.0.1:3306 --proxy-backend-addresses 192.168.0.2:3306
```

When using the option within the configuration file, you should separate multiple servers by commas. The equivalent of the preceding example would be:

```
...
proxy-backend-addresses = 192.168.0.1:3306,192.168.0.2:3306
```

- `--proxy-pool-no-change-user` [1811]

MySQL Proxy Command Options

Command-Line Format	<code>--proxy-pool-no-change-user</code>
Option-File Format	<code>proxy-pool-no-change-user</code>

Disable use of the MySQL protocol `CHANGE_USER` command when reusing a connection from the pool of connections specified by the `proxy-backend-addresses` list.

- `--proxy-skip-profiling` [1812]

Command-Line Format	<code>--proxy-skip-profiling</code>
Option-File Format	<code>proxy-skip-profiling</code>

Disable query profiling (statistics time tracking). The default is for tracking to be enabled.

- `--proxy-fix-bug-25371` [1812]

Version Removed	0.8.1
Command-Line Format	<code>--proxy-fix-bug-25371</code>
Option-File Format	<code>proxy-fix-bug-25371</code>

Enable a workaround for an issue when connecting to a MySQL server later than 5.1.12 when using a MySQL client library of any earlier version.

This option was removed in `mysql-proxy` 0.8.1. Now, `mysql-proxy` returns an error message at the protocol level if it sees a `COM_CHANGE_USER` being sent to a server that has a version from 5.1.14 to 5.1.17.

- `--proxy-lua-script=file_name` [1812], `-s file_name`

Command-Line Format	<code>--proxy-lua-script=file_name</code>	
	<code>-s file_name</code>	
Option-File Format	<code>proxy-lua-script=file_name</code>	
	Permitted Values	
	Type	<code>file name</code>

The Lua script file to be loaded. Note that the script file is not physically loaded and parsed until a connection is made. Also note that the specified Lua script is reloaded for each connection; if the content of the Lua script changes while `mysql-proxy` is running, the updated content is automatically used when a new connection is made.

- `--pid-file=file_name` [1812]

Command-Line Format	<code>--pid-file=file_name</code>	
Option-File Format	<code>pid-file=file_name</code>	
	Permitted Values	
	Type	<code>file name</code>

The name of the file in which to store the process ID.

- `--user=user_name` [1812]

Command-Line Format	<code>--user=user_name</code>	
Option-File Format	<code>user=user_name</code>	
	Permitted Values	
	Type	<code>string</code>

Run `mysql-proxy` as the specified `user`.

- `--version [1813], -V`

Command-Line Format	<code>--version</code>	
	<code>-V</code>	

Show the version number.

The most common usage is as a simple proxy service (that is, without additional scripting). For basic proxy operation, you must specify at least one `proxy-backend-addresses` option to specify the MySQL server to connect to by default:

```
shell> mysql-proxy --proxy-backend-addresses=MySQL.example.com:3306
```

The default proxy port is `4040`, so you can connect to your MySQL server through the proxy by specifying the host name and port details:

```
shell> mysql --host=localhost --port=4040
```

If your server requires authentication information, this will be passed through natively without alteration by `mysql-proxy`, so you must also specify the required authentication information:

```
shell> mysql --host=localhost --port=4040 \
--user=user_name --password=password
```

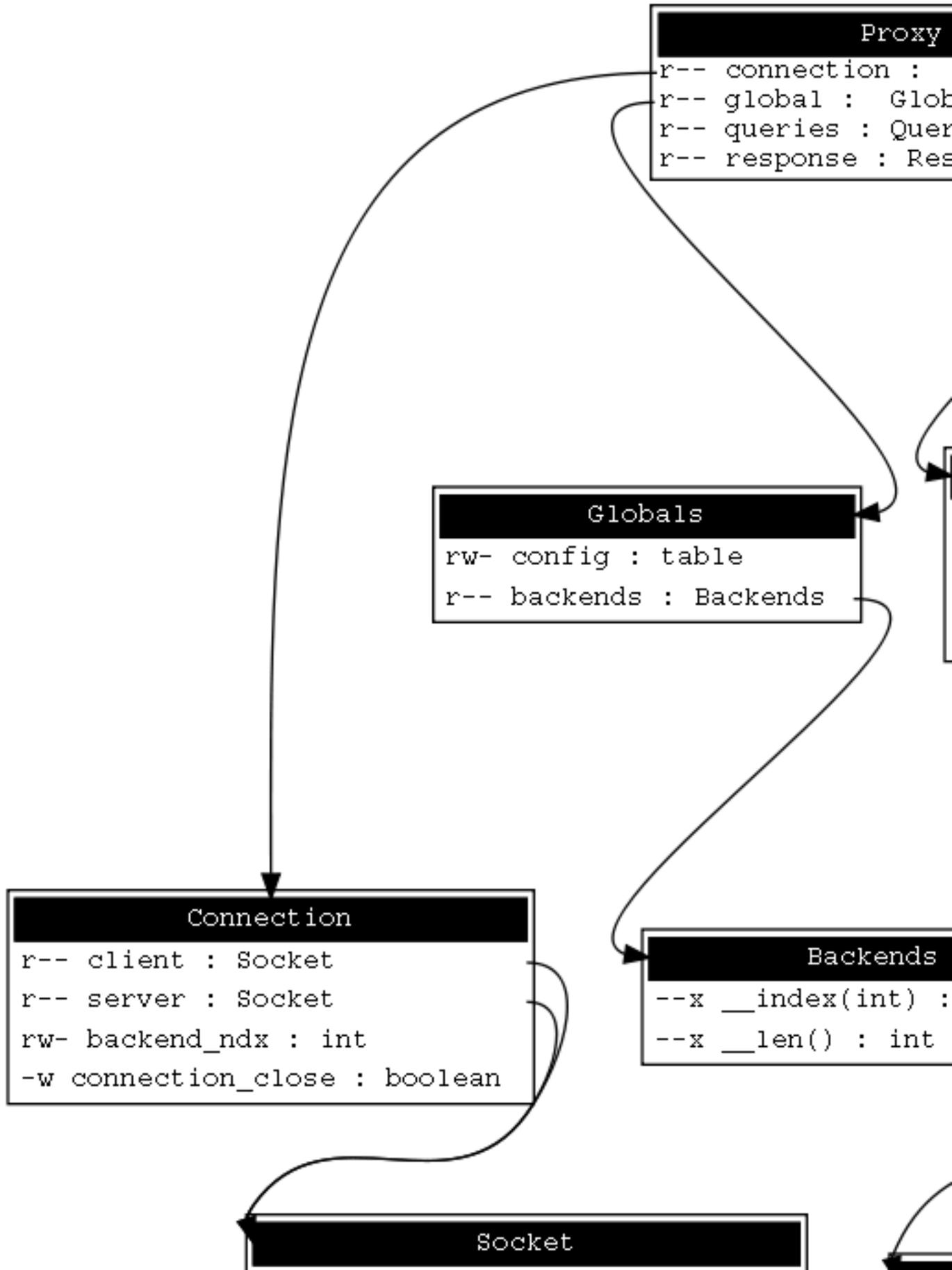
You can also connect to a read-only port (which filters out `UPDATE` and `INSERT` queries) by connecting to the read-only port. By default the host name is the default, and the port is `4042`, but you can alter the host/port information by using the `--proxy-read-only-backend-addresses [1810]` command-line option.

For more detailed information on how to use these command-line options, and `mysql-proxy` in general in combination with Lua scripts, see [Section 15.7.5, "Using MySQL Proxy"](#).

15.7.4. MySQL Proxy Scripting

You can control how MySQL Proxy manipulates and works with the queries and results that are passed on to the MySQL server through the use of the embedded Lua scripting language. You can find out more about the Lua programming language from the [Lua Web site](#).

The following diagram shows an overview of the classes exposed by MySQL Proxy.



The primary interaction between MySQL Proxy and the server is provided by defining one or more functions through an Lua script. A number of functions are supported, according to different events and operations in the communication sequence between a client and one or more backend MySQL servers:

- `connect_server()`: This function is called each time a connection is made to MySQL Proxy from a client. You can use this function during load-balancing to intercept the original connection and decide which server the client should ultimately be attached to. If you do not define a special solution, a simple round-robin style distribution is used by default.
- `read_handshake()`: This function is called when the initial handshake information is returned by the server. You can capture the handshake information returned and provide additional checks before the authorization exchange takes place.
- `read_auth()`: This function is called when the authorization packet (user name, password, default database) are submitted by the client to the server for authentication.
- `read_auth_result()`: This function is called when the server returns an authorization packet to the client indicating whether the authorization succeeded.
- `read_query()`: This function is called each time a query is sent by the client to the server. You can use this to edit and manipulate the original query, including adding new queries before and after the original statement. You can also use this function to return information directly to the client, bypassing the server, which can be useful to filter unwanted queries or queries that exceed known limits.
- `read_query_result()`: This function is called each time a result is returned from the server, providing you have manually injected queries into the query queue. If you have not explicitly injected queries within the `read_query()` function, this function is not triggered. You can use this to edit the result set, or to remove or filter the result sets generated from additional queries you injected into the queue when using `read_query()`.

The following table describes the direction of information flow at the point when the function is triggered.

Function	Supplied Information	Direction
<code>connect_server()</code>	None	Client to Server
<code>read_handshake()</code>	None	Server to Client
<code>read_auth()</code>	None	Client to Server
<code>read_auth_result()</code>	None	Server to Client
<code>read_query()</code>	Query	Client to Server
<code>read_query_result()</code>	Query result	Server to Client

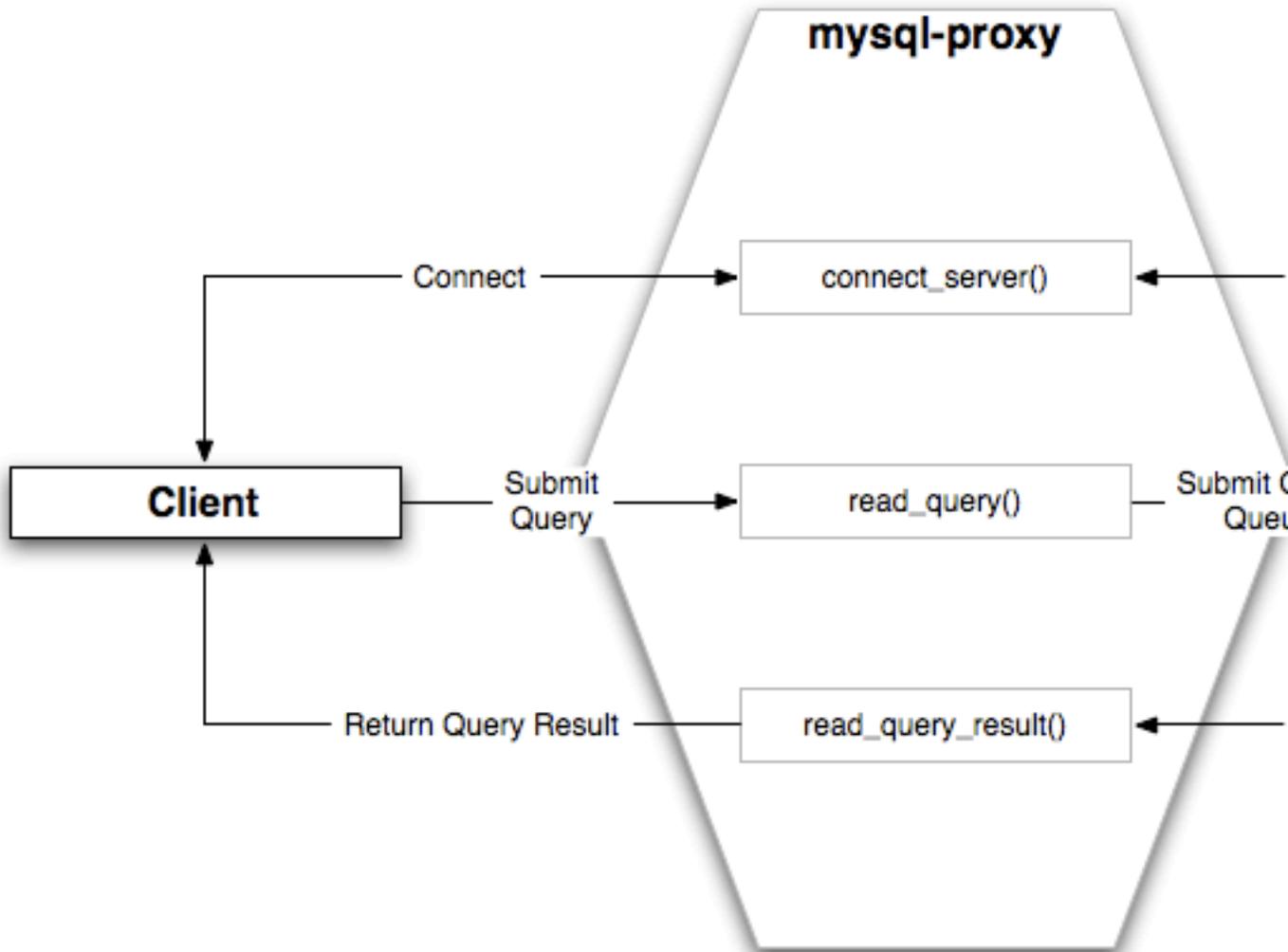
By default, all functions return a result that indicates whether the data should be passed on to the client or server (depending on the direction of the information being transferred). This return value can be overridden by explicitly returning a constant indicating that a particular response should be sent. For example, it is possible to construct result set information by hand within `read_query()` and to return the result set directly to the client without ever sending the original query to the server.

In addition to these functions, a number of built-in structures provide control over how MySQL Proxy forwards queries and returns the results by providing a simplified interface to elements such as the list of queries and the groups of result sets that are returned.

15.7.4.1. Proxy Scripting Sequence During Query Injection

The following figure gives an example of how the proxy might be used when injecting queries into the query queue. Because the proxy sits between the client and MySQL server, what the proxy sends to the

server, and the information that the proxy ultimately returns to the client, need not match or correlate. Once the client has connected to the proxy, the sequence shown in the following diagram occurs for each individual query sent by the client.



1. When the client submits one query to the proxy, the `read_query()` function within the proxy is triggered. The function adds the query to the query queue.
2. Once manipulation by `read_query()` has completed, the queries are submitted, sequentially, to the MySQL server.
3. The MySQL server returns the results from each query, one result set for each query submitted. The `read_query_result()` function is triggered for each result set, and each invocation can decide which result set to return to the client

For example, you can queue additional queries into the global query queue to be processed by the server. This can be used to add statistical information by adding queries before and after the original query, changing the original query:

```
SELECT * FROM City;
```

Into a sequence of queries:

```
SELECT NOW();
SELECT * FROM City;
SELECT NOW();
```

You can also modify the original statement; for example, to add `EXPLAIN` to each statement executed to get information on how the statement was processed, again altering our original SQL statement into a number of statements:

```
SELECT * FROM City;
EXPLAIN SELECT * FROM City;
```

In both of these examples, the client would have received more result sets than expected. Regardless of how you manipulate the incoming query and the returned result, the number of queries returned by the proxy must match the number of original queries sent by the client.

You could adjust the client to handle the multiple result sets sent by the proxy, but in most cases you will want the existence of the proxy to remain transparent. To ensure that the number of queries and result sets match, you can use the MySQL Proxy `read_query_result()` to extract the additional result set information and return only the result set the client originally requested back to the client. You can achieve this by giving each query that you add to the query queue a unique ID, then filter out queries that do not match the original query ID when processing them with `read_query_result()`.

15.7.4.2. Internal Structures

There are a number of internal structures within the scripting element of MySQL Proxy. The primary structure is `proxy` and this provides an interface to the many common structures used throughout the script, such as connection lists and configured backend servers. Other structures, such as the incoming packet from the client and result sets are only available within the context of one of the scriptable functions.

Attribute	Description
<code>connection</code>	A structure containing the active client connections. For a list of attributes, see <code>proxy.connection</code> [1817].
<code>servers</code>	A structure containing the list of configured backend servers. For a list of attributes, see <code>proxy.global.backends</code> [1818].
<code>queries</code>	A structure containing the queue of queries that will be sent to the server during a single client query. For a list of attributes, see <code>proxy.queries</code> [1818].
<code>PROXY_VERSION</code>	The version number of MySQL Proxy, encoded in hex. You can use this to check that the version number supports a particular option from within the Lua script. Note that the value is encoded as a hex value, so to check the version is at least 0.5.1 you compare against <code>0x00501</code> .

`proxy.connection`

The `proxy.connection` object is read only, and provides information about the current connection, and is split into a `client` and `server` tables. This enables you to examine information about both the incoming client connections to the proxy (`client`), and to the backend servers (`server`).

Attribute	Description
<code>client.default_db</code>	Default database requested by the client
<code>client.username</code>	User name used to authenticate
<code>client.scrambled_password</code>	The scrambled version of the password used to authenticate
<code>client.dst.name</code>	The combined <code>address:port</code> of the Proxy port used by this client (should match the <code>--proxy-address</code> [1810] configuration parameter)

Attribute	Description
<code>client.dst.address</code>	The IP address of the of the Proxy port used by this client
<code>client.dst.port</code>	The port number of the of the Proxy port used by this client
<code>client.src.name</code>	The combined <code>address:port</code> of the client (originating) TCP/IP endpoint
<code>client.src.address</code>	The IP address of the client (originating) TCP/IP port
<code>client.src.port</code>	The port of the client (originating) TCP/IP endpoint
<code>server.scramble_buffer</code>	The scramble buffer used to scramble the password
<code>server.mysql_d_version</code>	The MySQL version number of the server
<code>server.thread_id</code>	The ID of the thread handling the connection to the current server
<code>server.dst.name</code>	The combined <code>address:port</code> for the backend server for the current connection (i.e. the connection to the MySQL server)
<code>server.dst.address</code>	The address for the backend server
<code>server.dst.port</code>	The port for the backend server
<code>server.src.name</code>	The combined <code>address:port</code> for the TCP/IP endpoint used by the Proxy to connect to the backend server
<code>server.src.address</code>	The address of the endpoint for the proxy-side connection to the MySQL server
<code>server.src.port</code>	The port of the endpoint for the proxy-side connection to the MySQL server

`proxy.global.backends`

The `proxy.global.backends` table is partially writable and contains an array of all the configured backend servers and the server metadata (IP address, status, etc.). You can determine the array index of the current connection using `proxy.connection["backend_ndx"]` which is the index into this table of the backend server being used by the active connection.

The attributes for each entry within the `proxy.global.backends` table are shown in this table.

Attribute	Description
<code>dst.name</code>	The combined <code>address:port</code> of the backend server.
<code>dst.address</code>	The IP address of the backend server.
<code>dst.port</code>	The port of the backend server.
<code>connected_clients</code>	The number of clients currently connected.
<code>state</code>	The status of the backend server. See Backend State/Type Constants [1821] .
<code>type</code>	The type of the backend server. You can use this to identify whether the backed was configured as a standard read/write backend, or a read-only backend. You can compare this value to the <code>proxy.BACKEND_TYPE_RW</code> and <code>proxy.BACKEND_TYPE_RO</code> .

`proxy.queries`

The `proxy.queries` object is a queue representing the list of queries to be sent to the server. The queue is not populated automatically, but if you do not explicitly populate the queue, queries are passed on to the backend server verbatim. Also, if you do not populate the query queue by hand, the `read_query_result()` function is not triggered.

The following methods are supported for populating the `proxy.queries` object.

Function	Description
<code>append(id,packet,[options])</code>	Appends a query to the end of the query queue. The <code>id</code> is an integer identifier that you can use to recognize the query results when they are returned by the server. The packet should be a properly formatted query packet. The optional <code>options</code> should be a table containing the options specific to this packet.
<code>prepend(id,packet)</code>	Prepends a query to the query queue. The <code>id</code> is an identifier that you can use to recognize the query results when they are returned by the server. The packet should be a properly formatted query packet.
<code>reset()</code>	Empties the query queue.
<code>len()</code>	Returns the number of query packets in the queue.

For example, you could append a query packet to the `proxy.queries` queue by using the `append()`:

```
proxy.queries:append(1,packet)
```

The optional third argument to `append()` should contain the options for the packet. To have access to the result set through the `read_query_result()` function, set the `resultset_is_needed` flag to `true`:

```
proxy.queries:append( 1, packet, { resultset_is_needed = true } )
```

If that flag is `false` (the default), proxy will:

- Send the result set to the client as soon as it is received
- Reduce memory usage (because the result set is not stored internally for processing)
- Reduce latency of returning results to the client
- Pass data from server to client unaltered

The default mode is therefore quicker and useful if you only want to monitor the queries sent, and the basic statistics.

To perform any kind of manipulation on the returned data, you must set the flag to `true`, which will:

- Store the result set so that it can be processed.
- Enable modification of the result set before it is returned to the client.
- Enable you to discard the result set instead of returning it to the client.

`proxy.response`

The `proxy.response` structure is used when you want to return your own MySQL response, instead of forwarding a packet that you have received a backend server. The structure holds the response type information, an optional error message, and the result set (rows/columns) to return.

Attribute	Description
<code>type</code>	The type of the response. The type must be either <code>MYSQLD_PACKET_OK</code> or <code>MYSQLD_PACKET_ERR</code> . If the <code>MYSQLD_PACKET_ERR</code> , you should set the value of the <code>mysql.response.errmsg</code> with a suitable error message.

Attribute	Description
<code>errmsg</code>	A string containing the error message that will be returned to the client.
<code>resultset</code>	A structure containing the result set information (columns and rows), identical to what would be returned when returning a results from a <code>SELECT</code> query.

When using `proxy.response` you either set `proxy.response.type` to `proxy.MYSQLD_PACKET_OK` and then build `resultset` to contain the results to return, or set `proxy.response.type` to `proxy.MYSQLD_PACKET_ERR` and set the `proxy.response.errmsg` to a string with the error message. To send the completed result set or error message, you should return the `proxy.PROXY_SEND_RESULT` to trigger the return of the packet information.

An example of this can be seen in the `tutorial-resultset.lua` script within the MySQL Proxy package:

```
if string.lower(command) == "show" and string.lower(option) == "querycounter" then
    ---
    -- proxy.PROXY_SEND_RESULT requires
    --
    -- proxy.response.type to be either
    -- * proxy.MYSQLD_PACKET_OK or
    -- * proxy.MYSQLD_PACKET_ERR
    --
    -- for proxy.MYSQLD_PACKET_OK you need a resultset
    -- * fields
    -- * rows
    --
    -- for proxy.MYSQLD_PACKET_ERR
    -- * errmsg
    proxy.response.type = proxy.MYSQLD_PACKET_OK
    proxy.response.resultset = {
        fields = {
            { type = proxy.MYSQL_TYPE_LONG, name = "global_query_counter", },
            { type = proxy.MYSQL_TYPE_LONG, name = "query_counter", },
        },
        rows = {
            { proxy.global.query_counter, query_counter }
        }
    }

    -- we have our result, send it back
    return proxy.PROXY_SEND_RESULT
elseif string.lower(command) == "show" and string.lower(option) == "myerror" then
    proxy.response.type = proxy.MYSQLD_PACKET_ERR
    proxy.response.errmsg = "my first error"

    return proxy.PROXY_SEND_RESULT
```

`proxy.response.resultset`

The `proxy.response.resultset` structure should be populated with the rows and columns of data to return. The structure contains the information about the entire result set, with the individual elements of the data shown in the following table.

Attribute	Description
<code>fields</code>	The definition of the columns being returned. This should be a dictionary structure with the <code>type</code> specifying the MySQL data type, and the <code>name</code> specifying the column name. Columns should be listed in the order of the column data that will be returned.

Attribute	Description
<code>flags</code>	A number of flags related to the result set. Valid flags include <code>auto_commit</code> (whether an automatic commit was triggered), <code>no_good_index_used</code> (the query executed without using an appropriate index), and <code>no_index_used</code> (the query executed without using any index).
<code>rows</code>	The actual row data. The information should be returned as an array of arrays. Each inner array should contain the column data, with the outer array making up the entire result set.
<code>warning_count</code>	The number of warnings for this result set.
<code>affected_rows</code>	The number of rows affected by the original statement.
<code>insert_id</code>	The last insert ID for an auto-incremented column in a table.
<code>query_status</code>	The status of the query operation. You can use the <code>MYSQLD_PACKET_OK</code> or <code>MYSQLD_PACKET_ERR</code> constants to populate this parameter.

For an example showing how to use this structure, see `proxy.response` [1819].

Proxy Return State Constants

The following constants are used internally by the proxy to specify the response to send to the client or server. All constants are exposed as values within the main `proxy` table.

Constant	Description
<code>PROXY_SEND_QUERY</code>	Causes the proxy to send the current contents of the queries queue to the server.
<code>PROXY_SEND_RESULT</code>	Causes the proxy to send a result set back to the client.
<code>PROXY_IGNORE_RESULT</code>	Causes the proxy to drop the result set (nothing is returned to the client).

As constants, these entities are available without qualification in the Lua scripts. For example, at the end of the `read_query_result()` you might return `PROXY_IGNORE_RESULT`:

```
return proxy.PROXY_IGNORE_RESULT
```

Packet State Constants

The following states describe the status of a network packet. These items are entries within the main `proxy` table.

Constant	Description
<code>MYSQLD_PACKET_OK</code>	The packet is OK
<code>MYSQLD_PACKET_ERR</code>	The packet contains error information
<code>MYSQLD_PACKET_RAW</code>	The packet contains raw data

Backend State/Type Constants

The following constants are used either to define the status or type of the backend MySQL server to which the proxy is connected. These items are entries within the main `proxy` table.

Constant	Description
<code>BACKEND_STATE_UNKNOWN</code>	The current status is unknown

Constant	Description
BACKEND_STATE_UP	The backend is known to be up (available)
BACKEND_STATE_DOWN	The backend is known to be down (unavailable)
BACKEND_TYPE_UNKNOWN	Backend type is unknown
BACKEND_TYPE_RW	Backend is available for read/write
BACKEND_TYPE_RO	Backend is available only for read-only use

Server Command Constants

The following values are used in the packets exchanged between the client and server to identify the information in the rest of the packet. These items are entries within the main `proxy` table. The packet type is defined as the first character in the sent packet. For example, when intercepting packets from the client to edit or monitor a query, you would check that the first byte of the packet was of type `proxy.COM_QUERY`.

Constant	Description
COM_SLEEP	Sleep
COM_QUIT	Quit
COM_INIT_DB	Initialize database
COM_QUERY	Query
COM_FIELD_LIST	Field List
COM_CREATE_DB	Create database
COM_DROP_DB	Drop database
COM_REFRESH	Refresh
COM_SHUTDOWN	Shutdown
COM_STATISTICS	Statistics
COM_PROCESS_INFO	Process List
COM_CONNECT	Connect
COM_PROCESS_KILL	Kill
COM_DEBUG	Debug
COM_PING	Ping
COM_TIME	Time
COM_DELAYED_INSERT	Delayed insert
COM_CHANGE_USER	Change user
COM_BINLOG_DUMP	Binlog dump
COM_TABLE_DUMP	Table dump
COM_CONNECT_OUT	Connect out
COM_REGISTER_SLAVE	Register slave
COM_STMT_PREPARE	Prepare server-side statement
COM_STMT_EXECUTE	Execute server-side statement
COM_STMT_SEND_LONG_DATA	Long data
COM_STMT_CLOSE	Close server-side statement

Constant	Description
COM_STMT_RESET	Reset statement
COM_SET_OPTION	Set option
COM_STMT_FETCH	Fetch statement
COM_DAEMON	Daemon (MySQL 5.1 only)
COM_ERROR	Error

MySQL Type Constants

These constants are used to identify the field types in the query result data returned to clients from the result of a query. These items are entries within the main `proxy` table.

Constant	Field Type
MYSQL_TYPE_DECIMAL	Decimal
MYSQL_TYPE_NEWDECIMAL	Decimal (MySQL 5.0 or later)
MYSQL_TYPE_TINY	Tiny
MYSQL_TYPE_SHORT	Short
MYSQL_TYPE_LONG	Long
MYSQL_TYPE_FLOAT	Float
MYSQL_TYPE_DOUBLE	Double
MYSQL_TYPE_NULL	Null
MYSQL_TYPE_TIMESTAMP	Timestamp
MYSQL_TYPE_LONGLONG	Long long
MYSQL_TYPE_INT24	Integer
MYSQL_TYPE_DATE	Date
MYSQL_TYPE_TIME	Time
MYSQL_TYPE_DATETIME	Datetime
MYSQL_TYPE_YEAR	Year
MYSQL_TYPE_NEWDATE	Date (MySQL 5.0 or later)
MYSQL_TYPE_ENUM	Enumeration
MYSQL_TYPE_SET	Set
MYSQL_TYPE_TINY_BLOB	Tiny Blob
MYSQL_TYPE_MEDIUM_BLOB	Medium Blob
MYSQL_TYPE_LONG_BLOB	Long Blob
MYSQL_TYPE_BLOB	Blob
MYSQL_TYPE_VAR_STRING	Varstring
MYSQL_TYPE_STRING	String
MYSQL_TYPE_TINY	Tiny (compatible with <code>MYSQL_TYPE_CHAR</code>)
MYSQL_TYPE_ENUM	Enumeration (compatible with <code>MYSQL_TYPE_INTERVAL</code>)
MYSQL_TYPE_GEOMETRY	Geometry
MYSQL_TYPE_BIT	Bit

15.7.4.3. Capturing a Connection with `connect_server()`

When the proxy accepts a connection from a MySQL client, the `connect_server()` function is called.

There are no arguments to the function, but you can use and if necessary manipulate the information in the `proxy.connection` table, which is unique to each client session.

For example, if you have multiple backend servers, you can specify which server that connection should use by setting the value of `proxy.connection.backend_ndx` to a valid server number. The following code chooses between two servers based on whether the current time in minutes is odd or even:

```
function connect_server()
  print("--> a client really wants to talk to a server")
  if (tonumber(os.date("%M")) % 2 == 0) then
    proxy.connection.backend_ndx = 2
    print("Choosing backend 2")
  else
    proxy.connection.backend_ndx = 1
    print("Choosing backend 1")
  end
  print("Using " .. proxy.global.backends[proxy.connection.backend_ndx].dst.name)
end
```

This example also displays the IP address/port combination by accessing the information from the internal `proxy.global.backends` table.

15.7.4.4. Examining the Handshake with `read_handshake()`

Handshake information is sent by the server to the client after the initial connection (through `connect_server()`) has been made. The handshake information contains details about the MySQL version, the ID of the thread that will handle the connection information, and the IP address of the client and server. This information is exposed through the `proxy.connection` structure.

- `proxy.connection.server.mysql_d_version`: The version of the MySQL server.
- `proxy.connection.server.thread_id`: The thread ID.
- `proxy.connection.server.scramble_buffer`: The password scramble buffer.
- `proxy.connection.server.dst.name`: The IP address of the server.
- `proxy.connection.client.src.name`: The IP address of the client.

For example, you can print out the handshake data and refuse clients by IP address with the following function:

```
function read_handshake()
  print("<-- let's send him some information about us")
  print("  mysql_d-version: " .. proxy.connection.server.mysql_d_version)
  print("  thread-id      : " .. proxy.connection.server.thread_id)
  print("  scramble-buf   : " .. string.format("%q", proxy.connection.server.scramble_buffer))
  print("  server-addr    : " .. proxy.connection.server.dst.name)
  print("  client-addr    : " .. proxy.connection.client.dst.name)

  if not proxy.connection.client.src.name:match("^127.0.0.1:") then
    proxy.response.type = proxy.MYSQLD_PACKET_ERR
    proxy.response.errmsg = "only local connects are allowed"

    print("we don't like this client");

    return proxy.PROXY_SEND_RESULT
  end
end
```

Note that you must return an error packet to the client by using `proxy.PROXY_SEND_RESULT`.

15.7.4.5. Examining the Authentication Credentials with `read_auth()`

The `read_auth()` function is triggered when an authentication handshake is initiated by the client. In the execution sequence, `read_auth()` occurs immediately after `read_handshake()`, so the server selection has already been made, but the connection and authorization information has not yet been provided to the backend server.

You can obtain the authentication information by examining the `proxy.connection.client` structure. For more information, see `proxy.connection` [1817].

For example, you can print the user name and password supplied during authorization using:

```
function read_auth()
    print("    username      : " .. proxy.connection.client.username)
    print("    password      : " .. string.format("%q", proxy.connection.client.scrambled_password))
end
```

You can interrupt the authentication process within this function and return an error packet back to the client by constructing a new packet and returning `proxy.PROXY_SEND_RESULT`:

```
proxy.response.type = proxy.MYSQLD_PACKET_ERR
proxy.response.errmsg = "Logins are not allowed"
return proxy.PROXY_SEND_RESULT
```

15.7.4.6. Accessing Authentication Information with `read_auth_result()`

The return packet from the server during authentication is captured by `read_auth_result()`. The only argument to this function is the authentication packet returned by the server. As the packet is a raw MySQL network protocol packet, you must access the first byte to identify the packet type and contents. The `MYSQLD_PACKET_ERR` and `MYSQLD_PACKET_OK` constants can be used to identify whether the authentication was successful:

```
function read_auth_result(auth)
    local state = auth.packet:byte()

    if state == proxy.MYSQLD_PACKET_OK then
        print("<-- auth ok");
    elseif state == proxy.MYSQLD_PACKET_ERR then
        print("<-- auth failed");
    else
        print("<-- auth ... don't know: " .. string.format("%q", auth.packet));
    end
end
```

If a long-password capable client tries to authenticate to a server that supports long passwords, but the user password provided is actually short, `read_auth_result()` will be called twice. The first time, `auth.packet:byte()` will equal 254, indicating that the client should try again using the old password protocol. The second time `read_auth_result()/` is called, `auth.packet:byte()` will indicate whether the authentication actually succeeded.

15.7.4.7. Manipulating Queries with `read_query()`

The `read_query()` function is called once for each query submitted by the client and accepts a single argument, the query packet that was provided. To access the content of the packet, you must parse the packet contents manually.

For example, you can intercept a query packet and print out the contents using the following function definition:

```
function read_query( packet )
  if packet:byte() == proxy.COM_QUERY then
    print("we got a normal query: " .. packet:sub(2))
  end
end
```

This example checks the first byte of the packet to determine the type. If the type is `COM_QUERY` (see [Server Command Constants \[1822\]](#)), we extract the query from the packet and print it. The structure of the packet type supplied is important. In the case of a `COM_QUERY` packet, the remaining contents of the packet are the text of the query string. In this example, no changes have been made to the query or the list of queries that will ultimately be sent to the MySQL server.

To modify a query, or add new queries, you must populate the query queue (`proxy.queries`), then execute the queries that you have placed into the queue. If you do not modify the original query or the queue, the query received from the client is sent to the MySQL server verbatim.

When adding queries to the queue, you should follow these guidelines:

- The packets inserted into the queue must be valid query packets. For each packet, you must set the initial byte to the packet type. If you are appending a query, you can append the query statement to the rest of the packet.
- Once you add a query to the queue, the queue is used as the source for queries sent to the server. If you add a query to the queue to add more information, you must also add the original query to the queue or it will not be executed.
- Once the queue has been populated, you must set the return value from `read_query()` to indicate whether the query queue should be sent to the server.
- When you add queries to the queue, you should add an ID. The ID you specify is returned with the result set so that you identify each query and corresponding result set. The ID has no other purpose than as an identifier for correlating the query and result set. When operating in a passive mode, during profiling for example, you identify the original query and the corresponding result set so that the results expected by the client can be returned correctly.
- Unless your client is designed to cope with more result sets than queries, you should ensure that the number of queries from the client match the number of results sets returned to the client. Using the unique ID and removing result sets you inserted will help.

Normally, the `read_query()` and `read_query_result()` function are used in conjunction with each other to inject additional queries and remove the additional result sets. However, `read_query_result()` is only called if you populate the query queue within `read_query()`.

15.7.4.8. Manipulating Results with `read_query_result()`

The `read_query_result()` is called for each result set returned by the server only if you have manually injected queries into the query queue. If you have not manipulated the query queue, this function is not called. The function supports a single argument, the result packet, which provides a number of properties:

- `id`: The ID of the result set, which corresponds to the ID that was set when the query packet was submitted to the server when using `append(id)` on the query queue. You must have set the `resultset_is_needed` flag to `append` to intercept the result set before it is returned to the client. See [proxy.queries \[1818\]](#).
- `query`: The text of the original query.
- `query_time`: The number of microseconds required to receive the first row of a result set since the query was sent to the server.

- `response_time`: The number of microseconds required to receive the last row of the result set since the query was sent to the server.
- `resultset`: The content of the result set data.

By accessing the result information from the MySQL server, you can extract the results that match the queries that you injected, return different result sets (for example, from a modified query), and even create your own result sets.

The following Lua script, for example, will output the query, followed by the query time and response time (that is, the time to execute the query and the time to return the data for the query) for each query sent to the server:

```
function read_query( packet )
    if packet:byte() == proxy.COM_QUERY then
        print("we got a normal query: " .. packet:sub(2))

        proxy.queries:append(1, packet )

        return proxy.PROXY_SEND_QUERY
    end
end

function read_query_result(inj)
    print("query-time: " .. (inj.query_time / 1000) .. "ms")
    print("response-time: " .. (inj.response_time / 1000) .. "ms")
end
```

You can access the rows of returned results from the result set by accessing the `rows` property of the `resultset` property of the result that is exposed through `read_query_result()`. For example, you can iterate over the results showing the first column from each row using this Lua fragment:

```
for row in inj.resultset.rows do
    print("injected query returned: " .. row[1])
end
```

Just like `read_query()`, `read_query_result()` can return different values for each result according to the result returned. If you have injected additional queries into the query queue, for example, remove the results returned from those additional queries and return only the results from the query originally submitted by the client.

The following example injects additional `SELECT NOW()` statements into the query queue, giving them a different ID to the ID of the original query. Within `read_query_result()`, if the ID for the injected queries is identified, we display the result row, and return the `proxy.PROXY_IGNORE_RESULT` from the function so that the result is not returned to the client. If the result is from any other query, we print out the query time information for the query and return the default, which passes on the result set unchanged. We could also have explicitly returned `proxy.PROXY_IGNORE_RESULT` to the MySQL client.

```
function read_query( packet )
    if packet:byte() == proxy.COM_QUERY then
        proxy.queries:append(2, string.char(proxy.COM_QUERY) .. "SELECT NOW()", {resultset_is_needed = true})
        proxy.queries:append(1, packet, {resultset_is_needed = true})
        proxy.queries:append(2, string.char(proxy.COM_QUERY) .. "SELECT NOW()", {resultset_is_needed = true})

        return proxy.PROXY_SEND_QUERY
    end
end

function read_query_result(inj)
    if inj.id == 2 then
        for row in inj.resultset.rows do
            print("injected query returned: " .. row[1])
        end
    else
        print("query-time: " .. (inj.query_time / 1000) .. "ms")
        print("response-time: " .. (inj.response_time / 1000) .. "ms")
    end
end
```

```

        end
        return proxy.PROXY_IGNORE_RESULT
    else
        print("query-time: " .. (inj.query_time / 1000) .. "ms")
        print("response-time: " .. (inj.response_time / 1000) .. "ms")
    end
end
end

```

For further examples, see [Section 15.7.5, “Using MySQL Proxy”](#).

15.7.5. Using MySQL Proxy

There are a number of different ways to use MySQL Proxy. At the most basic level, you can allow MySQL Proxy to pass queries from clients to a single server. To use MySQL Proxy in this mode, you just have to specify on the command line the backend server to which the proxy should connect:

```
shell> mysql-proxy --proxy-backend-addresses=sakila:3306
```

If you specify multiple backend MySQL servers, the proxy connects each client to each server in a round-robin fashion. Suppose that you have two MySQL servers, A and B. The first client to connect is connected to server A, the second to server B, the third to server A. For example:

```
shell> mysql-proxy \
--proxy-backend-addresses=narcissus:3306 \
--proxy-backend-addresses=nostromo:3306
```

When you specify multiple servers in this way, the proxy automatically identifies when a MySQL server has become unavailable and marks it accordingly. New connections are automatically attached to a server that is available, and a warning is reported to the standard output from `mysql-proxy`:

```
network-mysqld.c.367: connect(nostromo:3306) failed: Connection refused
network-mysqld-proxy.c.2405: connecting to backend (nostromo:3306) failed, marking it as down for ...
```

Lua scripts enable a finer level of control, both over the connections and their distribution and how queries and result sets are processed. When using an Lua script, you must specify the name of the script on the command line using the `--proxy-lua-script [1812]` option:

```
shell> mysql-proxy --proxy-lua-script=mc.lua --proxy-backend-addresses=sakila:3306
```

When you specify a script, the script is not executed until a connection is made. This means that faults with the script are not raised until the script is executed. Script faults will not affect the distribution of queries to backend MySQL servers.



Note

Because a script is not read until the connection is made, you can modify the contents of the Lua script file while the proxy is still running and the modified script is automatically used for the next connection. This ensures that MySQL Proxy remains available because it need not be restarted for the changes to take effect.

15.7.5.1. Using the Administration Interface

The `mysql-proxy` administration interface can be accessed using any MySQL client using the standard protocols. You can use the administration interface to gain information about the proxy server as a whole - standard connections to the proxy are isolated to operate as if you were connected directly to the backend MySQL server.

In `mysql-proxy` 0.8.0 and earlier, a rudimentary interface was built into the proxy. In later versions this was replaced so that you must specify an administration script to be used when users connect to the administration interface.

To use the administration interface, specify the user name and password required to connect to the admin service, using the `--admin-username` [1807] and `--admin-password` [1807] options. You must also specify the Lua script to be used as the interface to the administration service by using the `admin-lua-script` [1806] script option to point to a Lua script.

For example, you can create a basic interface to the internal components of the `mysql-proxy` system using the following script, written by Diego Medina:

```
--[[
Copyright 2008, 2010, Oracle and/or its affiliates. All rights reserved.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; version 2 of the License.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
--]]

-- admin.lua

--[[
See http://www.chriscalender.com/?p=41
(Thanks to Chris Calender)
See http://datacharmer.blogspot.com/2009/01/mysql-proxy-is-back.html
(Thanks Giuseppe Maxia)
--]]

function set_error(errmsg)
    proxy.response = {
        type = proxy.MYSQLD_PACKET_ERR,
        errmsg = errmsg or "error"
    }
end

function read_query(packet)
    if packet:byte() ~= proxy.COM_QUERY then
        set_error("[admin] we only handle text-based queries (COM_QUERY)")
        return proxy.PROXY_SEND_RESULT
    end

    local query = packet:sub(2)
    local rows = { }
    local fields = { }

    -- try to match the string up to the first non-alphanum
    local f_s, f_e, command = string.find(packet, "^%s*(%w+)", 2)
    local option

    if f_e then
        -- if that match, take the next sub-string as option
        f_s, f_e, option = string.find(packet, "^%s+(%w+)", f_e + 1)
    end

    -- we got our commands, execute it
    if command == "show" and option == "querycounter" then
```

```

---
-- proxy.PROXY_SEND_RESULT requires
--
-- proxy.response.type to be either
-- * proxy.MYSQLD_PACKET_OK or
-- * proxy.MYSQLD_PACKET_ERR
--
-- for proxy.MYSQLD_PACKET_OK you need a resultset
-- * fields
-- * rows
--
-- for proxy.MYSQLD_PACKET_ERR
-- * errmsg
proxy.response.type = proxy.MYSQLD_PACKET_OK
proxy.response.resultset = {
    fields = {
        { type = proxy.MYSQL_TYPE_LONG, name = "query_counter", },
    },
    rows = {
        { proxy.global.query_counter }
    }
}

-- we have our result, send it back
return proxy.PROXY_SEND_RESULT
elseif command == "show" and option == "myerror" then
    proxy.response.type = proxy.MYSQLD_PACKET_ERR
    proxy.response.errmsg = "my first error"

    return proxy.PROXY_SEND_RESULT

elseif string.sub(packet, 2):lower() == 'select help' then
    return show_process_help()

elseif string.sub(packet, 2):lower() == 'show proxy processlist' then
    return show_process_table()

elseif query == "SELECT * FROM backends" then
    fields = {
        { name = "backend_ndx",
          type = proxy.MYSQL_TYPE_LONG },

        { name = "address",
          type = proxy.MYSQL_TYPE_STRING },
        { name = "state",
          type = proxy.MYSQL_TYPE_STRING },
        { name = "type",
          type = proxy.MYSQL_TYPE_STRING },
    }

    for i = 1, #proxy.global.backends do
        local b = proxy.global.backends[i]

        rows[#rows + 1] = {
            i, b.dst.name, b.state, b.type
        }
    end
else
    set_error()
    return proxy.PROXY_SEND_RESULT
end

proxy.response = {
    type = proxy.MYSQLD_PACKET_OK,
    resultset = {
        fields = fields,
        rows = rows
    }
}

```

```

    }
  }
  return proxy.PROXY_SEND_RESULT
end

function make_dataset (header, dataset)
  proxy.response.type = proxy.MYSQLD_PACKET_OK

  proxy.response.resultset = {
    fields = {},
    rows = {}
  }
  for i,v in pairs (header) do
    table.insert(proxy.response.resultset.fields, {type = proxy.MYSQL_TYPE_STRING, name = v})
  end
  for i,v in pairs (dataset) do
    table.insert(proxy.response.resultset.rows, v )
  end
  return proxy.PROXY_SEND_RESULT
end

function show_process_table()
  local dataset = {}
  local header = { 'Id', 'IP Address', 'Time' }
  local rows = {}
  for t_i, t_v in pairs (proxy.global.process) do
    for s_i, s_v in pairs ( t_v ) do
      table.insert(rows, { t_i, s_v.ip, os.date('%c',s_v.ts) })
    end
  end
  return make_dataset(header,rows)
end

function show_process_help()
  local dataset = {}
  local header = { 'command', 'description' }
  local rows = {
    {'SELECT HELP', 'This command.'},
    {'SHOW PROXY PROCESSLIST', 'Show all connections and their true IP Address.'},
  }
  return make_dataset(header,rows)
end

function dump_process_table()
  proxy.global.initialize_process_table()
  print('current contents of process table')
  for t_i, t_v in pairs (proxy.global.process) do
    print ('session id: ', t_i)
    for s_i, s_v in pairs ( t_v ) do
      print ( '\t', s_i, s_v.ip, s_v.ts )
    end
  end
  print ('---END PROCESS TABLE---')
end

--[[ Help

we use a simple string-match to split commands are word-boundaries

mysql> show querycounter

is split into
command = "show"
option = "querycounter"

spaces are ignored, the case has to be as is.

```

```
mysql> show myerror
returns a error-packet
--]]
```

The script works in combination with a main proxy script, [reporter.lua](#):

```
--[[
    Copyright 2008, 2010, Oracle and/or its affiliates. All rights reserved.

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; version 2 of the License.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
--]]

-- reporter.lua

--[[
    See http://www.chriscalender.com/?p=41
    (Thanks to Chris Calender)
    See http://datacharmer.blogspot.com/2009/01/mysql-proxy-is-back.html
    (Thanks Giuseppe Maxia)
--]]

proxy.global.query_counter = proxy.global.query_counter or 0

function proxy.global.initialize_process_table()
    if proxy.global.process == nil then
        proxy.global.process = {}
    end
    if proxy.global.process[proxy.connection.server.thread_id] == nil then
        proxy.global.process[proxy.connection.server.thread_id] = {}
    end
end

function read_auth_result( auth )
    local state = auth.packet:byte()
    if state == proxy.MYSQLD_PACKET_OK then
        proxy.global.initialize_process_table()
        table.insert( proxy.global.process[proxy.connection.server.thread_id],
            { ip = proxy.connection.client.src.name, ts = os.time() } )
    end
end

function disconnect_client()
    local connection_id = proxy.connection.server.thread_id
    if connection_id then
        -- client has disconnected, set this to nil
        proxy.global.process[connection_id] = nil
    end
end
```

```

---
-- read_query() can return a resultset
--
-- You can use read_query() to return a result-set.
--
-- @param packet the mysql-packet sent by the client
--
-- @return
-- * nothing to pass on the packet as is,
-- * proxy.PROXY_SEND_QUERY to send the queries from the proxy.queries queue
-- * proxy.PROXY_SEND_RESULT to send your own result-set
--
function read_query( packet )
    -- a new query came in in this connection
    -- using proxy.global.* to make it available to the admin plugin
    proxy.global.query_counter = proxy.global.query_counter + 1
end

```

To use the script, save the first script to a file (`admin.lua` in the following example) and the other to `reporter.lua`, then run `mysql-proxy` specifying the admin script and a backend MySQL server:

```

shell> mysql-proxy --admin-lua-script=admin.lua --admin-password=password \ »
--admin-username=root --proxy-backend-addresses=127.0.0.1:3306 -proxy-lua-script=reporter.lua

```

In a different window, connect to the MySQL server through the proxy:

```

shell> mysql --user=root --password=password --port=4040
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1798669
Server version: 5.0.70-log Gentoo Linux mysql-5.0.70-r1

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>

```

In another different window, connect to the `mysql-proxy` admin service using the specified user name and password:

```

shell> mysql --user=root --password=password --port=4041 --host=localhost
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.0.99-agent-admin

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>

```

To monitor the status of the proxy, ask for a list of the current active processes:

```

mysql> show proxy processlist;
+-----+-----+-----+
| Id      | IP Address          | Time                |
+-----+-----+-----+
| 1798669 | 192.168.0.112:52592 | Wed Jan 20 16:58:00 2010 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

For more information on the example, see [MySQL Proxy Admin Example](#).

15.7.6. MySQL Proxy FAQ

Questions

- [16.7.6.1: \[1835\]](#) In load balancing, how can I separate reads from writes?
- [16.7.6.2: \[1835\]](#) How do I use a socket with MySQL Proxy? Proxy change logs mention that support for UNIX sockets has been added.
- [16.7.6.3: \[1835\]](#) Can I use MySQL Proxy with all versions of MySQL?
- [16.7.6.4: \[1835\]](#) Can I run MySQL Proxy as a daemon?
- [16.7.6.5: \[1835\]](#) Do proxy applications run on a separate server? If not, what is the overhead incurred by Proxy on the DB server side?
- [16.7.6.6: \[1836\]](#) With load balancing, what happens to transactions? Are all queries sent to the same server?
- [16.7.6.7: \[1836\]](#) Is it possible to use MySQL Proxy with updating a Lucene index (or Solr) by making TCP calls to that server to update?
- [16.7.6.8: \[1836\]](#) Is the system context switch expensive, how much overhead does the Lua script add?
- [16.7.6.9: \[1836\]](#) How much latency does a proxy add to a connection?
- [16.7.6.10: \[1836\]](#) Do you have to make one large script and call it at proxy startup, can I change scripts without stopping and restarting (interrupting) the proxy?
- [16.7.6.11: \[1836\]](#) If MySQL Proxy has to live on same machine as MySQL, are there any tuning considerations to ensure both perform optimally?
- [16.7.6.12: \[1836\]](#) I currently use SQL Relay for efficient connection pooling with a number of Apache processes connecting to a MySQL server. Can MySQL Proxy currently accomplish this? My goal is to minimize connection latency while keeping temporary tables available.
- [16.7.6.13: \[1836\]](#) Are these reserved function names (for example, `error_result()`) that get automatically called?
- [16.7.6.14: \[1836\]](#) As the script is re-read by MySQL Proxy, does it cache this or is it looking at the file system with each request?
- [16.7.6.15: \[1836\]](#) Given that there is a `connect_server()` function, can a Lua script link up with multiple servers?
- [16.7.6.16: \[1836\]](#) Is the MySQL Proxy an API?
- [16.7.6.17: \[1837\]](#) The global namespace variable example with quotas does not persist after a reboot, is that correct?
- [16.7.6.18: \[1837\]](#) Can MySQL Proxy handle SSL connections?
- [16.7.6.19: \[1837\]](#) Could MySQL Proxy be used to capture passwords?
- [16.7.6.20: \[1837\]](#) Are there tools for isolating problems? How can someone figure out whether a problem is in the client, the database, or the proxy?
- [16.7.6.21: \[1837\]](#) Is MySQL Proxy similar to what is provided by Java connection pools?
- [16.7.6.22: \[1837\]](#) So authentication with connection pooling has to be done at every connection? What is the authentication latency?
- [16.7.6.23: \[1837\]](#) If you have multiple databases on the same box, can you use proxy to connect to databases on default port 3306?

- [16.7.6.24: \[1837\]](#) What about caching the authorization information so clients connecting are given back-end connections that were established with identical authorization information, thus saving a few more round trips?
- [16.7.6.25: \[1837\]](#) Is there any big web site using MySQL Proxy? For what purpose and what transaction rate have they achieved?
- [16.7.6.26: \[1837\]](#) How does MySQL Proxy compare to DBSLayer?
- [16.7.6.27: \[1837\]](#) I tried using MySQL Proxy without any Lua script to try a round-robin type load balancing. In this case, if the first database in the list is down, MySQL Proxy would not connect the client to the second database in the list.
- [16.7.6.28: \[1838\]](#) Is it “safe” to use `LuaSocket` with proxy scripts?
- [16.7.6.29: \[1838\]](#) How different is MySQL Proxy from DBCP (Database connection pooling) for Apache in terms of connection pooling?
- [16.7.6.30: \[1838\]](#) MySQL Proxy can handle about 5000 connections, what is the limit on a MySQL server?
- [16.7.6.31: \[1838\]](#) Would the Java-only connection pooling solution work for multiple web servers? With this, I would assume that you can pool across many web servers at once?

Questions and Answers

16.7.6.1: In load balancing, how can I separate reads from writes?

There is no automatic separation of queries that perform reads or writes to the different backend servers. However, you can specify to `mysql-proxy` that one or more of the “backend” MySQL servers are read only.

```
shell> mysql-proxy \  
--proxy-backend-addresses=10.0.1.2:3306 \  
--proxy-read-only-backend-addresses=10.0.1.3:3306 &
```

16.7.6.2: How do I use a socket with MySQL Proxy? Proxy change logs mention that support for UNIX sockets has been added.

Specify the path to the socket:

```
--proxy-backend-addresses=/path/to/socket
```

16.7.6.3: Can I use MySQL Proxy with all versions of MySQL?

MySQL Proxy is designed to work with MySQL 5.0 or higher, and supports the MySQL network protocol for 5.0 and higher.

16.7.6.4: Can I run MySQL Proxy as a daemon?

Use the `--daemon [1807]` option. To keep track of the process ID, the daemon can be started with the `--pid-file=file [1812]` option to save the PID to a known file name. On version 0.5.x, the Proxy cannot be started natively as a daemon.

16.7.6.5: Do proxy applications run on a separate server? If not, what is the overhead incurred by Proxy on the DB server side?

You can run the proxy on the application server, on its own box, or on the DB-server depending on the use case.

16.7.6.6: With load balancing, what happens to transactions? Are all queries sent to the same server?

Without any special customization the whole connection is sent to the same server. That keeps the whole connection state intact.

16.7.6.7: Is it possible to use MySQL Proxy with updating a Lucene index (or Solr) by making TCP calls to that server to update?

Yes, but it is not advised for now.

16.7.6.8: Is the system context switch expensive, how much overhead does the Lua script add?

Lua is fast and the overhead should be small enough for most applications. The raw packet overhead is around 400 microseconds.

16.7.6.9: How much latency does a proxy add to a connection?

In the range of 400 microseconds per request.

16.7.6.10: Do you have to make one large script and call it at proxy startup, can I change scripts without stopping and restarting (interrupting) the proxy?

You can just change the script and the proxy will reload it when a client connects.

16.7.6.11: If MySQL Proxy has to live on same machine as MySQL, are there any tuning considerations to ensure both perform optimally?

MySQL Proxy can live on any box: application, database, or its own box. MySQL Proxy uses comparatively little CPU or RAM, with negligible additional requirements or overhead.

16.7.6.12: I currently use SQL Relay for efficient connection pooling with a number of Apache processes connecting to a MySQL server. Can MySQL Proxy currently accomplish this? My goal is to minimize connection latency while keeping temporary tables available.

Yes.

16.7.6.13: Are these reserved function names (for example, `error_result()`) that get automatically called?

Only functions and values starting with `proxy.*` are provided by the proxy. All others are user provided.

16.7.6.14: As the script is re-read by MySQL Proxy, does it cache this or is it looking at the file system with each request?

It looks for the script at client-connect and reads it if it has changed, otherwise it uses the cached version.

16.7.6.15: Given that there is a `connect_server()` function, can a Lua script link up with multiple servers?

MySQL Proxy provides some tutorials in the source package; one is [examples/tutorial-keepalive.lua](#).

16.7.6.16: Is the MySQL Proxy an API?

No, MySQL Proxy is an application that forwards packets from a client to a server using the MySQL network protocol. The MySQL Proxy provides a API allowing you to change its behavior.

16.7.6.17: The global namespace variable example with quotas does not persist after a reboot, is that correct?

Yes. If you restart the proxy, you lose the results, unless you save them in a file.

16.7.6.18: Can MySQL Proxy handle SSL connections?

No, being the man-in-the-middle, Proxy cannot handle encrypted sessions because it cannot share the SSL information.

16.7.6.19: Could MySQL Proxy be used to capture passwords?

The MySQL network protocol does not allow passwords to be sent in cleartext, all you could capture is the encrypted version.

16.7.6.20: Are there tools for isolating problems? How can someone figure out whether a problem is in the client, the database, or the proxy?

You can set a debug script in the proxy, which is an exceptionally good tool for this purpose. You can see very clearly which component is causing the problem, if you set the right breakpoints.

16.7.6.21: Is MySQL Proxy similar to what is provided by Java connection pools?

Yes and no. Java connection pools are specific to Java applications, MySQL Proxy works with any client API that talks the MySQL network protocol. Also, connection pools do not provide any functionality for intelligently examining the network packets and modifying the contents.

16.7.6.22: So authentication with connection pooling has to be done at every connection? What is the authentication latency?

You can skip the round-trip and use the connection as it was added to the pool. As long as the application cleans up the temporary tables it used. The overhead is (as always) around 400 microseconds.

16.7.6.23: If you have multiple databases on the same box, can you use proxy to connect to databases on default port 3306?

Yes, MySQL Proxy can listen on any port, provided that none of the MySQL servers are listening on the same port.

16.7.6.24: What about caching the authorization information so clients connecting are given back-end connections that were established with identical authorization information, thus saving a few more round trips?

There is an `--proxy-pool-no-change-user` [1811] option that provides this functionality.

16.7.6.25: Is there any big web site using MySQL Proxy? For what purpose and what transaction rate have they achieved?

Yes, [gaiaonline](#). They have tested MySQL Proxy and seen it handle 2400 queries per second through the proxy.

16.7.6.26: How does MySQL Proxy compare to DBSLayer?

DBSLayer is a REST->MySQL tool, MySQL Proxy is transparent to your application. No change to the application is needed.

16.7.6.27: I tried using MySQL Proxy without any Lua script to try a round-robin type load balancing. In this case, if the first database in the list is down, MySQL Proxy would not connect the client to the second database in the list.

This issue is fixed in version 0.7.0.

16.7.6.28: Is it “safe” to use [LuaSocket](#) with proxy scripts?

You can, but it is not advised because it may block.

16.7.6.29: How different is MySQL Proxy from DBCP (Database connection pooling) for Apache in terms of connection pooling?

Connection Pooling is just one use case of the MySQL Proxy. You can use it for a lot more and it works in cases where you cannot use DBCP (for example, if you do not have Java).

16.7.6.30: MySQL Proxy can handle about 5000 connections, what is the limit on a MySQL server?

The server limit is given by the value of the [max_connections \[519\]](#) system variable. The default value is version dependent.

16.7.6.31: Would the Java-only connection pooling solution work for multiple web servers? With this, I would assume that you can pool across many web servers at once?

Yes. But you can also start one proxy on each application server to get a similar behavior as you have it already.

Chapter 16. Replication

Table of Contents

16.1. Replication Configuration	1840
16.1.1. How to Set Up Replication	1841
16.1.2. Replication Formats	1850
16.1.3. Replication and Binary Logging Options and Variables	1858
16.1.4. Common Replication Administration Tasks	1905
16.2. Replication Implementation	1907
16.2.1. Replication Implementation Details	1908
16.2.2. Replication Relay and Status Logs	1910
16.2.3. How Servers Evaluate Replication Filtering Rules	1913
16.3. Replication Solutions	1919
16.3.1. Using Replication for Backups	1920
16.3.2. Using Replication with Different Master and Slave Storage Engines	1923
16.3.3. Using Replication for Scale-Out	1925
16.3.4. Replicating Different Databases to Different Slaves	1926
16.3.5. Improving Replication Performance	1927
16.3.6. Switching Masters During Failover	1928
16.3.7. Setting Up Replication Using SSL	1931
16.3.8. Semisynchronous Replication	1932
16.4. Replication Notes and Tips	1937
16.4.1. Replication Features and Issues	1937
16.4.2. Replication Compatibility Between MySQL Versions	1960
16.4.3. Upgrading a Replication Setup	1962
16.4.4. Troubleshooting Replication	1963
16.4.5. How to Report Replication Bugs or Problems	1964

Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is asynchronous by default - slaves need not to be connected permanently to receive updates from the master. This means that updates can occur over long-distance connections and even over temporary or intermittent connections such as a dial-up service. Depending on the configuration, you can replicate all databases, selected databases, or even selected tables within a database.

For answers to some questions often asked by those who are new to MySQL Replication, see [Section B.13, “MySQL 5.5 FAQ: Replication”](#).

The target uses for replication in MySQL include:

- Scale-out solutions - spreading the load among multiple slaves to improve performance. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while dramatically increasing read speed across an increasing number of slaves.
- Data security - because data is replicated to the slave, and the slave can pause the replication process, it is possible to run backup services on the slave without corrupting the corresponding master data.
- Analytics - live data can be created on the master, while the analysis of the information can take place on the slave without affecting the performance of the master.

- Long-distance data distribution - if a branch office would like to work with a copy of your main data, you can use replication to create a local copy of the data for their use without requiring permanent access to the master.

Replication in MySQL features support for one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves. This is in contrast to the *synchronous* replication which is a characteristic of MySQL Cluster (see [Chapter 17, MySQL Cluster NDB 7.2](#)). In MySQL 5.5, an interface to semisynchronous replication is supported in addition to the built-in asynchronous replication. With semisynchronous replication, a commit performed on the master side blocks before returning to the session that performed the transaction until at least one slave acknowledges that it has received and logged the events for the transaction. See [Section 16.3.8, “Semisynchronous Replication”](#)

There are a number of solutions available for setting up replication between two servers, but the best method to use depends on the presence of data and the engine types you are using. For more information on the available options, see [Section 16.1.1, “How to Set Up Replication”](#).

There are two core types of replication format, Statement Based Replication (SBR), which replicates entire SQL statements, and Row Based Replication (RBR), which replicates only the changed rows. You may also use a third variety, Mixed Based Replication (MBR). For more information on the different replication formats, see [Section 16.1.2, “Replication Formats”](#). In MySQL 5.5, statement-based format is the default.

Replication is controlled through a number of different options and variables. These control the core operation of the replication, timeouts, and the databases and filters that can be applied on databases and tables. For more information on the available options, see [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).

You can use replication to solve a number of different problems, including problems with performance, supporting the backup of different databases, and as part of a larger solution to alleviate system failures. For information on how to address these issues, see [Section 16.3, “Replication Solutions”](#).

For notes and tips on how different data types and statements are treated during replication, including details of replication features, version compatibility, upgrades, and problems and their resolution, including an FAQ, see [Section 16.4, “Replication Notes and Tips”](#).

For detailed information on the implementation of replication, how replication works, the process and contents of the binary log, background threads and the rules used to decide how statements are recorded and replication, see [Section 16.2, “Replication Implementation”](#).

16.1. Replication Configuration

Replication between servers in MySQL is based on the binary logging mechanism. The MySQL instance operating as the master (the source of the database changes) writes updates and changes as “events” to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Slaves are configured to read the binary log from the master and to execute the events in the binary log on the slave’s local database.

The master is “dumb” in this scenario. Once binary logging has been enabled, all statements are recorded in the binary log. Each slave receives a copy of the entire contents of the binary log. It is the responsibility of the slave to decide which statements in the binary log should be executed; you cannot configure the master to log only certain events. If you do not specify otherwise, all events in the master binary log are executed on the slave. If required, you can configure the slave to process only events that apply to particular databases or tables.

Each slave keeps a record of the binary log coordinates: The file name and position within the file that it has read and processed from the master. This means that multiple slaves can be connected to the master

and executing different parts of the same binary log. Because the slaves control this process, individual slaves can be connected and disconnected from the server without affecting the master's operation. Also, because each slave remembers the position within the binary log, it is possible for slaves to be disconnected, reconnect and then “catch up” by continuing from the recorded position.

Both the master and each slave must be configured with a unique ID (using the `server-id` [1858] option). In addition, each slave must be configured with information about the master host name, log file name, and position within that file. These details can be controlled from within a MySQL session using the `CHANGE MASTER TO` statement on the slave. The details are stored within the slave's `master.info` file.

This section describes the setup and configuration required for a replication environment, including step-by-step instructions for creating a new replication environment. The major components of this section are:

- For a guide to setting up two or more servers for replication, [Section 16.1.1, “How to Set Up Replication”](#), deals with the configuration of the systems and provides methods for copying data between the master and slaves.
- Events in the binary log are recorded using a number of formats. These are referred to as statement-based replication (SBR) or row-based replication (RBR). A third type, mixed-format replication (MIXED), uses SBR or RBR replication automatically to take advantage of the benefits of both SBR and RBR formats when appropriate. The different formats are discussed in [Section 16.1.2, “Replication Formats”](#).
- Detailed information on the different configuration options and variables that apply to replication is provided in [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).
- Once started, the replication process should require little administration or monitoring. However, for advice on common tasks that you may want to execute, see [Section 16.1.4, “Common Replication Administration Tasks”](#).

16.1.1. How to Set Up Replication

This section describes how to set up complete replication of a MySQL server. There are a number of different methods for setting up replication, and the exact method to use depends on how you are setting up replication, and whether you already have data within your master database.

There are some generic tasks that are common to all replication setups:

- On the master, you must enable binary logging and configure a unique server ID. This might require a server restart. See [Section 16.1.1.1, “Setting the Replication Master Configuration”](#).
- On each slave that you want to connect to the master, you must configure a unique server ID. This might require a server restart. See [Section 16.1.1.2, “Setting the Replication Slave Configuration”](#).
- You may want to create a separate user that will be used by your slaves to authenticate with the master to read the binary log for replication. The step is optional. See [Section 16.1.1.3, “Creating a User for Replication”](#).
- Before creating a data snapshot or starting the replication process, you should record the position of the binary log on the master. You will need this information when configuring the slave so that the slave knows where within the binary log to start executing events. See [Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”](#).
- If you already have data on your master and you want to use it to synchronize your slave, you will need to create a data snapshot. You can create a snapshot using `mysqldump` (see [Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”](#)) or by copying the data files directly (see [Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”](#)).

- You will need to configure the slave with settings for connecting to the master, such as the host name, login credentials, and binary log file name and position. See [Section 16.1.1.10, “Setting the Master Configuration on the Slave”](#).

Once you have configured the basic options, you will need to follow the instructions for your replication setup. A number of alternatives are provided:

- If you are establishing a new MySQL master and one or more slaves, you need only set up the configuration, as you have no data to exchange. For guidance on setting up replication in this situation, see [Section 16.1.1.7, “Setting Up Replication with New Master and Slaves”](#).
- If you are already running a MySQL server, and therefore already have data that must be transferred to your slaves before replication starts, have not previously configured the binary log and are able to shut down your MySQL server for a short period during the process, see [Section 16.1.1.8, “Setting Up Replication with Existing Data”](#).
- If you are adding slaves to an existing replication environment, you can set up the slaves without affecting the master. See [Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”](#).

If you will be administering MySQL replication servers, we suggest that you read this entire chapter through and try all statements mentioned in [Section 13.4.1, “SQL Statements for Controlling Master Servers”](#), and [Section 13.4.2, “SQL Statements for Controlling Slave Servers”](#). You should also familiarize yourself with the replication startup options described in [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).



Note

Note that certain steps within the setup process require the [SUPER \[697\]](#) privilege. If you do not have this privilege, it might not be possible to enable replication.

16.1.1.1. Setting the Replication Master Configuration

On a replication master, you must enable binary logging and establish a unique server ID. If this has not already been done, this part of master setup requires a server restart.

Binary logging *must* be enabled on the master because the binary log is the basis for sending data changes from the master to its slaves. If binary logging is not enabled, replication will not be possible.

Each server within a replication group must be configured with a unique server ID. This ID is used to identify individual servers within the group, and must be a positive integer between 1 and $(2^{32})-1$. How you organize and select the numbers is entirely up to you.

To configure the binary log and server ID options, you will need to shut down your MySQL server and edit the `my.cnf` or `my.ini` file. Add the following options to the configuration file within the `[mysqld]` section. If these options already exist, but are commented out, uncomment the options and alter them according to your needs. For example, to enable binary logging using a log file name prefix of `mysql-bin`, and configure a server ID of 1, use these lines:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

After making the changes, restart the server.



Note

If you omit `server-id [1858]` (or set it explicitly to its default value of 0), a master refuses connections from all slaves.

**Note**

For the greatest possible durability and consistency in a replication setup using **InnoDB** with transactions, you should use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in the master `my.cnf` file.

**Note**

Ensure that the `skip-networking` [461] option is not enabled on your replication master. If networking has been disabled, your slave will not be able to communicate with the master and replication will fail.

16.1.1.2. Setting the Replication Slave Configuration

On a replication slave, you must establish a unique server ID. If this has not already been done, this part of slave setup requires a server restart.

If the slave server ID is not already set, or the current value conflicts with the value that you have chosen for the master server, you should shut down your slave server and edit the configuration to specify a unique server ID. For example:

```
[mysqld]
server-id=2
```

After making the changes, restart the server.

If you are setting up multiple slaves, each one must have a unique `server-id` [1858] value that differs from that of the master and from each of the other slaves. Think of `server-id` [1858] values as something similar to IP addresses: These IDs uniquely identify each server instance in the community of replication partners.

**Note**

If you omit `server-id` [1858] (or set it explicitly to its default value of 0), a slave refuses to connect to a master.

You do not have to enable binary logging on the slave for replication to be enabled. However, if you enable binary logging on the slave, you can use the binary log for data backups and crash recovery on the slave, and also use the slave as part of a more complex replication topology (for example, where the slave acts as a master to other slaves).

16.1.1.3. Creating a User for Replication

Each slave must connect to the master using a MySQL user name and password, so there must be a user account on the master that the slave can use to connect. Any account can be used for this operation, providing it has been granted the `REPLICATION SLAVE` [697] privilege. You may wish to create a different account for each slave, or connect to the master using the same account for each slave.

You need not create an account specifically for replication. However, you should be aware that the user name and password will be stored in plain text within the `master.info` file (see [Section 16.2.2.2, “Slave Status Logs”](#)). Therefore, you may want to create a separate account that has privileges only for the replication process, to minimize the possibility of compromise to other accounts.

To create a new account, use `CREATE USER`. To grant this account the privileges required for replication, use the `GRANT` statement. If you create an account solely for the purposes of replication, that account

needs only the `REPLICATION SLAVE` [697] privilege. For example, to set up a new user, `repl`, that can connect for replication from any host within the `mydomain.com` domain, issue these statements on the master:

```
mysql> CREATE USER 'repl'@'%mydomain.com' IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%mydomain.com';
```

See [Section 13.7.1, “Account Management Statements”](#), for more information on statements for manipulation of user accounts.

16.1.1.4. Obtaining the Replication Master Binary Log Coordinates

To configure replication on the slave you must determine the master's current coordinates within its binary log. You will need this information so that when the slave starts the replication process, it is able to start processing events from the binary log at the correct point.

If you have existing data on your master that you want to synchronize on your slaves before starting the replication process, you must stop processing statements on the master, and then obtain its current binary log coordinates and dump its data, before permitting the master to continue executing statements. If you do not stop the execution of statements, the data dump and the master status information that you use will not match and you will end up with inconsistent or corrupted databases on the slaves.

To obtain the master binary log coordinates, follow these steps:

1. Start a session on the master by connecting to it with the command-line client, and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

For `InnoDB` tables, note that `FLUSH TABLES WITH READ LOCK` also blocks `COMMIT` operations.



Warning

Leave the client from which you issued the `FLUSH TABLES` statement running so that the read lock remains in effect. If you exit the client, the lock is released.

2. In a different session on the master, use the `SHOW MASTER STATUS` statement to determine the current binary log file name and position:

```
mysql > SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000003 | 73      | test         | manual,mysql      |
+-----+-----+-----+-----+
```

The `File` column shows the name of the log file and `Position` shows the position within the file. In this example, the binary log file is `mysql-bin.000003` and the position is 73. Record these values. You need them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

If the master has been running previously without binary logging enabled, the log file name and position values displayed by `SHOW MASTER STATUS` or `mysqldump --master-data` will be empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string (`'`) and 4.

You now have the information you need to enable the slave to start reading from the binary log in the correct place to start replication.

If you have existing data that needs to be synchronized with the slave before you start replication, leave the client running so that the lock remains in place and then proceed to [Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”](#), or [Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”](#). The idea here is to prevent any further changes so that the data copied to the slaves is in synchrony with the master.

If you are setting up a brand new master and slave replication group, you can exit the first session to release the read lock.

16.1.1.5. Creating a Data Snapshot Using `mysqldump`

One way to create a snapshot of the data in an existing master database is to use the `mysqldump` tool. Once the data dump has been completed, you then import this data into the slave before starting the replication process.

To obtain a snapshot of the data using `mysqldump`:

1. If you have not already locked the tables on the server to prevent statements that update data from executing:

Start a session on the server by connecting to it with the command-line client, and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Remember to use `SHOW MASTER STATUS` and record the binary log details for use when starting up the slave. The point in time of your snapshot and the binary log position must match. See [Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”](#).

2. In another session, use `mysqldump` to create a dump either of all the databases you want to replicate, or of selected individual databases. For example:

```
shell> mysqldump --all-databases --lock-all-tables >dbdump.db
```

An alternative to using a bare dump, is to use the `--master-data` option, which automatically appends the `CHANGE MASTER TO` statement required on the slave to start the replication process.

```
shell> mysqldump --all-databases --master-data >dbdump.db
```

3. In the client where you acquired the read lock, release the lock:

```
mysql> UNLOCK TABLES;
```

Alternatively, exit the first session to release the read lock.

When choosing databases to include in the dump, remember to filter out databases on each slave that you do not want to include in the replication process.

Either to copy the dump file to the slave, or use the file from the master when connecting remotely to the slave to import the data.

16.1.1.6. Creating a Data Snapshot Using Raw Data Files

If your database is large, copying the raw data files can be more efficient than using `mysqldump` and importing the file on each slave. This technique skips the overhead of updating indexes as the `INSERT` statements are replayed.

Using this method with tables in storage engines with complex caching or logging algorithms requires extra steps to produce a perfect “point in time” snapshot: the initial copy command might leave out cache

information and logging updates, even if you have acquired a global read lock. How the storage engine responds to this depends on its crash recovery abilities.

This method also does not work reliably if the master and slave have different values for `ft_stopword_file` [502], `ft_min_word_len` [501], or `ft_max_word_len` [501] and you are copying tables having full-text indexes.

If you use `InnoDB` tables, you can use the `mysqlbackup` command from the MySQL Enterprise Backup component to produce a consistent snapshot. This command records the log name and offset corresponding to the snapshot to be later used on the slave. MySQL Enterprise Backup is a commercial product that is included as part of a MySQL Enterprise subscription. See [Section 24.2, “MySQL Enterprise Backup”](#) for detailed information.

Otherwise, use the `cold backup` technique to obtain a reliable binary snapshot of `InnoDB` tables: copy all data files after doing a `slow shutdown` of the MySQL Server.

To create a raw data snapshot of `MyISAM` tables, you can use standard copy tools such as `cp` or `copy`, a remote copy tool such as `scp` or `rsync`, an archiving tool such as `zip` or `tar`, or a file system snapshot tool such as `dump`, providing that your MySQL data files exist on a single file system. If you are replicating only certain databases, copy only those files that relate to those tables. (For `InnoDB`, all tables in all databases are stored in the `system tablespace` files, unless you have the `innodb_file_per_table` [1555] option enabled.)

You might want to specifically exclude the following files from your archive:

- Files relating to the `mysql` database.
- The `master.info` file.
- The master's binary log files.
- Any relay log files.

To get the most consistent results with a raw data snapshot, shut down the master server during the process, as follows:

1. Acquire a read lock and get the master's status. See [Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”](#).
2. In a separate session, shut down the master server:

```
shell> mysqladmin shutdown
```

3. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

4. Restart the master server.

If you are not using `InnoDB` tables, you can get a snapshot of the system from a master without shutting down the server as described in the following steps:

1. Acquire a read lock and get the master's status. See [Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”](#).
2. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

3. In the client where you acquired the read lock, release the lock:

```
mysql> UNLOCK TABLES;
```

Once you have created the archive or copy of the database, copy the files to each slave before starting the slave replication process.

16.1.1.7. Setting Up Replication with New Master and Slaves

The easiest and most straightforward method for setting up replication is to use new master and slave servers.

You can also use this method if you are setting up new servers but have an existing dump of the databases from a different server that you want to load into your replication configuration. By loading the data into a new master, the data will be automatically replicated to the slaves.

To set up replication between a new master and slave:

1. Configure the MySQL master with the necessary configuration properties. See [Section 16.1.1.1, “Setting the Replication Master Configuration”](#).
2. Start up the MySQL master.
3. Set up a user. See [Section 16.1.1.3, “Creating a User for Replication”](#).
4. Obtain the master status information. See [Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”](#).
5. On the master, release the read lock:

```
mysql> UNLOCK TABLES;
```

6. On the slave, edit the MySQL configuration. See [Section 16.1.1.2, “Setting the Replication Slave Configuration”](#).
7. Start up the MySQL slave.
8. Execute a `CHANGE MASTER TO` statement to set the master replication server configuration. See [Section 16.1.1.10, “Setting the Master Configuration on the Slave”](#).

Perform the slave setup steps on each slave.

Because there is no data to load or exchange on a new server configuration you do not need to copy or import any information.

If you are setting up a new replication environment using the data from a different existing database server, you will now need to run the dump file generated from that server on the new master. The database updates will automatically be propagated to the slaves:

```
shell> mysql -h master < fulldb.dump
```

16.1.1.8. Setting Up Replication with Existing Data

When setting up replication with existing data, you will need to decide how best to get the data from the master to the slave before starting the replication service.

The basic process for setting up replication with existing data is as follows:

1. With the MySQL master running, create a user to be used by the slave when connecting to the master during replication. See [Section 16.1.1.3, “Creating a User for Replication”](#).
2. If you have not already configured the `server-id` [1858] and enabled binary logging on the master server, you will need to shut it down to configure these options. See [Section 16.1.1.1, “Setting the Replication Master Configuration”](#).

If you have to shut down your master server, this is a good opportunity to take a snapshot of its databases. You should obtain the master status (see [Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”](#)) before taking down the master, updating the configuration and taking a snapshot. For information on how to create a snapshot using raw data files, see [Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”](#).

3. If your master server is already correctly configured, obtain its status (see [Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”](#)) and then use `mysqldump` to take a snapshot (see [Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”](#)) or take a raw snapshot of the live server using the guide in [Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”](#).
4. Update the configuration of the slave. See [Section 16.1.1.2, “Setting the Replication Slave Configuration”](#).
5. The next step depends on how you created the snapshot of data on the master.

If you used `mysqldump`:

- a. Start the slave, using the `--skip-slave-start` [1882] option so that replication does not start.
- b. Import the dump file:

```
shell> mysql < fulldb.dump
```

If you created a snapshot using the raw data files:

- a. Extract the data files into your slave data directory. For example:

```
shell> tar xvf dbdump.tar
```

You may need to set permissions and ownership on the files so that the slave server can access and modify them.

- b. Start the slave, using the `--skip-slave-start` [1882] option so that replication does not start.
6. Configure the slave with the replication coordinates from the master. This tells the slave the binary log file and position within the file where replication needs to start. Also, configure the slave with the login credentials and host name of the master. For more information on the `CHANGE MASTER TO` statement required, see [Section 16.1.1.10, “Setting the Master Configuration on the Slave”](#).
 7. Start the slave threads:

```
mysql> START SLAVE;
```

After you have performed this procedure, the slave should connect to the master and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server-id` [1858] option for the master, slaves cannot connect to it.

If you have forgotten to set the `server-id` [1858] option for the slave, you get the following error in the slave's error log:

```
Warning: You should set server-id to a non-0 value if master_host
is set; we will force server id to 2, but this MySQL server will
not act as a slave.
```

You also find error messages in the slave's error log if it is not able to replicate for any other reason.

Once a slave is replicating, you can find in its data directory one file named `master.info` and another named `relay-log.info`. The slave uses these two files to keep track of how much of the master's binary log it has processed. Do *not* remove or edit these files unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement to change replication parameters. The slave will use the values specified in the statement to update the status files automatically.



Note

The content of `master.info` overrides some of the server options specified on the command line or in `my.cnf`. See [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#), for more details.

A single snapshot of the master suffices for multiple slaves. To set up additional slaves, use the same master snapshot and follow the slave portion of the procedure just described.

16.1.1.9. Introducing Additional Slaves to an Existing Replication Environment

To add another slave to an existing replication configuration, you can do so without stopping the master. Instead, set up the new slave by making a copy of an existing slave, except that you configure the new slave with a different `server-id` [1858] value.

To duplicate an existing slave:

1. Shut down the existing slave:

```
shell> mysqladmin shutdown
```

2. Copy the data directory from the existing slave to the new slave. You can do this by creating an archive using `tar` or `WinZip`, or by performing a direct copy using a tool such as `cp` or `rsync`. Ensure that you also copy the log files and relay log files.

A common problem that is encountered when adding new replication slaves is that the new slave fails with a series of warning and error messages like these:

```
071118 16:44:10 [Warning] Neither --relay-log nor --relay-log-index were used; so
replication may break when this MySQL server acts as a slave and has his hostname
changed!! Please use '--relay-log=new_slave_hostname-relay-bin' to avoid this problem.
071118 16:44:10 [ERROR] Failed to open the relay log './old_slave_hostname-relay-bin.003525'
(relay_log_pos 22940879)
071118 16:44:10 [ERROR] Could not find target log during relay log initialization
071118 16:44:10 [ERROR] Failed to initialize the master info structure
```

This is due to the fact that, if the `--relay-log` [1873] option is not specified, the relay log files contain the host name as part of their file names. (This is also true of the relay log index file if the `--relay-log-index` [1874] option is not used. See [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#), for more information about these options.)

To avoid this problem, use the same value for `--relay-log` [1873] on the new slave that was used on the existing slave. (If this option was not set explicitly on the existing slave, use `existing_slave_hostname-relay-bin`.) If this is not feasible, copy the existing slave's relay log index file to the new slave and set the `--relay-log-index` [1874] option on the new slave to match what was used on the existing slave. (If this option was not set explicitly on the existing slave,

use `existing_slave_hostname-relay-bin.index`.) Alternatively—if you have already tried to start the new slave (after following the remaining steps in this section) and have encountered errors like those described previously—then perform the following steps:

- a. If you have not already done so, issue a `STOP SLAVE` on the new slave.
If you have already started the existing slave again, issue a `STOP SLAVE` on the existing slave as well.
 - b. Copy the contents of the existing slave's relay log index file into the new slave's relay log index file, making sure to overwrite any content already in the file.
 - c. Proceed with the remaining steps in this section.
3. Copy the `master.info` and `relay-log.info` files from the existing slave to the new slave if they were not located in the data directory. These files hold the current log coordinates for the master's binary log and the slave's relay log.
 4. Start the existing slave.
 5. On the new slave, edit the configuration and give the new slave a unique `server-id` [1858] not used by the master or any of the existing slaves.
 6. Start the new slave. The slave will use the information in its `master.info` file to start the replication process.

16.1.1.10. Setting the Master Configuration on the Slave

To set up the slave to communicate with the master for replication, you must tell the slave the necessary connection information. To do this, execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
->     MASTER_HOST='master_host_name',
->     MASTER_USER='replication_user_name',
->     MASTER_PASSWORD='replication_password',
->     MASTER_LOG_FILE='recorded_log_file_name',
->     MASTER_LOG_POS=recorded_log_position;
```



Note

Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

The `CHANGE MASTER TO` statement has other options as well. For example, it is possible to set up secure replication using SSL. For a full list of options, and information about the maximum permissible length for the string-valued options, see [Section 13.4.2.1, “CHANGE MASTER TO Syntax”](#).

16.1.2. Replication Formats

Replication works because events written to the binary log are read from the master and then processed on the slave. The events are recorded within the binary log in different formats according to the type of event. The different replication formats used correspond to the binary logging format used when the events were recorded in the master's binary log. The correlation between binary logging formats and the terms used during replication are:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called *statement-based replication* (often abbreviated as *SBR*), which corresponds to the

standard statement-based binary logging format. In older versions of MySQL (5.1.4 and earlier), binary logging and replication used this format exclusively.

- Row-based binary logging logs changes in individual table rows. When used with MySQL replication, this is known as *row-based replication* (often abbreviated as *RBR*). In row-based replication, the master writes *events* to the binary log that indicate how individual table rows are changed.
- The server can change the binary logging format in real time according to the type of event using *mixed-format logging*.

When the mixed format is in effect, statement-based logging is used by default, but automatically switches to row-based logging in particular cases as described later. Replication using the mixed format is often referred to as *mixed-based replication* or *mixed-format replication*. For more information, see [Section 5.2.4.3, “Mixed Binary Logging Format”](#).

In MySQL 5.5, statement-based format is the default.

MySQL Cluster. The default binary logging format in all MySQL Cluster NDB 6.x and 7.x releases is [ROW](#). MySQL Cluster Replication always uses row-based replication, and the [NDBCLUSTER](#) storage engine is incompatible with statement-based replication. Using [NDBCLUSTER](#) sets row-based logging format automatically. See [Section 17.6.2, “General Requirements for MySQL Cluster Replication”](#), for more information.

When using [MIXED](#) format, the binary logging format is determined in part by the storage engine being used and the statement being executed. For more information on mixed-format logging and the rules governing the support of different logging formats, see [Section 5.2.4.3, “Mixed Binary Logging Format”](#).

The logging format in a running MySQL server is controlled by setting the [binlog_format \[1899\]](#) server system variable. This variable can be set with session or global scope. The rules governing when and how the new setting takes effect are the same as for other MySQL server system variables—setting the variable for the current session lasts only until the end of that session, and the change is not visible to other sessions; setting the variable globally requires a restart of the server to take effect. For more information, see [Section 13.7.4, “SET Syntax”](#).

You must have the [SUPER \[697\]](#) privilege to set either the global or session [binlog_format \[1899\]](#) value.

The statement-based and row-based replication formats have different issues and limitations. For a comparison of their relative advantages and disadvantages, see [Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”](#).

With statement-based replication, you may encounter issues with replicating stored routines or triggers. You can avoid these issues by using row-based replication instead. For more information, see [Section 19.7, “Binary Logging of Stored Programs”](#).

16.1.2.1. Advantages and Disadvantages of Statement-Based and Row-Based Replication

Each binary logging format has advantages and disadvantages. For most users, the mixed replication format should provide the best combination of data integrity and performance. If, however, you want to take advantage of the features specific to the statement-based or row-based replication format when performing certain tasks, you can use the information in this section, which provides a summary of their relative advantages and disadvantages, to determine which is best for your needs.

- [Advantages of statement-based replication](#)
- [Disadvantages of statement-based replication](#)
- [Advantages of row-based replication](#)

- [Disadvantages of row-based replication](#)

Advantages of statement-based replication

- Proven technology that has existed in MySQL since 3.23.
- Less data written to log files. When updates or deletes affect many rows, this results in *much* less storage space required for log files. This also means that taking and restoring from backups can be accomplished more quickly.
- Log files contain all statements that made any changes, so they can be used to audit the database.

Disadvantages of statement-based replication

- **Statements that are unsafe for SBR.**
Not all statements which modify data (such as [INSERT](#), [DELETE](#), [UPDATE](#), and [REPLACE](#) statements) can be replicated using statement-based replication. Any nondeterministic behavior is difficult to replicate when using statement-based replication. Examples of such DML (Data Modification Language) statements include the following:
 - A statement that depends on a UDF or stored program that is nondeterministic, since the value returned by such a UDF or stored program or depends on factors other than the parameters supplied to it. (Row-based replication, however, simply replicates the value returned by the UDF or stored program, so its effect on table rows and data is the same on both the master and slave.) See [Section 16.4.1.10, “Replication of Invoked Features”](#), for more information.
 - [DELETE](#) and [UPDATE](#) statements that use a [LIMIT](#) clause without an [ORDER BY](#) are nondeterministic. See [Section 16.4.1.14, “Replication and LIMIT”](#).
 - Statements using any of the following functions cannot be replicated properly using statement-based replication:
 - [LOAD_FILE\(\)](#) [1105]
 - [UUID\(\)](#) [1201], [UUID_SHORT\(\)](#) [1202]
 - [USER\(\)](#) [1197]
 - [FOUND_ROWS\(\)](#) [1193]
 - [SYSDATE\(\)](#) [1146] (unless both the master and the slave are started with the `--sysdate-is-now` [465] option)
 - [GET_LOCK\(\)](#) [1199]
 - [IS_FREE_LOCK\(\)](#) [1200]
 - [IS_USED_LOCK\(\)](#) [1200]
 - [MASTER_POS_WAIT\(\)](#) [1200]
 - [RAND\(\)](#) [1128]
 - [RELEASE_LOCK\(\)](#) [1201]
 - [SLEEP\(\)](#) [1201]
 - [VERSION\(\)](#) [1198]

However, all other functions are replicated correctly using statement-based replication, including `NOW()` [1143] and so forth.

For more information, see [Section 16.4.1.13, “Replication and System Functions”](#).

Statements that cannot be replicated correctly using statement-based replication are logged with a warning like the one shown here:

```
090213 16:58:54 [Warning] Statement is not safe to log in statement format.
```

A similar warning is also issued to the client in such cases. The client can display it using `SHOW WARNINGS`.

- `INSERT ... SELECT` requires a greater number of row-level locks than with row-based replication.
- `UPDATE` statements that require a table scan (because no index is used in the `WHERE` clause) must lock a greater number of rows than with row-based replication.
- For `InnoDB`: An `INSERT` statement that uses `AUTO_INCREMENT` blocks other nonconflicting `INSERT` statements.
- For complex statements, the statement must be evaluated and executed on the slave before the rows are updated or inserted. With row-based replication, the slave only has to modify the affected rows, not execute the full statement.
- If there is an error in evaluation on the slave, particularly when executing complex statements, statement-based replication may slowly increase the margin of error across the affected rows over time. See [Section 16.4.1.26, “Slave Errors During Replication”](#).
- Stored functions execute with the same `NOW()` [1143] value as the calling statement. However, this is not true of stored procedures.
- Deterministic UDFs must be applied on the slaves.
- Table definitions must be (nearly) identical on master and slave. See [Section 16.4.1.8, “Replication with Differing Table Definitions on Master and Slave”](#), for more information.

Advantages of row-based replication

- All changes can be replicated. This is the safest form of replication.

The `mysql` database is not replicated. The `mysql` database is instead seen as a node-specific database. Row-based replication is not supported on tables in this database. Instead, statements that would normally update this information—such as `GRANT`, `REVOKE` and the manipulation of triggers, stored routines (including stored procedures), and views—are all replicated to slaves using statement-based replication.

For statements such as `CREATE TABLE ... SELECT`, a `CREATE` statement is generated from the table definition and replicated using statement-based format, while the row insertions are replicated using row-based format.

- The technology is the same as in most other database management systems; knowledge about other systems transfers to MySQL.
- Fewer row locks are required on the master, which thus achieves higher concurrency, for the following types of statements:
 - `INSERT ... SELECT`

- `INSERT` statements with `AUTO_INCREMENT`
- `UPDATE` or `DELETE` statements with `WHERE` clauses that do not use keys or do not change most of the examined rows.
- Fewer row locks are required on the slave for any `INSERT`, `UPDATE`, or `DELETE` statement.

Disadvantages of row-based replication

- RBR tends to generate more data that must be logged. To replicate a DML statement (such as an `UPDATE` or `DELETE` statement), statement-based replication writes only the statement to the binary log. By contrast, row-based replication writes each changed row to the binary log. If the statement changes many rows, row-based replication may write significantly more data to the binary log; this is true even for statements that are rolled back. This also means that taking and restoring from backup can require more time. In addition, the binary log is locked for a longer time to write the data, which may cause concurrency problems.
- Deterministic UDFs that generate large `BLOB` values take longer to replicate with row-based replication than with statement-based replication. This is because the `BLOB` column value is logged, rather than the statement generating the data.
- You cannot examine the logs to see what statements were executed, nor can you see on the slave what statements were received from the master and executed.

However, you can see what data was changed using `mysqlbinlog` with the options `--base64-output=DECODE-ROWS` [372] and `--verbose` [377].

- For tables using the `MyISAM` storage engine, a stronger lock is required on the slave for `INSERT` statements when applying them as row-based events to the binary log than when applying them as statements. This means that concurrent inserts on `MyISAM` tables are not supported when using row-based replication.

16.1.2.2. Usage of Row-Based Logging and Replication

Major changes in the replication environment and in the behavior of applications can result from using row-based logging (RBL) or row-based replication (RBR) rather than statement-based logging or replication. This section describes a number of issues known to exist when using row-based logging or replication, and discusses some best practices for taking advantage of row-based logging and replication.

For additional information, see [Section 16.1.2, “Replication Formats”](#), and [Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”](#).

For information about issues specific to MySQL Cluster Replication (which depends on row-based replication), see [Section 17.6.3, “Known Issues in MySQL Cluster Replication”](#).

- **RBL, RBR, and temporary tables.** As noted in [Section 16.4.1.21, “Replication and Temporary Tables”](#), temporary tables are not replicated when using row-based format. When mixed format is in effect, “safe” statements involving temporary tables are logged using statement-based format. For more information, see [Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”](#).

Temporary tables are not replicated when using row-based format because there is no need. In addition, because temporary tables can be read only from the thread which created them, there is seldom if ever any benefit obtained from replicating them, even when using statement-based format.

Beginning with MySQL 5.5.5, you can switch from statement-based to row-based binary logging mode even when temporary tables have been created. However, while using the row-based format, the

MySQL server cannot determine the logging mode that was in effect when a given temporary table was created. For this reason, the server in such cases logs a `DROP TEMPORARY TABLE IF EXISTS` statement for each temporary table that still exists for a given client session when that session ends. (Bug #11760229, Bug #11762267) While this means that it is possible that an unnecessary `DROP TEMPORARY TABLE` statement might be logged in some cases, the statement is harmless, and does not cause an error even if the table does not exist, due to the presence of the `IF NOT EXISTS` option.

- **RBL and synchronization of nontransactional tables.** When many rows are affected, the set of changes is split into several events; when the statement commits, all of these events are written to the binary log. When executing on the slave, a table lock is taken on all tables involved, and then the rows are applied in batch mode. (This may or may not be effective, depending on the engine used for the slave's copy of the table.)
- **Latency and binary log size.** Because RBL writes changes for each row to the binary log, its size can increase quite rapidly. In a replication environment, this can significantly increase the time required to make changes on the slave that match those on the master. You should be aware of the potential for this delay in your applications.
- **Reading the binary log.** `mysqlbinlog` displays row-based events in the binary log using the `BINLOG` statement (see [Section 13.7.6.1, “BINLOG Syntax”](#)). This statement displays an event in printable form, but as a base 64-encoded string the meaning of which is not evident. When invoked with the `--base64-output=DECODE-ROWS` [372] and `--verbose` [377] options, `mysqlbinlog` formats the contents of the binary log in a manner that is easily human readable. This is helpful when binary log events were written in row-based format if you want to read or recover from a replication or database failure using the contents of the binary log. For more information, see [Section 4.6.7.2, “mysqlbinlog Row Event Display”](#).
- **Binary log execution errors and `slave_exec_mode`.** If `slave_exec_mode` [1887] is `IDEMPOTENT`, a failure to apply changes from RBL because the original row cannot be found does not trigger an error or cause replication to fail. This means that it is possible that updates are not applied on the slave, so that the master and slave are no longer synchronized. Latency issues and use of nontransactional tables with RBR when `slave_exec_mode` [1887] is `IDEMPOTENT` can cause the master and slave to diverge even further. For more information about `slave_exec_mode` [1887], see [Section 5.1.4, “Server System Variables”](#).



Note

`slave_exec_mode=IDEMPOTENT` [1887] is generally useful only for circular replication or multi-master replication with MySQL Cluster, for which `IDEMPOTENT` is the default value (see [Section 17.6, “MySQL Cluster Replication”](#)).

For other scenarios, setting `slave_exec_mode` [1887] to `STRICT` is normally sufficient; this is the default value for storage engines other than `NDB`.

- **Lack of binary log checksums.** RBL uses no checksums. This means that network, disk, and other errors may not be identified when processing the binary log. To ensure that data is transmitted without network corruption, you may want to consider using SSL, which adds another layer of checksumming, for replication connections. The `CHANGE MASTER TO` statement has options to enable replication over SSL. See also [Section 13.4.2.1, “CHANGE MASTER TO Syntax”](#), for general information about setting up MySQL with SSL.
- **Filtering based on server ID not supported.** A common practice is to filter out changes on some slaves by using a `WHERE` clause that includes the relation `@server_id <> id_value` clause with `UPDATE` and `DELETE` statements, a simple example of such a clause being `WHERE @server_id <>`

1. However, this does not work correctly with row-based logging. If you must use the `server_id` [551] system variable for statement filtering, you must also use `--binlog_format=STATEMENT` [1899].

In MySQL 5.5, you can do filtering based on server ID by using the `IGNORE_SERVER_IDS` option for the `CHANGE MASTER TO` statement. This option works with the statement-based and row-based logging formats.

- **Database-level replication options.** The effects of the `--replicate-do-db` [1875], `--replicate-ignore-db` [1877], and `--replicate-rewrite-db` [1878] options differ considerably depending on whether row-based or statement-based logging is used. Because of this, it is recommended to avoid database-level options and instead use table-level options such as `--replicate-do-table` [1878] and `--replicate-ignore-table` [1878]. For more information about these options and the impact that your choice of replication format has on how they operate, see Section 16.1.3, “Replication and Binary Logging Options and Variables”.
- **RBL, nontransactional tables, and stopped slaves.** When using row-based logging, if the slave server is stopped while a slave thread is updating a nontransactional table, the slave database may reach an inconsistent state. For this reason, it is recommended that you use a transactional storage engine such as `InnoDB` for all tables replicated using the row-based format.

Use of `STOP SLAVE` (or `STOP SLAVE SQL_THREAD` in MySQL 5.5.9 and later) prior to shutting down the slave MySQL server helps prevent such issues from occurring, and is always recommended regardless of the logging format or storage engines employed.

16.1.2.3. Determination of Safe and Unsafe Statements in Binary Logging

When speaking of the “safeness” of a statement in MySQL Replication, we are referring to whether a statement and its effects can be replicated correctly using statement-based format. If this is true of the statement, we refer to the statement as *safe*; otherwise, we refer to it as *unsafe*.

In general, a statement is safe if it deterministic, and unsafe if it is not. However, certain nondeterministic functions are *not* considered unsafe (see [Nondeterministic functions not considered unsafe](#), later in this section). In addition, statements using results from floating-point math functions—which are hardware-dependent—are always considered safe (see [Section 16.4.1.11, “Replication and Floating-Point Values”](#)).

Handling of safe and unsafe statements. A statement is treated differently depending on whether the statement is considered safe, and with respect to the binary logging format (that is, the current value of `binlog_format` [1899]).

- No distinction is made in the treatment of safe and unsafe statements when the binary logging mode is `ROW`.
- If the binary logging format is `MIXED`, statements flagged as unsafe are logged using the row-based format; statements regarded as safe are logged using the statement-based format.
- If the binary logging format is `STATEMENT`, statements flagged as being unsafe generate a warning to this effect. (Safe statements are logged normally.)

Each statement flagged as unsafe generates a warning. Formerly, in cases where a great many such statements were executed on the master, this could lead to very large error log files, sometimes even filling up an entire disk unexpectedly. To guard against this, MySQL 5.5.27 introduced a warning suppression mechanism, which behaves as follows: Whenever the 50 most recent `ER_BINLOG_UNSAFE_STATEMENT` [4019] warnings have been generated more than 50 times in any 50-second period, warning suppression is enabled. When activated, this causes such warnings not to be written to the error log; instead, for each 50 warnings of this type, a note `The last warning was repeated N times in last S seconds` is written to the error log. This continues as long as the 50 most recent such warnings were issued in 50 seconds or less; once the rate has decreased below this

threshold, the warnings are once again logged normally. Warning suppression has no effect on how the safety of statements for statement-based logging is determined, nor on how warnings are sent to the client (MySQL clients still receive one warning for each such statement).

For more information, see [Section 16.1.2, “Replication Formats”](#).

Statements considered unsafe.

Statements having the following characteristics are considered unsafe:

- **Statements containing system functions that may return a different value on slave.**

These functions include `FOUND_ROWS()` [1193], `GET_LOCK()` [1199], `IS_FREE_LOCK()` [1200], `IS_USED_LOCK()` [1200], `LOAD_FILE()` [1105], `MASTER_POS_WAIT()` [1200], `RAND()` [1128], `RELEASE_LOCK()` [1201], `ROW_COUNT()` [1196], `SESSION_USER()` [1197], `SLEEP()` [1201], `SYSDATE()` [1146], `SYSTEM_USER()` [1197], `USER()` [1197], `UUID()` [1201], and `UUID_SHORT()` [1202].

Nondeterministic functions not considered unsafe. Although these functions are not deterministic, they are treated as safe for purposes of logging and replication: `CONNECTION_ID()` [1191], `CURDATE()` [1135], `CURRENT_DATE()` [1135], `CURRENT_TIME()` [1135], `CURRENT_TIMESTAMP()` [1135], `CURTIME()` [1135], `LOCALTIME()` [1142], `LOCALTIMESTAMP()` [1142], `NOW()` [1143], `UNIX_TIMESTAMP()` [1150], `UTC_DATE()` [1150], `UTC_TIME()` [1150], `UTC_TIMESTAMP()` [1151], and `LAST_INSERT_ID()` [1194]

For more information, see [Section 16.4.1.13, “Replication and System Functions”](#).

- **References to system variables.** Most system variables are not replicated correctly using the statement-based format. For exceptions, see [Section 5.2.4.3, “Mixed Binary Logging Format”](#).

See [Section 16.4.1.35, “Replication and Variables”](#).

- **UDFs.** Since we have no control over what a UDF does, we must assume that it is executing unsafe statements.

- **Updates a table having an `AUTO_INCREMENT` column.** This is unsafe because the order in which the rows are updated may differ on the master and the slave.

In addition, an `INSERT` into a table that has a composite primary key containing an `AUTO_INCREMENT` column that is not the first column of this composite key is unsafe.

For more information, see [Section 16.4.1.1, “Replication and `AUTO_INCREMENT`”](#).

- **`INSERT DELAYED` statement.** This statement is considered unsafe because the insertion of the rows may interleave with concurrently executing statements.

- **`INSERT ... ON DUPLICATE KEY UPDATE` statements on tables with multiple primary or unique keys.** When executed against a table that contains more than one primary or unique key, this statement is considered unsafe, being sensitive to the order in which the storage engine checks the keys, which is not deterministic, and on which the choice of rows updated by the MySQL Server depends.

An `INSERT ... ON DUPLICATE KEY UPDATE` statement against a table having more than one unique or primary key is marked as unsafe for statement-based replication beginning with MySQL 5.5.24. (Bug #11765650, Bug #58637)

- **Updates using `LIMIT`.** The order in which rows are retrieved is not specified.

See [Section 16.4.1.14, “Replication and `LIMIT`”](#).

- **Accesses or references log tables.** The contents of the system log table may differ between master and slave.
- **Nontransactional operations after transactional operations.** Within a transaction, allowing any nontransactional reads or writes to execute after any transactional reads or writes is considered unsafe.

For more information, see [Section 16.4.1.31, “Replication and Transactions”](#).

- **Accesses or references self-logging tables.** All reads and writes to self-logging tables are considered unsafe. Within a transaction, any statement following a read or write to self-logging tables is also considered unsafe.
- **LOAD DATA INFILE statements.** Beginning with MySQL 5.5.6, `LOAD DATA INFILE` is considered unsafe, it causes a warning in statement-based mode, and a switch to row-based format when using mixed-format logging. See [Section 16.4.1.15, “Replication and LOAD DATA INFILE”](#).

For additional information, see [Section 16.4.1, “Replication Features and Issues”](#).

16.1.3. Replication and Binary Logging Options and Variables

The next few sections contain information about `mysqld` options and server variables that are used in replication and for controlling the binary log. Options and variables for use on replication masters and replication slaves are covered separately, as are options and variables relating to binary logging. A set of quick-reference tables providing basic information about these options and variables is also included (in the next section following this one).

Of particular importance is the `--server-id [1858]` option.

Command-Line Format	<code>--server-id=#</code>
Option-File Format	<code>server-id</code>
Option Sets Variable	Yes, <code>server_id</code> [551]
Variable Name	<code>server_id</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>0</code>
	Range <code>0 .. 4294967295</code>

This option is common to both master and slave replication servers, and is used in replication to enable master and slave servers to identify themselves uniquely. For additional information, see [Section 16.1.3.2, “Replication Master Options and Variables”](#), and [Section 16.1.3.3, “Replication Slave Options and Variables”](#).

On the master and each slave, you *must* use the `--server-id [1858]` option to establish a unique replication ID in the range from 1 to $2^{32} - 1$. “Unique”, means that each ID must be different from every other ID in use by any other replication master or slave. Example: `server-id=3`.

If you omit `--server-id [1858]`, the default ID is 0, in which case a master refuses connections from all slaves, and a slave refuses to connect to a master. For more information, see [Section 16.1.1.2, “Setting the Replication Slave Configuration”](#).

16.1.3.1. Replication and Binary Logging Option and Variable Reference

The following tables list basic information about the MySQL command-line options and system variables applicable to replication and the binary log.

Table 16.1. Replication Options and Variables: MySQL 5.5

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Com_change_master [597]	N	N	Both
	N	Y	N
Com_show_master_status [597]	N	N	Both
	N	Y	N
Com_show_new_master [597]	N	N	Both
	N	Y	N
Com_show_slave_hosts [597]	N	N	Both
	N	Y	N
Com_show_slave_status [597]	N	N	Both
	N	Y	N
Com_slave_start [597]	N	N	Both
	N	Y	N
Com_slave_stop [597]	N	N	Both
	N	Y	N
Rpl_semi_sync_master_clients [605]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_net_avg_wait_time [605]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_net_wait_time [605]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_net_waits [606]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_no_times [606]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_no_tx [606]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_status [606]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_timefunc_failures [606]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_tx_avg_wait_time [606]	N	N	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
	N	Y	N
Rpl_semi_sync_master_tx_wait_time [606]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_tx_waits [606]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_wait_pos_backtraverse [606]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_wait_sessions [607]	N	N	Global
	N	Y	N
Rpl_semi_sync_master_yes_tx [607]	N	N	Global
	N	Y	N
Rpl_semi_sync_slave_status [607]	N	N	Global
	N	Y	N
Rpl_status [607]	N	N	Global
	N	Y	N
slave_exec_mode [1887]	Y	Y	Global
	Y	N	Y
Slave_open_temp_tables [607]	N	N	Global
	N	Y	N
Slave_retried_transactions [608]	N	N	Global
	N	Y	N
Slave_running [608]	N	N	Global
	N	Y	N
abort-slave-event-count [1870]	Y	N	Global
	Y	N	N
disconnect-slave-event-count [1870]	Y	N	Global
	Y	N	N
init_slave [1885]	Y	Y	Global
	Y	N	Y
log-slave-updates [1870]	Y	Y	Global
	Y	N	N
log_slave_updates [1897]	Y	Y	Global
	Y	N	N
master-connect-retry [1884]	Y	N	Global
	Y	N	N

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
master-host [1884]	Y	N	Global
	Y	N	N
master-info-file [1872]	Y	N	Global
	Y	N	N
master-password [1884]	Y	N	Global
	Y	N	N
master-port [1884]	Y	N	Global
	Y	N	N
master-retry-count [1872]	Y	N	Global
	Y	N	N
master-ssl [1885]	Y	N	Global
	Y	N	N
master-ssl-ca [1885]	Y	N	Global
	Y	N	N
master-ssl-capath [1885]	Y	N	Global
	Y	N	N
master-ssl-cert [1885]	Y	N	Global
	Y	N	N
master-ssl-cipher [1885]	Y	N	Global
	Y	N	N
master-ssl-key [1885]	Y	N	Global
	Y	N	N
master-user [1884]	Y	N	Global
	Y	N	N
relay-log [1873]	Y	Y	Global
	Y	N	N
relay-log-index [1874]	Y	Y	Global
	Y	N	N
relay-log-info-file [1874]	Y	N	Global
	Y	N	N
relay_log_info_file [1886]	Y	Y	Global
	Y	N	N
relay_log_index [1885]	Y	Y	Global
	Y	N	N
relay_log_purge [546]	Y	Y	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
	Y	N	Y
relay_log_recovery [1886]	Y	Y	Global
	Y	N	Y
relay_log_space_limit [546]	Y	Y	Global
	Y	N	N
replicate-do-db [1875]	Y	N	Global
	Y	N	N
replicate-do-table [1878]	Y	N	Global
	Y	N	N
replicate-ignore-db [1877]	Y	N	Global
	Y	N	N
replicate-ignore-table [1878]	Y	N	Global
	Y	N	N
replicate-rewrite-db [1878]	Y	N	Global
	Y	N	N
replicate-same-server-id [1879]	Y	N	Global
	Y	N	N
replicate-wild-do-table [1879]	Y	N	Global
	Y	N	N
replicate-wild-ignore-table [1880]	Y	N	Global
	Y	N	N
report-host [1880]	Y	Y	Global
	Y	N	N
report-password [1880]	Y	Y	Global
	Y	N	N
report-port [1881]	Y	Y	Global
	Y	N	N
report-user [1881]	Y	Y	Global
	Y	N	N
rpl_recovery_rank [1887]	N	Y	Global
	N	N	Y
rpl_semi_sync_master_enabled [548]	N	Y	Global
	N	N	Y
rpl_semi_sync_master_timeout [548]	N	Y	Global
	N	N	Y

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
rpl_semi_sync_master_trace_level [549]	N	Y	Global
	N	N	Y
rpl_semi_sync_master_wait_no_slave [549]	N	Y	Global
	N	N	Y
rpl_semi_sync_slave_enabled [550]	N	Y	Global
	N	N	Y
rpl_semi_sync_slave_trace_level [550]	N	Y	Global
	N	N	Y
show-slave-auth-info [1882]	Y	N	Global
	Y	N	N
skip-slave-start [1882]	Y	N	Global
	Y	N	N
slave-load-tmpdir [1882]	Y	Y	Global
	Y	N	N
slave-skip-errors [1883]	Y	Y	Global
	Y	N	N
slave_compressed_protocol [1887]	Y	Y	Global
	Y	N	Y
slave-max-allowed-packet [1869]	Y	N	Global
	Y	N	N
slave_max_allowed_packet [1869]	N	Y	Global
	N	N	Y
slave_net_timeout [1883]	Y	Y	Global
	Y	N	Y
slave_transaction_retries [1889]	Y	Y	Global
	Y	N	Y
slave_type_conversions [1889]	Y	Y	Global
	Y	N	N
sql_slave_skip_counter [1890]	N	Y	Global
	N	N	Y
sync_master_info [1890]	Y	Y	Global
	Y	N	Y
sync_relay_log [1890]	Y	Y	Global
	Y	N	Y
sync_relay_log_info [1891]	Y	Y	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
	Y	N	Y

Section 16.1.3.2, “Replication Master Options and Variables”, provides more detailed information about options and variables relating to replication master servers. For more information about options and variables relating to replication slaves, see Section 16.1.3.3, “Replication Slave Options and Variables”.

Table 16.2. Binary Logging Options and Variables: MySQL 5.5

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Binlog_cache_disk_use [596]	N	N	Global
	N	Y	N
Binlog_stmt_cache_disk_use [596]	N	N	Global
	N	Y	N
Binlog_cache_use [596]	N	N	Global
	N	Y	N
Binlog_stmt_cache_use [597]	N	N	Global
	N	Y	N
Com_show_binlog_events [597]	N	N	Both
	N	Y	N
Com_show_binlogs [597]	N	N	Both
	N	Y	N
binlog-do-db [1894]	Y	N	Global
	Y	N	N
binlog-ignore-db [1896]	Y	N	Global
	Y	N	N
binlog-row-event-max-size [1891]	Y	N	Global
	Y	N	N
binlog_cache_size [1898]	Y	Y	Global
	Y	N	Y
binlog_stmt_cache_size [1901]	Y	Y	Global
	Y	N	Y
binlog_format [436]	Y	Y	Both
	Y	N	Y
binlog_direct_non_transactional_updates [1898]	Y	Y	Both
	Y	N	Y
log-bin-use-v1-row-events [1893]	Y	Y	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
	Y	N	N
<code>log_bin_use_v1_row_events</code> [1901]	Y	Y	Global
	Y	N	N
<code>max-binlog-dump-events</code> [1897]	Y	N	Global
	Y	N	N
<code>max_binlog_cache_size</code> [1902]	Y	Y	Global
	Y	N	Y
<code>max_binlog_size</code> [1903]	Y	Y	Global
	Y	N	Y
<code>max_binlog_stmt_cache_size</code> [1903]	Y	Y	Global
	Y	N	Y
<code>sporadic-binlog-dump-fail</code> [1897]	Y	N	Global
	Y	N	N

Section 16.1.3.4, “Binary Log Options and Variables”, provides more detailed information about options and variables relating to binary logging. For additional general information about the binary log, see Section 5.2.4, “The Binary Log”.

For a table showing *all* command-line options, system and status variables used with `mysqld`, see Section 5.1.1, “Server Option and Variable Reference”.

16.1.3.2. Replication Master Options and Variables

This section describes the server options and system variables that you can use on replication master servers. You can specify the options either on the [command line](#) or in an [option file](#). You can specify system variable values using `SET`.

On the master and each slave, you must use the `server-id` [1858] option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to $2^{32} - 1$, and each ID must be different from every other ID in use by any other replication master or slave. Example: `server-id=3`.

For options used on the master for controlling binary logging, see Section 16.1.3.4, “Binary Log Options and Variables”.

- `auto_increment_increment` [1865]

Command-Line Format	<code>--auto-increment-increment[=#]</code>
Option-File Format	<code>auto_increment_increment</code>
Option Sets Variable	Yes, <code>auto_increment_increment</code> [1865]
Variable Name	<code>auto_increment_increment</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values

	Type	numeric
	Default	1
	Range	1 .. 65535

`auto_increment_increment` [1865] and `auto_increment_offset` [1868] are intended for use with master-to-master replication, and can be used to control the operation of `AUTO_INCREMENT` columns. Both variables have global and session values, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of `auto_increment_increment` [1865] or `auto_increment_offset` [1868] to a noninteger value gives rise to an error, and the actual value of the variable remains unchanged.



Note

`auto_increment_increment` [1865] is supported for use with `NDB` tables only.

These two variables affect `AUTO_INCREMENT` column behavior as follows:

- `auto_increment_increment` [1865] controls the interval between successive column values. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| auto_increment_increment | 1     |
| auto_increment_offset  | 1     |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc1
  -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.04 sec)

mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| auto_increment_increment | 10    |
| auto_increment_offset  | 1     |
+-----+-----+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col  |
+-----+
| 1    |
| 11   |
| 21   |
| 31   |
+-----+
```

```
4 rows in set (0.00 sec)
```

- `auto_increment_offset` [1868] determines the starting point for the `AUTO_INCREMENT` column value. Consider the following, assuming that these statements are executed during the same session as the example given in the description for `auto_increment_increment` [1865]:

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
-> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc2;
+-----+
| col |
+-----+
| 5 |
| 15 |
| 25 |
| 35 |
+-----+
4 rows in set (0.02 sec)
```

If the value of `auto_increment_offset` [1868] is greater than that of `auto_increment_increment` [1865], the value of `auto_increment_offset` [1868] is ignored.

Should one or both of these variables be changed and then new rows inserted into a table containing an `AUTO_INCREMENT` column, the results may seem counterintuitive because the series of `AUTO_INCREMENT` values is calculated without regard to any values already present in the column, and the next value inserted is the least value in the series that is greater than the maximum existing value in the `AUTO_INCREMENT` column. In other words, the series is calculated like so:

$$\text{auto_increment_offset} + N \times \text{auto_increment_increment}$$

where N is a positive integer value in the series [1, 2, 3, ...]. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1 |
```

```

| 11 |
| 21 |
| 31 |
+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1   |
| 11  |
| 21  |
| 31  |
| 35  |
| 45  |
| 55  |
| 65  |
+-----+
8 rows in set (0.00 sec)

```

The values shown for `auto_increment_increment` [1865] and `auto_increment_offset` [1868] generate the series $5 + N \times 10$, that is, [5, 15, 25, 35, 45, ...]. The greatest value present in the `col` column prior to the `INSERT` is 31, and the next available value in the `AUTO_INCREMENT` series is 35, so the inserted values for `col` begin at that point and the results are as shown for the `SELECT` query.

It is not possible to confine the effects of these two variables to a single table, and thus they do not take the place of the sequences offered by some other database management systems; these variables control the behavior of all `AUTO_INCREMENT` columns in *all* tables on the MySQL server. If the global value of either variable is set, its effects persist until the global value is changed or overridden by setting the session value, or until `mysqld` is restarted. If the local value is set, the new value affects `AUTO_INCREMENT` columns for all tables into which new rows are inserted by the current user for the duration of the session, unless the values are changed during that session.

The default value of `auto_increment_increment` [1865] is 1. See Section 16.4.1.1, “Replication and `AUTO_INCREMENT`”.

- `auto_increment_offset` [1868]

Command-Line Format	<code>--auto-increment-offset [=#]</code>	
Option-File Format	<code>auto_increment_offset</code>	
Option Sets Variable	Yes, <code>auto_increment_offset</code> [1868]	
Variable Name	<code>auto_increment_offset</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>1</code>
	Range	<code>1 .. 65535</code>

This variable has a default value of 1. For particulars, see the description for `auto_increment_increment` [1865].

**Note**

`auto_increment_offset` [1868] is supported for use with NDB tables only.

- `slave_max_allowed_packet` [1869]

Version Introduced	5.5.26	
Variable Name	<code>slave_max_allowed_packet</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1073741824
	Range	1024 .. 1073741824

In MySQL 5.5.26 and later, this variable sets the maximum packet size for the slave SQL and I/O threads, so that large updates using row-based replication do not cause replication to fail because an update exceeded `max_allowed_packet` [519].

This global variable always has a value that is a positive integer multiple of 1024; if you set it to some value that is not, the value is rounded down to the next highest multiple of 1024 for it is stored or used; setting `slave_max_allowed_packet` to 0 causes 1024 to be used. (A truncation warning is issued in all such cases.) The default and maximum value is 1073741824 (1 GB); the minimum is 1024.

`slave_max_allowed_packet` can also be set at startup, using the `--slave-max-allowed-packet` [1869] option.

- `slave-max-allowed-packet=bytes` [1869]

Version Introduced	5.5.26	
Command-Line Format	<code>--slave-max-allowed-packet=#</code>	
Option-File Format	<code>slave-max-allowed-packet</code>	
Option Sets Variable	Yes, <code>slave_max_allowed_packet</code> [1869]	
	Permitted Values	
	Type	numeric
	Default	1073741824
	Range	1024 .. 1073741824

In MySQL 5.5.26 and later, this option sets the maximum packet size in bytes for the slave SQL and I/O threads, so that large updates using row-based replication do not cause replication to fail because an update exceeded `max_allowed_packet` [519]. (Bug #12400221, BUG#60926)

The corresponding server variable `slave_max_allowed_packet` [1869] always has a value that is a positive integer multiple of 1024; if you set it to some value that is not such a multiple, the value is automatically rounded down to the next highest multiple of 1024. (For example, if you start the server with `--slave-max-allowed-packet=10000`, the value used is 9216; setting 0 as the value causes 1024 to be used.) A truncation warning is issued in such cases.

The maximum (and default) value is 1073741824 (1 GB); the minimum is 1024.

16.1.3.3. Replication Slave Options and Variables

This section describes the server options and system variables that apply to slave replication servers. You can specify the options either on the [command line](#) or in an [option file](#). Many of the options can be set while the server is running by using the [CHANGE MASTER TO](#) statement. You can specify system variable values using [SET](#).

Server ID. On the master and each slave, you must use the [server-id \[1858\]](#) option to establish a unique replication ID in the range from 1 to $2^{32} - 1$. “Unique” means that each ID must be different from every other ID in use by any other replication master or slave. Example [my.cnf](#) file:

```
[mysqld]
server-id=3
```

Startup options for replication slaves. The following list describes startup options for controlling replication slave servers. Many of these options can be set while the server is running by using the [CHANGE MASTER TO](#) statement. Others, such as the [--replicate-*](#) options, can be set only when the slave server starts. Replication-related system variables are discussed later in this section.

- [--abort-slave-event-count \[1870\]](#)

Command-Line Format	--abort-slave-event-count=#	
Option-File Format	abort-slave-event-count	
	Permitted Values	
	Type	numeric
	Default	0
	Min Value	0

When this option is set to some positive integer *value* other than 0 (the default) it affects replication behavior as follows: After the slave SQL thread has started, *value* log events are permitted to be executed; after that, the slave SQL thread does not receive any more events, just as if the network connection from the master were cut. The slave thread continues to run, and the output from [SHOW SLAVE STATUS](#) displays [Yes](#) in both the [Slave_IO_Running](#) and the [Slave_SQL_Running](#) columns, but no further events are read from the relay log.

This option is used internally by the MySQL test suite for replication testing and debugging. It is not intended for use in a production setting.

- [--disconnect-slave-event-count \[1870\]](#)

Command-Line Format	--disconnect-slave-event-count=#	
Option-File Format	disconnect-slave-event-count	
	Permitted Values	
	Type	numeric
	Default	0

This option is used internally by the MySQL test suite for replication testing and debugging.

- [--log-slave-updates \[1870\]](#)

Command-Line Format	<code>--log-slave-updates</code>
Option-File Format	<code>log-slave-updates</code>
Option Sets Variable	Yes, <code>log_slave_updates</code> [1897]
Variable Name	<code>log_slave_updates</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>boolean</code>
	Default <code>FALSE</code>

Normally, a slave does not log to its own binary log any updates that are received from a master server. This option tells the slave to log the updates performed by its SQL thread to its own binary log. For this option to have any effect, the slave must also be started with the `--log-bin` [1892] option to enable binary logging. Prior to MySQL 5.5, the server would not start when using the `--log-slave-updates` [1870] option without also starting the server with the `--log-bin` [1892] option, and would fail with an error; in MySQL 5.5, only a warning is generated. (Bug #44663) `--log-slave-updates` [1870] is used when you want to chain replication servers. For example, you might want to set up replication servers using this arrangement:

```
A --> B --> C
```

Here, **A** serves as the master for the slave **B**, and **B** serves as the master for the slave **C**. For this to work, **B** must be both a master *and* a slave. You must start both **A** and **B** with `--log-bin` [1892] to enable binary logging, and **B** with the `--log-slave-updates` [1870] option so that updates received from **A** are logged by **B** to its binary log.

- `--log-slow-slave-statements` [1871]

Command-Line Format	<code>--log-slow-slave-statements</code>
Option-File Format	<code>log-slow-slave-statements</code>
	Permitted Values
	Type <code>boolean</code>
	Default <code>off</code>

When the slow query log is enabled, this option enables logging for queries that have taken more than `long_query_time` [517] seconds to execute on the slave.

- `--log-warnings[=level]` [451]

Command-Line Format	<code>--log-warnings[=#]</code>
	<code>-W [#]</code>
Option-File Format	<code>log-warnings</code>
Option Sets Variable	Yes, <code>log_warnings</code> [516]
Variable Name	<code>log_warnings</code>
Variable Scope	Global, Session
Dynamic Variable	Yes

Disabled by	<code>skip-log-warnings</code>	
	Permitted Values	
Platform Bit Size	<code>64</code>	
Type	<code>numeric</code>	
Default	<code>1</code>	
Range	<code>0 .. 18446744073709547520</code>	

This option causes a server to print more messages to the error log about what it is doing. With respect to replication, the server generates warnings that it succeeded in reconnecting after a network/connection failure, and informs you as to how each slave thread started. This option is enabled by default; to disable it, use `--skip-log-warnings` [451]. If the value is greater than 1, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See [Section C.5.2.11, “Communication Errors and Aborted Connections”](#).

Note that the effects of this option are not limited to replication. It produces warnings across a spectrum of server activities.

- `--master-info-file=file_name` [1872]

Command-Line Format	<code>--master-info-file=file_name</code>	
Option-File Format	<code>master-info-file=file_name</code>	
	Permitted Values	
Type	<code>file name</code>	
Default	<code>master.info</code>	

The name to use for the file in which the slave records information about the master. The default name is `master.info` in the data directory. For information about the format of this file, see [Section 16.2.2.2, “Slave Status Logs”](#).

- `--master-retry-count=count` [1872]

Command-Line Format	<code>--master-retry-count=#</code>	
Option-File Format	<code>master-retry-count</code>	
Deprecated	<code>5.6.1</code>	
	Permitted Values	
Platform Bit Size	<code>32</code>	
Type	<code>numeric</code>	
Default	<code>86400</code>	
Range	<code>0 .. 4294967295</code>	
	Permitted Values	
Platform Bit Size	<code>64</code>	
Type	<code>numeric</code>	
Default	<code>86400</code>	

	Range	0 .. 18446744073709551615
--	--------------	---------------------------

The number of times that the slave tries to connect to the master before giving up. Reconnects are attempted at intervals set by the `MASTER_CONNECT_RETRY` option of the `CHANGE MASTER TO` statement (default 60). Reconnects are triggered when data reads by the slave time out according to the `--slave-net-timeout` [1883] option. The default value is 86400. A value of 0 means “infinite”; the slave attempts to connect forever.

- `--max-relay-log-size=size` [1873]

The size at which the server rotates relay log files automatically. For more information, see [Section 16.2.2, “Replication Relay and Status Logs”](#). The default size is 1GB.

- `--read-only` [1873]

Cause the slave to permit no updates except from slave threads or from users having the `SUPER` [697] privilege. On a slave server, this can be useful to ensure that the slave accepts updates only from its master server and not from clients. This variable does not apply to `TEMPORARY` tables.

- `--relay-log=file_name` [1873]

Command-Line Format	<code>--relay-log=name</code>	
Option-File Format	<code>relay-log</code>	
Option Sets Variable	Yes, <code>relay_log</code> [1885]	
Variable Name	<code>relay-log</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The basename for the relay log. The default basename is `host_name-relay-bin`. The server writes the file in the data directory unless the basename is given with a leading absolute path name to specify a different directory. The server creates relay log files in sequence by adding a numeric suffix to the basename.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default basename is used only if the option is not actually specified*. If you use the `--relay-log` [1873] option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see [Section 4.2.3, “Specifying Program Options”](#).

If you specify this option, the value specified is also used as the basename for the relay log index file. You can override this behavior by specifying a different relay log index file basename using the `--relay-log-index` [1874] option.

Starting with MySQL 5.5.20, when the server reads an entry from the index file, it checks whether the entry contains a relative path. If it does, the relative part of the path is replaced with the absolute path set using the `--relay-log` option. An absolute path remains unchanged; in such a case, the index must be edited manually to enable the new path or paths to be used. Prior to MySQL 5.5.20, manual intervention was required whenever relocating the binary log or relay log files. (Bug #11745230, Bug #12133)

You may find the `--relay-log` [1873] option useful in performing the following tasks:

- Creating relay logs whose names are independent of host names.
- If you need to put the relay logs in some area other than the data directory because your relay logs tend to be very large and you do not want to decrease `max_relay_log_size` [523].
- To increase speed by using load-balancing between disks.
- `--relay-log-index=file_name` [1874]

Command-Line Format	<code>--relay-log-index=name</code>	
Option-File Format	<code>relay-log-index</code>	
Option Sets Variable	Yes, <code>relay_log_index</code> [1885]	
Variable Name	<code>relay-log-index</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>

The name to use for the relay log index file. The default name is `host_name-relay-bin.index` in the data directory, where `host_name` is the name of the slave server.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default basename is used only if the option is not actually specified*. If you use the `--relay-log-index` [1874] option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see [Section 4.2.3, “Specifying Program Options”](#).

If you specify this option, the value specified is also used as the basename for the relay logs. You can override this behavior by specifying a different relay log file basename using the `--relay-log` [1873] option.

- `--relay-log-info-file=file_name` [1874]

Command-Line Format	<code>--relay-log-info-file=file_name</code>	
Option-File Format	<code>relay-log-info-file</code>	
Option Sets Variable	Yes, <code>relay_log_info_file</code> [1886]	
	Permitted Values	
	Type	<code>file name</code>
	Default	<code>relay-log.info</code>

The name to use for the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory. For information about the format of this file, see [Section 16.2.2.2, “Slave Status Logs”](#).

- `--relay-log-purge={0|1}` [1874]

Disable or enable automatic purging of relay logs as soon as they are no longer needed. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = N`.

- `--relay-log-recovery={0|1}` [1875]

Command-Line Format	<code>--relay-log-recovery</code>	
Option-File Format	<code>relay-log-recovery</code>	
Option Sets Variable	Yes, <code>relay_log_recovery</code> [1886]	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Enables automatic relay log recovery immediately following server startup, which means that the replication slave discards all unprocessed relay logs and retrieves them from the replication master. This should be used following a crash on the replication slave to ensure that no possibly corrupted relay logs are processed. The default value is 0 (disabled).

- `--relay-log-space-limit=size` [1875]

This option places an upper limit on the total size in bytes of all relay logs on the slave. A value of 0 means “no limit.” This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs because not doing so would cause a deadlock. You should not set `--relay-log-space-limit` [1875] to less than twice the value of `--max-relay-log-size` [1873] (or `--max-binlog-size` [1903] if `--max-relay-log-size` [1873] is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` [1875] is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to ignore `--relay-log-space-limit` [1875] temporarily.

- `--replicate-do-db=db_name` [1875]

Command-Line Format	<code>--replicate-do-db=name</code>	
Option-File Format	<code>replicate-do-db</code>	
	Permitted Values	
	Type	<code>string</code>

The effects of this option depend on whether statement-based or row-based replication is in use.

Statement-based replication. Tell the slave SQL thread to restrict replication to statements where the default database (that is, the one selected by `USE`) is `db_name`. To specify more than one database, use this option multiple times, once for each database; however, doing so does *not* replicate cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` while a different database (or no database) is selected.



Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

An example of what does not work as you might expect when using statement-based replication: If the slave is started with `--replicate-do-db=sales [1875]` and you issue the following statements on the master, the `UPDATE` statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “check just the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Row-based replication. Tells the slave SQL thread to restrict replication to database `db_name`. Only tables belonging to `db_name` are changed; the current database has no effect on this. Suppose that the slave is started with `--replicate-do-db=sales [1875]` and row-based replication is in effect, and then the following statements are run on the master:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The `february` table in the `sales` database on the slave is changed in accordance with the `UPDATE` statement; this occurs whether or not the `USE` statement was issued. However, issuing the following statements on the master has no effect on the slave when using row-based replication and `--replicate-do-db=sales [1875]`:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the statement `USE prices` were changed to `USE sales`, the `UPDATE` statement's effects would still not be replicated.

Another important difference in how `--replicate-do-db [1875]` is handled in statement-based replication as opposed to row-based replication occurs with regard to statements that refer to multiple databases. Suppose that the slave is started with `--replicate-do-db=db1 [1875]`, and the following statements are executed on the master:

```
USE db1;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

If you are using statement-based replication, then both tables are updated on the slave. However, when using row-based replication, only `table1` is affected on the slave; since `table2` is in a different database, `table2` on the slave is not changed by the `UPDATE`. Now suppose that, instead of the `USE db1` statement, a `USE db4` statement had been used:

```
USE db4;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

In this case, the `UPDATE` statement would have no effect on the slave when using statement-based replication. However, if you are using row-based replication, the `UPDATE` would change `table1` on the slave, but not `table2`—in other words, only tables in the database named by `--replicate-do-db [1875]` are changed, and the choice of default database has no effect on this behavior.

If you need cross-database updates to work, use `--replicate-wild-do-table=db_name.%` [1879] instead. See [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#).



Note

This option affects replication in the same manner that `--binlog-do-db` [1894] affects binary logging, and the effects of the replication format on how `--replicate-do-db` [1875] affects replication behavior are the same as those of the logging format on the behavior of `--binlog-do-db` [1894].

This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-ignore-db=db_name` [1877]

Command-Line Format	<code>--replicate-ignore-db=name</code>	
Option-File Format	<code>replicate-ignore-db</code>	
	Permitted Values	
	Type	<code>string</code>

As with `--replicate-do-db` [1875], the effects of this option depend on whether statement-based or row-based replication is in use.

Statement-based replication. Tells the slave SQL thread not to replicate any statement where the default database (that is, the one selected by `USE`) is `db_name`.

Row-based replication. Tells the slave SQL thread not to update any tables in the database `db_name`. The default database has no effect.

When using statement-based replication, the following example does not work as you might expect. Suppose that the slave is started with `--replicate-ignore-db=sales` [1877] and you issue the following statements on the master:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The `UPDATE` statement *is* replicated in such a case because `--replicate-ignore-db` [1877] applies only to the default database (determined by the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based replication, the `UPDATE` statement's effects are *not* propagated to the slave, and the slave's copy of the `sales.january` table is unchanged; in this instance, `--replicate-ignore-db=sales` [1877] causes *all* changes made to tables in the master's copy of the `sales` database to be ignored by the slave.

To specify more than one database to ignore, use this option multiple times, once for each database. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#).

If you need cross-database updates to work, use `--replicate-wild-ignore-table=db_name.%` [1880] instead. See [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#).

**Note**

This option affects replication in the same manner that `--binlog-ignore-db` [1896] affects binary logging, and the effects of the replication format on how `--replicate-ignore-db` [1877] affects replication behavior are the same as those of the logging format on the behavior of `--binlog-ignore-db` [1896].

This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-do-table=db_name.tbl_name` [1878]

Command-Line Format	<code>--replicate-do-table=name</code>	
Option-File Format	<code>replicate-do-table</code>	
	Permitted Values	
	Type	<code>string</code>

Tells the slave SQL thread to restrict replication to the specified table. To specify more than one table, use this option multiple times, once for each table. This works for both cross-database updates and default database updates, in contrast to `--replicate-do-db` [1875]. See [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#).

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-ignore-table=db_name.tbl_name` [1878]

Command-Line Format	<code>--replicate-ignore-table=name</code>	
Option-File Format	<code>replicate-ignore-table</code>	
	Permitted Values	
	Type	<code>string</code>

Tells the slave SQL thread not to replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-ignore-db` [1877]. See [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#).

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-rewrite-db=from_name->to_name` [1878]

Command-Line Format	<code>--replicate-rewrite-db=old_name->new_name</code>	
Option-File Format	<code>replicate-rewrite-db</code>	
	Permitted Values	
	Type	<code>string</code>

Tells the slave to translate the default database (that is, the one selected by `USE`) to `to_name` if it was `from_name` on the master. Only statements involving tables are affected (not statements such as

`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), and only if `from_name` is the default database on the master. This does not work for cross-database updates. To specify multiple rewrites, use this option multiple times. The server uses the first one with a `from_name` value that matches. The database name translation is done *before* the `--replicate-*` rules are tested.

If you use this option on the command line and the “>” character is special to your command interpreter, quote the option value. For example:

```
shell> mysql --replicate-rewrite-db="olddb->newdb"
```

- `--replicate-same-server-id` [1879]

Command-Line Format	<code>--replicate-same-server-id</code>
Option-File Format	<code>replicate-same-server-id</code>
	Permitted Values
	Type <code>boolean</code>
	Default <code>FALSE</code>

To be used on slave servers. Usually you should use the default setting of 0, to prevent infinite loops caused by circular replication. If set to 1, the slave does not skip events having its own server ID. Normally, this is useful only in rare configurations. Cannot be set to 1 if `--log-slave-updates` [1870] is used. By default, the slave I/O thread does not write binary log events to the relay log if they have the slave's server ID (this optimization helps save disk usage). If you want to use `--replicate-same-server-id` [1879], be sure to start the slave with this option before you make the slave read its own events that you want the slave SQL thread to execute.

- `--replicate-wild-do-table=db_name.tbl_name` [1879]

Command-Line Format	<code>--replicate-wild-do-table=name</code>
Option-File Format	<code>replicate-wild-do-table</code>
	Permitted Values
	Type <code>string</code>

Tells the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the “%” and “_” wildcard characters, which have the same meaning as for the `LIKE` [1111] pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#).

This option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Example: `--replicate-wild-do-table=foo%.bar%` [1879] replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

If the table name pattern is `%`, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.%` [1879], database-level statements are replicated if the database name matches the pattern `foo%`.

To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own%db`, but not replicate

tables from the `my1ownAABCdb` database, you should escape the “_” and “%” characters like this: `--replicate-wild-do-table=my_own\%db` [1879]. If you use the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the `bash` shell, you would need to type `--replicate-wild-do-table=my_own\\%db` [1879].

- `--replicate-wild-ignore-table=db_name.tbl_name` [1880]

Command-Line Format	<code>--replicate-wild-ignore-table=name</code>	
Option-File Format	<code>replicate-wild-ignore-table</code>	
	Permitted Values	
	Type	<code>string</code>

Tells the slave thread not to replicate a statement where any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#).

Example: `--replicate-wild-ignore-table=foo%.bar%` [1880] does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

For information about how matching works, see the description of the `--replicate-wild-do-table` [1879] option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` [1880] as well.

- `--report-host=host_name` [1880]

Command-Line Format	<code>--report-host=host_name</code>	
Option-File Format	<code>report-host</code>	
Option Sets Variable	Yes, <code>report_host</code> [547]	
Variable Name	<code>report-host</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>

The host name or IP address of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master. Note that it is not sufficient for the master to simply read the IP address of the slave from the TCP/IP socket after the slave connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

- `--report-password=password` [1880]

Command-Line Format	<code>--report-password=name</code>	
Option-File Format	<code>report-password</code>	
Option Sets Variable	Yes, <code>report_password</code> [547]	
Variable Name	<code>report-password</code>	
Variable Scope	Global	1880

Dynamic Variable	No	
	Permitted Values	
	Type	string

The account password of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` [1882] option is given.

Although the name of this option might imply otherwise, `--report-password` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the password for the MySQL replication user account.

- `--report-port=slave_port_num` [1881]

Command-Line Format	<code>--report-port=#</code>	
Option-File Format	<code>report-port</code>	
Option Sets Variable	Yes, <code>report_port</code> [547]	
Variable Name	<code>report-port</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values (>= 5.5.23)	
	Type	numeric
	Default	0

The TCP/IP port number for connecting to the slave, to be reported to the master during slave registration. Set this only if the slave is listening on a nondefault port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, do not use this option.

Prior to MySQL 5.5.23, the default value for this option was 3306. In MySQL 5.5.23 and later, the value shown is the port number actually used by the slave (Bug #13333431). This change also affects the default value displayed by `SHOW SLAVE HOSTS`.

- `--report-user=user_name` [1881]

Command-Line Format	<code>--report-user=name</code>	
Option-File Format	<code>report-user</code>	
Option Sets Variable	Yes, <code>report_user</code> [548]	
Variable Name	<code>report-user</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The account user name of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` [1882] option is given.

Although the name of this option might imply otherwise, `--report-user` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the name of the MySQL replication user account.

- `--show-slave-auth-info` [1882]

Command-Line Format	<code>--show-slave-auth-info</code>	
Option-File Format	<code>show-slave-auth-info</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Display slave user names and passwords in the output of `SHOW SLAVE HOSTS` on the master server for slaves started with the `--report-user` [1881] and `--report-password` [1880] options.

- `--skip-slave-start` [1882]

Command-Line Format	<code>--skip-slave-start</code>	
Option-File Format	<code>skip-slave-start</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave_compressed_protocol={0|1}` [1887]

Command-Line Format	<code>--slave_compressed_protocol</code>	
Option-File Format	<code>slave_compressed_protocol</code>	
Option Sets Variable	Yes, <code>slave_compressed_protocol</code> [1887]	
Variable Name	<code>slave_compressed_protocol</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

If this option is set to 1, use compression for the slave/master protocol if both the slave and the master support it. The default is 0 (no compression).

- `--slave-load-tmpdir=file_name` [1882]

Command-Line Format	<code>--slave-load-tmpdir=path</code>	
Option-File Format	<code>slave-load-tmpdir</code>	
Option Sets Variable	Yes, <code>slave_load_tmpdir</code> [1887]	
Variable Name	<code>slave_load_tmpdir</code>	

Variable Scope	Global
Dynamic Variable	No
	Permitted Values
Type	file name
Default	/tmp

The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` [568] system variable. When the slave SQL thread replicates a `LOAD DATA INFILE` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some file system that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `--relay-log` [1873] option to place the relay logs in that file system.

The directory specified by this option should be located in a disk-based file system (not a memory-based file system) because the temporary files used to replicate `LOAD DATA INFILE` must survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process.

- `--slave-net-timeout=seconds` [1883]

Command-Line Format	<code>--slave-net-timeout=#</code>
Option-File Format	<code>slave-net-timeout</code>
Option Sets Variable	Yes, <code>slave_net_timeout</code> [1888]
Variable Name	<code>slave_net_timeout</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	numeric
Default	3600
Min Value	1

The number of seconds to wait for more data from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `MASTER_CONNECT_RETRY` option for the `CHANGE MASTER TO` statement, and the number of reconnection attempts is limited by the `--master-retry-count` [1872] option. The default is 3600 seconds (one hour).

- `--slave-skip-errors=[err_code1,err_code2,...|all|ddl_exist_errors]` [1883]

Command-Line Format	<code>--slave-skip-errors=name</code>
Option-File Format	<code>slave-skip-errors</code>
Option Sets Variable	Yes, <code>slave_skip_errors</code> [1888]
Variable Name	<code>slave_skip_errors</code>
Variable Scope	Global
Dynamic Variable	No

Permitted Values	
Type	string
Default	OFF
Valid Values	[list of error codes] all ddl_exist_errors

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This option tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of synchrony with the master, with you having no idea why this has occurred.

For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`. [Appendix C, Errors, Error Codes, and Common Problems](#), lists server error codes.

You can also (but should not) use the very nonrecommended value of `all` to cause the slave to ignore all error messages and keeps going regardless of what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the slave's data is not anywhere close to what it is on the master. *You have been warned.*

For MySQL Cluster Replication in MySQL Cluster NDB 7.2.6 and later, an additional shorthand value `ddl_exists_errors` is supported for use with the enhanced failover mechanism which is implemented in that and later version of MySQL Cluster NDB 7.2. This value is equivalent to the error code list `1007,1008,4050,1051,1054,1060,1061,1068,1094,1146`. *This value is supported only by the `mysqld` that is included with the MySQL Cluster NDB 7.2 distribution. (Bug #11762277, Bug #54854)* For more information, see [Section 17.6.8, "Implementing Failover with MySQL Cluster Replication"](#).

Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
--slave-skip-errors=ddl_exist_errors
```

Obsolete options. *The following options are removed in MySQL 5.5. If you attempt to start `mysqld` with any of these options in MySQL 5.5, the server aborts with an `unknown variable` error. To set the replication parameters formerly associated with these options, you must use the `CHANGE MASTER TO ...` statement (see [Section 13.4.2.1, "CHANGE MASTER TO Syntax"](#)).*

The options affected are shown in this list:

- `--master-host`
- `--master-user`
- `--master-password`
- `--master-port`
- `--master-connect-retry`

- `--master-ssl`
- `--master-ssl-ca`
- `--master-ssl-capath`
- `--master-ssl-cert`
- `--master-ssl-cipher`
- `--master-ssl-key`

System variables used on replication slaves. The following list describes system variables for controlling replication slave servers. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used with replication slaves are listed earlier in this section.

- `init_slave` [1885]

Command-Line Format	<code>--init-slave=name</code>
Option-File Format	<code>init_slave</code>
Option Sets Variable	Yes, <code>init_slave</code> [1885]
Variable Name	<code>init_slave</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>string</code>

This variable is similar to `init_connect` [505], but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` [505] variable.



Note

The SQL thread sends an acknowledgment to the client before it executes `init_slave` [1885]. Therefore, it is not guaranteed that `init_slave` [1885] has been executed when `START SLAVE` returns. See [Section 13.4.2.5, “START SLAVE Syntax”](#), for more information.

- `relay_log` [1885]

Command-Line Format	<code>--relay-log=name</code>
Option-File Format	<code>relay-log</code>
Option Sets Variable	Yes, <code>relay_log</code> [1885]
Variable Name	<code>relay-log</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>file name</code>

The name of the relay log file.

- `relay_log_index` [1885]

Command-Line Format	<code>--relay-log-index</code>
Option-File Format	<code>relay_log_index</code>
Option Sets Variable	Yes, <code>relay_log_index</code> [1885]
Variable Name	<code>relay_log_index</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>string</code>
	Default <code>*host_name*-relay-bin.index</code>

The name of the relay log index file. The default name is `host_name-relay-bin.index` in the data directory, where `host_name` is the name of the slave server.

- `relay_log_info_file` [1886]

Command-Line Format	<code>--relay-log-info-file=file_name</code>
Option-File Format	<code>relay_log_info_file</code>
Option Sets Variable	Yes, <code>relay_log_info_file</code> [1886]
Variable Name	<code>relay_log_info_file</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>string</code>
	Default <code>relay-log.info</code>

The name of the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory.

- `relay_log_recovery` [1886]

Command-Line Format	<code>--relay-log-recovery</code>
Option-File Format	<code>relay_log_recovery</code>
Option Sets Variable	Yes, <code>relay_log_recovery</code> [1886]
Variable Name	<code>relay_log_recovery</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>boolean</code>
	Default <code>FALSE</code>

Enables automatic relay log recovery immediately following server startup, which means that the replication slave discards all unprocessed relay logs and retrieves them from the replication master. This should be used following a crash on the replication slave to ensure that no possibly corrupted relay logs

are processed. The default value is 0 (disabled). This global variable can be changed dynamically, or by starting the slave with the `--relay-log-recovery` [1875] option.

- `rpl_recovery_rank` [1887]

This variable is unused, and is removed in MySQL 5.6.

- `slave_compressed_protocol` [1887]

Command-Line Format	<code>--slave_compressed_protocol</code>	
Option-File Format	<code>slave_compressed_protocol</code>	
Option Sets Variable	Yes, <code>slave_compressed_protocol</code> [1887]	
Variable Name	<code>slave_compressed_protocol</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Whether to use compression of the slave/master protocol if both the slave and the master support it.

- `slave_exec_mode` [1887]

Command-Line Format	<code>--slave-exec-mode=mode</code>	
Option-File Format	<code>slave_exec_mode</code>	
Option Sets Variable	Yes, <code>slave_exec_mode</code> [1887]	
Variable Name	<code>slave_exec_mode</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>enumeration</code>
	Default	<code>STRICT (ALL)</code>
	Default	<code>IDEMPOTENT (NDB)</code>
	Valid Values	<code>IDEMPOTENT</code> <code>STRICT</code>

Controls whether `IDEMPOTENT` or `STRICT` mode is used in replication conflict resolution and error checking. `IDEMPOTENT` mode causes suppression of duplicate-key and no-key-found errors. This mode should be employed in multi-master replication, circular replication, and some other special replication scenarios. `STRICT` mode is the default, and is suitable for most other cases.



Note

MySQL Cluster ignores any value explicitly set for `slave_exec_mode` [1887], and always treats it as `IDEMPOTENT`.

- `slave_load_tmpdir` [1887]

Command-Line Format	<code>--slave-load-tmpdir=path</code>
Option-File Format	<code>slave-load-tmpdir</code>
Option Sets Variable	Yes, <code>slave_load_tmpdir</code> [1887]
Variable Name	<code>slave_load_tmpdir</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>file name</code>
	Default <code>/tmp</code>

The name of the directory where the slave creates temporary files for replicating `LOAD DATA INFILE` statements.

- `slave_net_timeout` [1888]

Command-Line Format	<code>--slave-net-timeout=#</code>
Option-File Format	<code>slave-net-timeout</code>
Option Sets Variable	Yes, <code>slave_net_timeout</code> [1888]
Variable Name	<code>slave_net_timeout</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>3600</code>
	Min Value <code>1</code>

The number of seconds to wait for more data from a master/slave connection before aborting the read.

- `slave_skip_errors` [1888]

Command-Line Format	<code>--slave-skip-errors=name</code>
Option-File Format	<code>slave-skip-errors</code>
Option Sets Variable	Yes, <code>slave_skip_errors</code> [1888]
Variable Name	<code>slave_skip_errors</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>string</code>
	Default <code>OFF</code>
	Valid Values <code>[list of error codes]</code> <code>all</code>

	<code>ddl_exist_errors</code>
--	-------------------------------

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This variable tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the variable value.

- `slave_transaction_retries` [1889]

Command-Line Format	<code>--slave_transaction_retries=#</code>
Option-File Format	<code>slave_transaction_retries</code>
Option Sets Variable	Yes, <code>slave_transaction_retries</code> [1889]
Variable Name	<code>slave_transaction_retries</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	32
Type	<code>numeric</code>
Default	10
Range	0 .. 4294967295
	Permitted Values
Platform Bit Size	64
Type	<code>numeric</code>
Default	10
Range	0 .. 18446744073709547520

If a replication slave SQL thread fails to execute a transaction because of an InnoDB deadlock or because the transaction's execution time exceeded InnoDB's `innodb_lock_wait_timeout` [1560] or NDBCLUSTER's `TransactionDeadlockDetectionTimeout` [2067] or `TransactionInactiveTimeout` [2066], it automatically retries `slave_transaction_retries` [1889] times before stopping with an error. The default value is 10.

- `slave_type_conversions` [1889]

Version Introduced	5.5.3
Command-Line Format	<code>--slave_type_conversions=set</code>
Option-File Format	<code>slave_type_conversions</code>
Option Sets Variable	Yes, <code>slave_type_conversions</code> [1889]
Variable Name	<code>slave_type_conversions</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
Type	<code>set</code>

	Default	
	Valid Values	ALL_LOSSY ALL_NON_LOSSY ALL_LOSSY,ALL_NON_LOSSY

Controls the type conversion mode in effect on the slave when using row-based replication, including MySQL Cluster Replication. Its value is a comma-delimited set of zero or more elements from the list: [ALL_LOSSY](#), [ALL_NON_LOSSY](#). Set this variable to an empty string to disallow type conversions between the master and the slave. Changes require a restart of the slave to take effect.

For additional information on type conversion modes applicable to attribute promotion and demotion in row-based replication, see [Row-based replication: attribute promotion and demotion](#).

This variable was added in MySQL 5.5.3.

- [sql_slave_skip_counter \[1890\]](#)

Variable Name	sql_slave_skip_counter	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric

The number of events from the master that a slave server should skip.



Important

If skipping the number of events specified by setting this variable would cause the slave to begin in the middle of an event group, the slave continues to skip until it finds the beginning of the next event group and begins from that point. For more information, see [Section 13.4.2.4, “SET GLOBAL sql_slave_skip_counter Syntax”](#).

- [sync_master_info \[1890\]](#)

Command-Line Format	<code>--sync-master-info=#</code>
Option-File Format	<code>sync_master_info</code>
Option Sets Variable	Yes, sync_master_info [1890]
Variable Name	<code>sync_master_info</code>
Variable Scope	Global
Dynamic Variable	Yes

If the value of this variable is greater than 0, a replication slave synchronizes its `master.info` file to disk (using `fdatasync()`) after every [sync_master_info \[1890\]](#) events. The default value of [sync_relay_log_info \[1891\]](#) is 0 (recommended in most situations), which does not force any synchronization to disk by the MySQL server; in this case, the server relies on the operating system to flush the `master.info` file's contents from time to time as for any other file.

- [sync_relay_log \[1890\]](#)

Command-Line Format	<code>--sync-relay-log=#</code>
Option-File Format	<code>sync_relay_log</code>
Option Sets Variable	Yes, <code>sync_relay_log</code> [1890]
Variable Name	<code>sync_relay_log</code>
Variable Scope	Global
Dynamic Variable	Yes

If the value of this variable is greater than 0, the MySQL server synchronizes its relay log to disk (using `fdatasync()`) after every `sync_relay_log` [1890] writes to the relay log. There is one write to the relay log per statement if autocommit is enabled, and one write per transaction otherwise. The default value of `sync_relay_log` [1890] is 0, which does no synchronizing to disk—in this case, the server relies on the operating system to flush the relay log's contents from time to time as for any other file. A value of 1 is the safest choice because in the event of a crash you lose at most one statement or transaction from the relay log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

- `sync_relay_log_info` [1891]

Command-Line Format	<code>--sync-relay-log-info=#</code>
Option-File Format	<code>sync_relay_log_info</code>
Option Sets Variable	Yes, <code>sync_relay_log_info</code> [1891]
Variable Name	<code>sync_relay_log_info</code>
Variable Scope	Global
Dynamic Variable	Yes

If the value of this variable is greater than 0, a replication slave synchronizes its `relay-log.info` file to disk (using `fdatasync()`) after every `sync_relay_log_info` [1891] transactions. A value of 1 is generally the best choice. The default value of `sync_relay_log_info` [1891] is 0, which does not force any synchronization to disk by the MySQL server—in this case, the server relies on the operating system to flush the `relay-log.info` file's contents from time to time as for any other file.

16.1.3.4. Binary Log Options and Variables

You can use the `mysqld` options and system variables that are described in this section to affect the operation of the binary log as well as to control which statements are written to the binary log. For additional information about the binary log, see [Section 5.2.4, “The Binary Log”](#). For additional information about using MySQL server options and system variables, see [Section 5.1.3, “Server Command Options”](#), and [Section 5.1.4, “Server System Variables”](#).

Startup options used with binary logging. The following list describes startup options for enabling and configuring the binary log. System variables used with binary logging are discussed later in this section.

- `--binlog-row-event-max-size=N` [1891]

Command-Line Format	<code>--binlog-row-event-max-size=#</code>
Option-File Format	<code>binlog-row-event-max-size</code>

Specify the maximum size of a row-based binary log event, in bytes. Rows are grouped into events smaller than this size if possible. The value should be a multiple of 256. The default is 1024. See [Section 16.1.2, “Replication Formats”](#).

- `--log-bin[=base_name] [1892]`

Command-Line Format	<code>--log-bin</code>	
Option-File Format	<code>log-bin</code>	
Variable Name	<code>log_bin</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>file name</code>
	Default	<code>OFF</code>

Enable binary logging. The server logs all statements that change data to the binary log, which is used for backup and replication. See [Section 5.2.4, “The Binary Log”](#).

The option value, if given, is the basename for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the basename. It is recommended that you specify a basename (see [Section C.5.8, “Known Issues in MySQL”](#), for the reason). Otherwise, MySQL uses `host_name-bin` as the basename.

In MySQL 5.5.20 and later, when the server reads an entry from the index file, it checks whether the entry contains a relative path, and if it does, the relative part of the path is replaced with the absolute path set using the `--log-bin` option. An absolute path remains unchanged; in such a case, the index must be edited manually to enable the new path or paths to be used. Previous to MySQL 5.5.20, manual intervention was required whenever relocating the binary log or relay log files. (Bug #11745230, Bug #12133)

Setting this option causes the `log_bin` [514] system variable to be set to `ON` (or `1`), and not to the basename. This is a known issue; see Bug #19614 for more information.

- `--log-bin-index[=file_name] [1892]`

Command-Line Format	<code>--log-bin-index=name</code>	
Option-File Format	<code>log-bin-index</code>	
	Permitted Values	
	Type	<code>file name</code>
	Default	<code>OFF</code>

The index file for binary log file names. See [Section 5.2.4, “The Binary Log”](#). If you omit the file name, and if you did not specify one with `--log-bin` [1892], MySQL uses `host_name-bin.index` as the file name.

- `--log-bin-trust-function-creators[={0|1}] [1892]`

Command-Line Format	<code>--log-bin-trust-function-creators</code>
Option-File Format	<code>log-bin-trust-function-creators</code>

Option Sets Variable	Yes, <code>log_bin_trust_function_creators</code> [514]	
Variable Name	<code>log_bin_trust_function_creators</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

This option sets the corresponding `log_bin_trust_function_creators` [514] system variable. If no argument is given, the option sets the variable to 1. `log_bin_trust_function_creators` [514] affects how MySQL enforces restrictions on stored function and trigger creation. See Section 19.7, “Binary Logging of Stored Programs”.

- `--log-bin-use-v1-row-events[={0|1}]` [1893]

Version Introduced	5.5.15-ndb-7.2.1	
Command-Line Format	<code>--log-bin-use-v1-row-events[={0 1}]</code>	
Option-File Format	<code>log-bin-use-v1-row-events</code>	
Option Sets Variable	Yes, <code>log_bin_use_v1_row_events</code> [1901]	
Variable Name	<code>log-bin-use-v1-row-events</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>1</code>
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>1</code>
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>0</code>

Version 2 binary log row events are used by default beginning with MySQL Cluster NDB 7.2.1; however, Version 2 events cannot be read by previous MySQL Cluster releases. Setting `--log-bin-use-v1-row-events` to 1 causes `mysqld` to write the binary log using Version 1 logging events, which is the only version of binary log events used in previous releases, and thus produce binary logs that can be read by older slaves.

The value used for this option can be obtained from the read-only `log_bin_use_v1_row_events` [1901] system variable.

`--log-bin-use-v1-row-events` is chiefly of interest when setting up replication conflict detection and resolution using `NDB$EPOCH_TRANS()` as the conflict detection function, which requires Version 2 binary log row events. Thus, this option and `--ndb-log-transaction-id` [2143] are not compatible.

**Note**

Version 2 binary log row events are also available in MySQL Cluster NDB 7.0.27 and MySQL Cluster NDB 7.1.6 and later MySQL Cluster NDB 7.0 and 7.1 releases. However, prior to MySQL Cluster NDB 7.2.1, Version 1 events are the default (and so the default value for this option is 1 in those versions). You should keep this mind when planning upgrades for setups using MySQL Cluster Replication.

`--log-bin-use-v1-row-events` is not supported in mainline MySQL Server 5.5 releases.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `--log-short-format` [\[450\]](#)

Command-Line Format	<code>--log-short-format</code>	
Option-File Format	<code>log-short-format</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Log less information to the binary log and slow query log, if they have been activated.

Statement selection options. The options in the following list affect which statements are written to the binary log, and thus sent by a replication master server to its slaves. There are also options for slave servers that control which statements received from the master should be executed or ignored. For details, see [Section 16.1.3.3, “Replication Slave Options and Variables”](#).

- `--binlog-do-db=db_name` [\[1894\]](#)

Command-Line Format	<code>--binlog-do-db=name</code>	
Option-File Format	<code>binlog-do-db</code>	
	Permitted Values	
	Type	<code>string</code>

This option affects binary logging in a manner similar to the way that `--replicate-do-db` [\[1875\]](#) affects replication.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--replicate-do-db` [\[1875\]](#) depend on whether statement-based or row-based replication is in use. You should keep in mind that the format used to log a given statement may not necessarily be the same as that indicated by the value of `binlog_format` [\[1899\]](#). For example, DDL statements such as `CREATE TABLE` and `ALTER TABLE` are always logged as statements, without regard to the logging format in effect, so the following statement-based rules for `--binlog-do-db` always apply in determining whether or not the statement is logged.

Statement-based logging. Only those statements are written to the binary log where the default database (that is, the one selected by `USE`) is `db_name`. To specify more than one database, use this option multiple times, once for each database; however, doing so does *not* cause cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` to be logged while a different database (or no database) is selected.



Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

An example of what does not work as you might expect when using statement-based logging: If the server is started with `--binlog-do-db=sales` [1894] and you issue the following statements, the `UPDATE` statement is *not* logged:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Another case which may not be self-evident occurs when a given database is replicated even though it was not specified when setting the option. If the server is started with `--binlog-do-db=sales`, the following `UPDATE` statement is logged even though `prices` was not included when setting `--binlog-do-db`:

```
USE sales;
UPDATE prices.discounts SET percentage = percentage + 10;
```

Because `sales` is the default database when the `UPDATE` statement is issued, the `UPDATE` is logged.

Row-based logging. Logging is restricted to database `db_name`. Only changes to tables belonging to `db_name` are logged; the default database has no effect on this. Suppose that the server is started with `--binlog-do-db=sales` [1894] and row-based logging is in effect, and then the following statements are executed:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The changes to the `february` table in the `sales` database are logged in accordance with the `UPDATE` statement; this occurs whether or not the `USE` statement was issued. However, when using the row-based logging format and `--binlog-do-db=sales` [1894], changes made by the following `UPDATE` are not logged:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the `USE prices` statement were changed to `USE sales`, the `UPDATE` statement's effects would still not be written to the binary log.

Another important difference in `--binlog-do-db` [1894] handling for statement-based logging as opposed to the row-based logging occurs with regard to statements that refer to multiple databases. Suppose that the server is started with `--binlog-do-db=db1` [1894], and the following statements are executed:

```
USE db1;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

If you are using statement-based logging, the updates to both tables are written to the binary log. However, when using the row-based format, only the changes to `table1` are logged; `table2` is in a different database, so it is not changed by the `UPDATE`. Now suppose that, instead of the `USE db1` statement, a `USE db4` statement had been used:

```
USE db4;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

In this case, the `UPDATE` statement is not written to the binary log when using statement-based logging. However, when using row-based logging, the change to `table1` is logged, but not that to `table2`—in other words, only changes to tables in the database named by `--binlog-do-db [1894]` are logged, and the choice of default database has no effect on this behavior.

- `--binlog-ignore-db=db_name [1896]`

Command-Line Format	<code>--binlog-ignore-db=name</code>	
Option-File Format	<code>binlog-ignore-db</code>	
	Permitted Values	
	Type	<code>string</code>

This option affects binary logging in a manner similar to the way that `--replicate-ignore-db [1877]` affects replication.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--replicate-ignore-db [1877]` depend on whether statement-based or row-based replication is in use. You should keep in mind that the format used to log a given statement may not necessarily be the same as that indicated by the value of `binlog_format [1899]`. For example, DDL statements such as `CREATE TABLE` and `ALTER TABLE` are always logged as statements, without regard to the logging format in effect, so the following statement-based rules for `--binlog-ignore-db` always apply in determining whether or not the statement is logged.

Statement-based logging. Tells the server to not log any statement where the default database (that is, the one selected by `USE`) is `db_name`.

Row-based format. Tells the server not to log updates to any tables in the database `db_name`. The current database has no effect.

When using statement-based logging, the following example does not work as you might expect. Suppose that the server is started with `--binlog-ignore-db=sales [1896]` and you issue the following statements:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The `UPDATE` statement *is* logged in such a case because `--binlog-ignore-db [1896]` applies only to the default database (determined by the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based logging, the `UPDATE` statement's effects are *not* written to the binary log, which means that no changes to the `sales.january` table are logged; in this instance, `--binlog-ignore-db=sales [1896]` causes *all* changes made to tables in the master's copy of the `sales` database to be ignored for purposes of binary logging.

To specify more than one database to ignore, use this option multiple times, once for each database. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

You should not use this option if you are using cross-database updates and you do not want these updates to be logged.

Testing and debugging options. The following binary log options are used in replication testing and debugging. They are not intended for use in normal operations.

- `--max-binlog-dump-events=N` [1897]

Command-Line Format	<code>--max-binlog-dump-events=#</code>	
Option-File Format	<code>max-binlog-dump-events</code>	
	Permitted Values	
	Type	numeric
	Default	0

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--sporadic-binlog-dump-fail` [1897]

Command-Line Format	<code>--sporadic-binlog-dump-fail</code>	
Option-File Format	<code>sporadic-binlog-dump-fail</code>	
	Permitted Values	
	Type	boolean
	Default	FALSE

This option is used internally by the MySQL test suite for replication testing and debugging.

System variables used with the binary log. The following list describes system variables for controlling binary logging. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used to control binary logging are listed earlier in this section.

- `log_slave_updates` [1897]

Command-Line Format	<code>--log-slave-updates</code>	
Option-File Format	<code>log_slave_updates</code>	
Option Sets Variable	Yes, <code>log_slave_updates</code> [1897]	
Variable Name	<code>log_slave_updates</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	FALSE

Whether updates received by a slave server from a master server should be logged to the slave's own binary log. Binary logging must be enabled on the slave for this variable to have any effect. See [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).

- [binlog_cache_size \[1898\]](#)

Command-Line Format	<code>--binlog_cache_size=#</code>
Option-File Format	<code>binlog_cache_size</code>
Option Sets Variable	Yes, binlog_cache_size [1898]
Variable Name	<code>binlog_cache_size</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	32
Type	numeric
Default	32768
Range	4096 .. 4294967295
	Permitted Values
Platform Bit Size	64
Type	numeric
Default	32768
Range	4096 .. 18446744073709547520

The size of the cache to hold changes to the binary log during a transaction. A binary log cache is allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (`--log-bin [1892]` option). If you often use large transactions, you can increase this cache size to get better performance. The [Binlog_cache_use \[596\]](#) and [Binlog_cache_disk_use \[596\]](#) status variables can be useful for tuning the size of this variable. See [Section 5.2.4, “The Binary Log”](#).

In MySQL 5.5.3, a separate binary log cache (the binary log statement cache) was introduced for nontransactional statements and in MySQL 5.5.3 through 5.5.8, this variable sets the size for both caches. This means that, in these MySQL versions, the total memory used for these caches is double the value set for `binlog_cache_size`.

Beginning with MySQL 5.5.9, `binlog_cache_size` sets the size for the transaction cache only, and the size of the statement cache is governed by the [binlog_stmt_cache_size \[1901\]](#) system variable.

- [binlog_direct_non_transactional_updates \[1898\]](#)

Version Introduced	5.5.2
Command-Line Format	<code>--binlog_direct_non_transactional_updates[=value]</code>
Option-File Format	<code>binlog_direct_non_transactional_updates</code>
Option Sets Variable	Yes, binlog_direct_non_transactional_updates [1898]

Variable Name	<code>binlog_direct_non_transactional_updates</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Due to concurrency issues, a slave can become inconsistent when a transaction contains updates to both transactional and non-transactional tables. MySQL tries to preserve causality among these statements by writing non-transactional statements to the transaction cache, which is flushed upon commit. However, problems arise when modifications done to nontransactional tables on behalf of a transaction become immediately visible to other connections because these changes may not be written immediately into the binary log.

Beginning with MySQL 5.5.2, the `binlog_direct_non_transactional_updates` [1898] variable offers one possible workaround to this issue. By default, this variable is disabled. Enabling `binlog_direct_non_transactional_updates` [1898] causes updates to nontransactional tables to be written directly to the binary log, rather than to the transaction cache.

`binlog_direct_non_transactional_updates` [1898] works only for statements that are replicated using the statement-based binary logging format; that is, it works only when the value of `binlog_format` [1899] is `STATEMENT`, or when `binlog_format` [1899] is `MIXED` and a given statement is being replicated using the statement-based format. This variable has no effect when the binary log format is `ROW`, or when `binlog_format` [1899] is set to `MIXED` and a given statement is replicated using the row-based format.



Important

Before enabling this variable, you must make certain that there are no dependencies between transactional and nontransactional tables; an example of such a dependency would be the statement `INSERT INTO myisam_table SELECT * FROM innodb_table`. Otherwise, such statements are likely to cause the slave to diverge from the master.

Beginning with MySQL 5.5.5, this variable has no effect when the binary log format is `ROW` or `MIXED`. (Bug #51291)

- `binlog_format` [1899]

Command-Line Format	<code>--binlog-format=format</code>	
Option-File Format	<code>binlog-format=format</code>	
Option Sets Variable	Yes, <code>binlog_format</code> [1899]	
Variable Name	<code>binlog_format</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>enumeration</code>
	Default	<code>STATEMENT</code>

	Valid Values	ROW
		STATEMENT
		MIXED

This variable sets the binary logging format, and can be any one of `STATEMENT`, `ROW`, or `MIXED`. See [Section 16.1.2, “Replication Formats”](#). `binlog_format` [1899] is set by the `--binlog-format` [436] option at startup, or by the `binlog_format` [1899] system variable at runtime.



Note

While you can change the logging format at runtime, it is *not* recommended that you change it while replication is ongoing. This is due in part to the fact that slaves do not honor the master's `binlog_format` [1899] setting; a given MySQL Server can change only its own logging format.

In MySQL 5.5 (including MySQL Cluster NDB 7.2.1 and later), the default format is `STATEMENT`.

You must have the `SUPER` [697] privilege to set either the global or session `binlog_format` [1899] value.

The rules governing when changes to this variable take effect and how long the effect lasts are the same as for other MySQL server system variables. See [Section 13.7.4, “SET Syntax”](#), for more information.

When `MIXED` is specified, statement-based replication is used, except for cases where only row-based replication is guaranteed to lead to proper results. For example, this happens when statements contain user-defined functions (UDF) or the `UUID()` [1201] function. An exception to this rule is that `MIXED` always uses statement-based replication for stored functions and triggers.

There are exceptions when you cannot switch the replication format at runtime:

- From within a stored function or a trigger.
- If the session is currently in row-based replication mode and has open temporary tables.
- Beginning with MySQL 5.5.3, within a transaction. (Bug #47863)

Trying to switch the format in those cases results in an error.



Note

Prior to MySQL Cluster NDB 7.2.1, it was also not possible to change the binary logging format at runtime when the `NDBCLUSTER` storage engine was enabled. In MySQL Cluster NDB 7.2.1 and later, this restriction is removed.

The binary log format affects the behavior of the following server options:

- `--replicate-do-db` [1875]
- `--replicate-ignore-db` [1877]
- `--binlog-do-db` [1894]
- `--binlog-ignore-db` [1896]

These effects are discussed in detail in the descriptions of the individual options.

- [binlog_stmt_cache_size \[1901\]](#)

Version Introduced	5.5.9
Command-Line Format	<code>--binlog_stmt_cache_size=#</code>
Option-File Format	<code>binlog_stmt_cache_size</code>
Option Sets Variable	Yes, binlog_stmt_cache_size [1901]
Variable Name	<code>binlog_stmt_cache_size</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Platform Bit Size 32
	Type numeric
	Default 32768
	Range 4096 .. 4294967295
	Permitted Values
	Platform Bit Size 64
	Type numeric
	Default 32768
	Range 4096 .. 18446744073709547520

Beginning with MySQL 5.5.9, this variable determines the size of the cache for the binary log to hold nontransactional statements issued during a transaction. In MySQL 5.5.3 and later, separate binary log transaction and statement caches are allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (`--log-bin [1892]` option). If you often use large nontransactional statements during transactions, you can increase this cache size to get more performance. The [Binlog_stmt_cache_use \[597\]](#) and [Binlog_stmt_cache_disk_use \[596\]](#) status variables can be useful for tuning the size of this variable. See [Section 5.2.4, “The Binary Log”](#).

In MySQL 5.5.3 through 5.5.8, the size for both caches is set using [binlog_cache_size \[1898\]](#). This means that, in these MySQL versions, the total memory used for these caches is double the value set for [binlog_cache_size \[1898\]](#). Beginning with MySQL 5.5.9, [binlog_cache_size \[1898\]](#) sets the size for the transaction cache only.

- [log_bin_use_v1_row_events \[1901\]](#)

Version Introduced	5.5.15-ndb-7.2.1
Command-Line Format	<code>--log-bin-use-v1-row-events[={0 1}]</code>
Option-File Format	<code>log_bin_use_v1_row_events</code>
Option Sets Variable	Yes, log_bin_use_v1_row_events [1901]
Variable Name	<code>log_bin_use_v1_row_events</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	boolean
	Default	1
	Permitted Values	
	Type	boolean
	Default	1
	Permitted Values	
	Type	boolean
	Default	0

Shows whether Version 2 binary logging, available beginning with MySQL Cluster NDB 7.2.1, is in use. A value of 1 shows that the server is writing the binary log using Version 1 logging events (the only version of binary log events used in previous releases), and thus producing a binary log that can be read by older slaves. 0 indicates that Version 2 binary log events are in use.

This variable is read-only. To switch between Version 1 and Version 2 binary event binary logging, it is necessary to restart `mysqld` with the `--log-bin-use-v1-row-events` [1893] option.

Other than when performing upgrades of MySQL Cluster Replication, `--log-bin-use-v1-events` is chiefly of interest when setting up replication conflict detection and resolution using `NDB $EPOCH_TRANS()`, which requires Version 2 binary row event logging. Thus, this option and `--ndb-log-transaction-id` [2143] are not compatible.



Note

MySQL Cluster NDB 7.2.1 and later use Version 2 binary log row events by default (and so the default value for this variable changes to 0 in those versions). You should keep this mind when planning upgrades for setups using MySQL Cluster Replication.

This variable is not supported in mainline MySQL Server 5.5.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `max_binlog_cache_size` [1902]

Command-Line Format	<code>--max_binlog_cache_size=#</code>	
Option-File Format	<code>max_binlog_cache_size</code>	
Option Sets Variable	Yes, <code>max_binlog_cache_size</code> [1902]	
Variable Name	<code>max_binlog_cache_size</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	18446744073709547520
	Range	4096 .. 18446744073709547520

If a transaction requires more than this many bytes of memory, the server generates a `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` error. The minimum value is 4096. The maximum possible value is 16EB (exabytes). The

maximum recommended value is 4GB; this is due to the fact that MySQL currently cannot work with binary log positions greater than 4GB.



Note

Prior to MySQL 5.5.28, 64-bit Windows platforms truncated the stored value for this variable to 4G, even when it was set to a greater value (Bug #13961678).

In MySQL 5.5.3, a separate binary log cache (the binary log statement cache) was introduced for nontransactional statements and in MySQL 5.5.3 through 5.5.8, this variable sets the upper limit for both caches. This means that, in these MySQL versions, the effective maximum for these caches is double the value set for `max_binlog_cache_size`.

Beginning with MySQL 5.5.9, `max_binlog_cache_size` sets the size for the transaction cache only, and the upper limit for the statement cache is governed by the `max_binlog_stmt_cache_size` [1903] system variable.

Also beginning with MySQL 5.5.9, the session visibility of the `max_binlog_cache_size` [1902] system variable matches that of the `binlog_cache_size` [1898] system variable: In MySQL 5.5.8 and earlier releases, a change in `max_binlog_cache_size` [1902] took immediate effect; in MySQL 5.5.9 and later, a change in `max_binlog_cache_size` [1902] takes effect only for new sessions that started after the value is changed.

- `max_binlog_size` [1903]

Command-Line Format	<code>--max_binlog_size=#</code>	
Option-File Format	<code>max_binlog_size</code>	
Option Sets Variable	Yes, <code>max_binlog_size</code> [1903]	
Variable Name	<code>max_binlog_size</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>1073741824</code>
	Range	<code>4096 .. 1073741824</code>

If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). The minimum value is 4096 bytes. The maximum and default value is 1GB.

A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary log files larger than `max_binlog_size` [1903].

If `max_relay_log_size` [523] is 0, the value of `max_binlog_size` [1903] applies to relay logs as well.

- `max_binlog_stmt_cache_size` [1903]

Version Introduced	5.5.9
Command-Line Format	<code>--max_binlog_stmt_cache_size=#</code>

Option-File Format	<code>max_binlog_stmt_cache_size</code>
Option Sets Variable	Yes, <code>max_binlog_stmt_cache_size</code> [1903]
Variable Name	<code>max_binlog_stmt_cache_size</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	<code>numeric</code>
Default	<code>18446744073709547520</code>
Range	<code>4096 .. 18446744073709547520</code>

If nontransactional statements within a transaction require more than this many bytes of memory, the server generates an error. The minimum value is 4096. The maximum and default values are 4GB on 32-bit platforms and 16EB (exabytes) on 64-bit platforms.



Note

Prior to MySQL 5.5.28, 64-bit Windows platforms truncated the stored value for this variable to 4G, even when it was set to a greater value (Bug #13961678).

In MySQL 5.5.3, a separate binary log cache (the binary log statement cache) was introduced for nontransactional statements and in MySQL 5.5.3 through 5.5.8, this variable sets the upper limit for both caches. This means that, in these MySQL versions, the effective maximum for these caches is double the value set for `max_binlog_cache_size` [1902].

Beginning with MySQL 5.5.9, `max_binlog_stmt_cache_size` sets the size for the statement cache only, and the upper limit for the transaction cache is governed exclusively by the `max_binlog_cache_size` [1902] system variable.

- `sync_binlog` [1904]

Command-Line Format	<code>--sync-binlog=#</code>
Option-File Format	<code>sync_binlog</code>
Option Sets Variable	Yes, <code>sync_binlog</code> [1904]
Variable Name	<code>sync_binlog</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Platform Bit Size	<code>32</code>
Type	<code>numeric</code>
Default	<code>0</code>
Range	<code>0 .. 4294967295</code>
	Permitted Values
Platform Bit Size	<code>64</code>
Type	<code>numeric</code>

Default	0
Range	0 .. 18446744073709547520

If the value of this variable is greater than 0, the MySQL server synchronizes its binary log to disk (using `fdatasync()`) after every `sync_binlog` [1904] writes to the binary log. There is one write to the binary log per statement if autocommit is enabled, and one write per transaction otherwise. The default value of `sync_binlog` [1904] is 0, which does no synchronizing to disk—in this case, the server relies on the operating system to flush the binary log's contents from time to time as for any other file. A value of 1 is the safest choice because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

16.1.4. Common Replication Administration Tasks

Once replication has been started it should execute without requiring much regular administration. Depending on your replication environment, you will want to check the replication status of each slave periodically, daily, or even more frequently.

16.1.4.1. Checking Replication Status

The most common task when managing a replication process is to ensure that replication is taking place and that there have been no errors between the slave and the master. The primary statement for this is `SHOW SLAVE STATUS`, which you must execute on each slave:

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: master1
      Master_User: root
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000004
      Read_Master_Log_Pos: 931
      Relay_Log_File: slavel-relay-bin.000056
      Relay_Log_Pos: 950
      Relay_Master_Log_File: mysql-bin.000004
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
      Replicate_Wild_Ignore_Table:
      Last_Errno: 0
      Last_Error:
      Skip_Counter: 0
      Exec_Master_Log_Pos: 931
      Relay_Log_Space: 1365
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
      Master_SSL_Allowed: No
      Master_SSL_CA_File:
      Master_SSL_CA_Path:
      Master_SSL_Cert:
      Master_SSL_Cipher:
      Master_SSL_Key:
      Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
      Last_IO_Errno: 0
```

```
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:
Replicate_Ignore_Server_Ids: 0
```

The key fields from the status report to examine are:

- `Slave_IO_State`: The current status of the slave. See [Section 8.12.5.6, “Replication Slave I/O Thread States”](#), and [Section 8.12.5.7, “Replication Slave SQL Thread States”](#), for more information.
- `Slave_IO_Running`: Whether the I/O thread for reading the master's binary log is running. Normally, you want this to be `Yes` unless you have not yet started replication or have explicitly stopped it with `STOP SLAVE`.
- `Slave_SQL_Running`: Whether the SQL thread for executing events in the relay log is running. As with the I/O thread, this should normally be `Yes`.
- `Last_IO_Error`, `Last_SQL_Error`: The last errors registered by the I/O and SQL threads when processing the relay log. Ideally these should be blank, indicating no errors.
- `Seconds_Behind_Master`: The number of seconds that the slave SQL thread is behind processing the master binary log. A high number (or an increasing one) can indicate that the slave is unable to handle events from the master in a timely fashion.

A value of 0 for `Seconds_Behind_Master` can usually be interpreted as meaning that the slave has caught up with the master, but there are some cases where this is not strictly true. For example, this can occur if the network connection between master and slave is broken but the slave I/O thread has not yet noticed this—that is, `slave_net_timeout [1888]` has not yet elapsed.

It is also possible that transient values for `Seconds_Behind_Master` may not reflect the situation accurately. When the slave SQL thread has caught up on I/O, `Seconds_Behind_Master` displays 0; but when the slave I/O thread is still queuing up a new event, `Seconds_Behind_Master` may show a large value until the SQL thread finishes executing the new event. This is especially likely when the events have old timestamps; in such cases, if you execute `SHOW SLAVE STATUS` several times in a relatively short period, you may see this value change back and forth repeatedly between 0 and a relatively large value.

Several pairs of fields provide information about the progress of the slave in reading events from the master binary log and processing them in the relay log:

- (`Master_Log_File`, `Read_Master_Log_Pos`): Coordinates in the master binary log indicating how far the slave I/O thread has read events from that log.
- (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`): Coordinates in the master binary log indicating how far the slave SQL thread has executed events received from that log.
- (`Relay_Log_File`, `Relay_Log_Pos`): Coordinates in the slave relay log indicating how far the slave SQL thread has executed the relay log. These correspond to the preceding coordinates, but are expressed in slave relay log coordinates rather than master binary log coordinates.

On the master, you can check the status of connected slaves using `SHOW PROCESSLIST` to examine the list of running processes. Slave connections have `Binlog Dump` in the `Command` field:

```
mysql> SHOW PROCESSLIST \G;
***** 4. row *****
  Id: 10
  User: root
  Host: slave1:58371
```

```

db: NULL
Command: Binlog Dump
Time: 777
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL

```

Because it is the slave that drives the replication process, very little information is available in this report.

For slaves that were started with the `--report-host [1880]` option and are connected to the master, the `SHOW SLAVE HOSTS` statement on the master shows basic information about the slaves. The output includes the ID of the slave server, the value of the `--report-host [1880]` option, the connecting port, and master ID:

```

mysql> SHOW SLAVE HOSTS;
+-----+-----+-----+-----+-----+
| Server_id | Host   | Port | Rpl_recovery_rank | Master_id |
+-----+-----+-----+-----+-----+
|          10 | slave1 | 3306 |          0         |          1 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

16.1.4.2. Pausing Replication on the Slave

You can stop and start the replication of statements on the slave using the `STOP SLAVE` and `START SLAVE` statements.

To stop processing of the binary log from the master, use `STOP SLAVE`:

```
mysql> STOP SLAVE;
```

When replication is stopped, the slave I/O thread stops reading events from the master binary log and writing them to the relay log, and the SQL thread stops reading events from the relay log and executing them. You can pause the I/O or SQL thread individually by specifying the thread type:

```

mysql> STOP SLAVE IO_THREAD;
mysql> STOP SLAVE SQL_THREAD;

```

To start execution again, use the `START SLAVE` statement:

```
mysql> START SLAVE;
```

To start a particular thread, specify the thread type:

```

mysql> START SLAVE IO_THREAD;
mysql> START SLAVE SQL_THREAD;

```

For a slave that performs updates only by processing events from the master, stopping only the SQL thread can be useful if you want to perform a backup or other task. The I/O thread will continue to read events from the master but they are not executed. This makes it easier for the slave to catch up when you restart the SQL thread.

Stopping only the I/O thread enables the events in the relay log to be executed by the SQL thread up to the point where the relay log ends. This can be useful when you want to pause execution to catch up with events already received from the master, when you want to perform administration on the slave but also ensure that it has processed all updates to a specific point. This method can also be used to pause event receipt on the slave while you conduct administration on the master. Stopping the I/O thread but permitting the SQL thread to run helps ensure that there is not a massive backlog of events to be executed when replication is started again.

16.2. Replication Implementation

Replication is based on the master server keeping track of all changes to its databases (updates, deletes, and so on) in its binary log. The binary log serves as a written record of all events that modify database structure or content (data) from the moment the server was started. Typically, `SELECT` statements are not recorded because they modify neither database structure nor content.

Each slave that connects to the master requests a copy of the binary log. That is, it pulls the data from the master, rather than the master pushing the data to the slave. The slave also executes the events from the binary log that it receives. This has the effect of repeating the original changes just as they were made on the master. Tables are created or their structure modified, and data is inserted, deleted, and updated according to the changes that were originally made on the master.

Because each slave is independent, the replaying of the changes from the master's binary log occurs independently on each slave that is connected to the master. In addition, because each slave receives a copy of the binary log only by requesting it from the master, the slave is able to read and update the copy of the database at its own pace and can start and stop the replication process at will without affecting the ability to update to the latest database status on either the master or slave side.

For more information on the specifics of the replication implementation, see [Section 16.2.1, “Replication Implementation Details”](#).

Masters and slaves report their status in respect of the replication process regularly so that you can monitor them. See [Section 8.12.5, “Examining Thread Information”](#), for descriptions of all replicated-related states.

The master binary log is written to a local relay log on the slave before it is processed. The slave also records information about the current position with the master's binary log and the local relay log. See [Section 16.2.2, “Replication Relay and Status Logs”](#).

Database changes are filtered on the slave according to a set of rules that are applied according to the various configuration options and variables that control event evaluation. For details on how these rules are applied, see [Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”](#).

16.2.1. Replication Implementation Details

MySQL replication capabilities are implemented using three threads, one on the master server and two on the slave:

- **Binlog dump thread.** The master creates a thread to send the binary log contents to a slave when the slave connects. This thread can be identified in the output of `SHOW PROCESSLIST` on the master as the `Binlog Dump` thread.

The binlog dump thread acquires a lock on the master's binary log for reading each event that is to be sent to the slave. As soon as the event has been read, the lock is released, even before the event is sent to the slave.

- **Slave I/O thread.** When a `START SLAVE` statement is issued on a slave server, the slave creates an I/O thread, which connects to the master and asks it to send the updates recorded in its binary logs.

The slave I/O thread reads the updates that the master's `Binlog Dump` thread sends (see previous item) and copies them to local files that comprise the slave's relay log.

The state of this thread is shown as `Slave_IO_running` in the output of `SHOW SLAVE STATUS` or as `Slave_running` [608] in the output of `SHOW STATUS`.

- **Slave SQL thread.** The slave creates an SQL thread to read the relay log that is written by the slave I/O thread and execute the events contained therein.

In the preceding description, there are three threads per master/slave connection. A master that has multiple slaves creates one binlog dump thread for each currently connected slave, and each slave has its own I/O and SQL threads.

A slave uses two threads to separate reading updates from the master and executing them into independent tasks. Thus, the task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is stored locally in the slave's relay logs, ready for execution the next time that the slave starts. This enables the master server to purge its binary logs sooner because it no longer needs to wait for the slave to fetch their contents.

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the master and on the slave regarding replication. For information on master states, see [Section 8.12.5.5, “Replication Master Thread States”](#). For slave states, see [Section 8.12.5.6, “Replication Slave I/O Thread States”](#), and [Section 8.12.5.7, “Replication Slave SQL Thread States”](#).

The following example illustrates how the three threads show up in the output from `SHOW PROCESSLIST`.

On the master server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 2
  User: root
  Host: localhost:32931
  db: NULL
Command: Binlog Dump
  Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
        be updated
  Info: NULL
```

Here, thread 2 is a `Binlog Dump` replication thread that services a connected slave. The `State` information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur. If you see no `Binlog Dump` threads on a master server, this means that replication is not running; that is, no slaves are currently connected.

On a slave server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 10
  User: system user
  Host:
  db: NULL
Command: Connect
  Time: 11
  State: Waiting for master to send event
  Info: NULL
***** 2. row *****
  Id: 11
  User: system user
  Host:
  db: NULL
Command: Connect
  Time: 11
  State: Has read all relay log; waiting for the slave I/O
        thread to update it
  Info: NULL
```

The `State` information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. At the time that `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the slave is compared to the master. See [Section B.13, “MySQL 5.5 FAQ: Replication”](#). If sufficient time elapses on the master side without activity on the `Binlog Dump` thread, the master determines that the slave is no longer connected. As for any other client connection, the timeouts for this depend on the values of `net_write_timeout` and `net_retry_count`; for more information about these, see [Section 5.1.4, “Server System Variables”](#).

The `SHOW SLAVE STATUS` statement provides additional information about replication processing on a slave server. See [Section 16.1.4.1, “Checking Replication Status”](#).

16.2.2. Replication Relay and Status Logs

During replication, a slave server creates several logs that hold the binary log events relayed from the master to the slave, and to record information about the current status and location within the relay log. There are three types of logs used in the process, listed here:

- The *relay log* consists of the events read from the binary log of the master and written by the slave I/O thread. Events in the relay log are executed on the slave as part of the SQL thread.
- The *master info log* contains status and current configuration information for the slave's connection to the master. This log holds information on the master host name, login credentials, and coordinates indicating how far the slave has read from the master's binary log.
- The *relay log info log* holds status information about the execution point within the slave's relay log.

16.2.2.1. The Slave Relay Log

The relay log, like the binary log, consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files.

The term “relay log file” generally denotes an individual numbered file containing database events. The term “relay log” collectively denotes the set of numbered relay log files plus the index file.

Relay log files have the same format as binary log files and can be read using `mysqlbinlog` (see [Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”](#)).

By default, relay log file names have the form `host_name-relay-bin.nnnnnn` in the data directory, where `host_name` is the name of the slave server host and `nnnnnn` is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `000001`. The slave uses an index file to track the relay log files currently in use. The default relay log index file name is `host_name-relay-bin.index` in the data directory.

The default relay log file and relay log index file names can be overridden with, respectively, the `--relay-log [1873]` and `--relay-log-index [1874]` server options (see [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#)).

If a slave uses the default host-based relay log file names, changing a slave's host name after replication has been set up can cause replication to fail with the errors `Failed to open the relay log` and `Could not find target log during relay log initialization`. This is a known issue (see Bug #2122). If you anticipate that a slave's host name might change in the future (for example, if networking is set up on the slave such that its host name can be modified using DHCP), you can avoid this issue entirely by using the `--relay-log [1873]` and `--relay-log-index [1874]` options to specify

relay log file names explicitly when you initially set up the slave. This will make the names independent of server host name changes.

If you encounter the issue after replication has already begun, one way to work around it is to stop the slave server, prepend the contents of the old relay log index file to the new one, and then restart the slave. On a Unix system, this can be done as shown here:

```
shell> cat new_relay_log_name.index >> old_relay_log_name.index
shell> mv old_relay_log_name.index new_relay_log_name.index
```

A slave server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed; for example, with `FLUSH LOGS` or `mysqladmin flush-logs`.
- When the size of the current relay log file becomes “too large,” determined as follows:
 - If the value of `max_relay_log_size` [523] is greater than 0, that is the maximum relay log file size.
 - If the value of `max_relay_log_size` [523] is 0, `max_binlog_size` [1903] determines the maximum relay log file size.

The SQL thread automatically deletes each relay log file as soon as it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread takes care of doing so. However, `FLUSH LOGS` rotates relay logs, which influences when the SQL thread deletes them.

16.2.2.2. Slave Status Logs

A replication slave server creates two logs. By default, these logs are files named `master.info` and `relay-log.info` and created in the data directory. The names and locations of these files can be changed by using the `--master-info-file` [1872] and `--relay-log-info-file` [1874] options, respectively. See [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).

The two status logs contain information like that shown in the output of the `SHOW SLAVE STATUS` statement, which is discussed in [Section 13.4.2, “SQL Statements for Controlling Slave Servers”](#). Because the status logs are stored on disk, they survive a slave server’s shutdown. The next time the slave starts up, it reads the two logs to determine how far it has proceeded in reading binary logs from the master and in processing its own relay logs.

The master info log should be protected because it contains the password for connecting to the master. See [Section 6.1.2.3, “Passwords and Logging”](#).

The slave I/O thread updates the master info log. The following table shows the correspondence between the lines in the `master.info` file and the columns displayed by `SHOW SLAVE STATUS`.

Line in <code>master.info</code>	<code>SHOW SLAVE STATUS</code> Column	Description
1		Number of lines in the file
2	<code>Master_Log_File</code>	The name of the master binary log currently being read from the master
3	<code>Read_Master_Log_Pos</code>	The current position within the master binary log that have been read from the master

Line in <code>master.info</code>	<code>SHOW SLAVE STATUS</code> Column	Description
4	<code>Master_Host</code>	The host name of the master
5	<code>Master_User</code>	The user name used to connect to the master
6	Password (not shown by <code>SHOW SLAVE STATUS</code>)	The password used to connect to the master
7	<code>Master_Port</code>	The network port used to connect to the master
8	<code>Connect_Retry</code>	The period (in seconds) that the slave will wait before trying to reconnect to the master
9	<code>Master_SSL_Allowed</code>	Indicates whether the server supports SSL connections
10	<code>Master_SSL_CA_File</code>	The file used for the Certificate Authority (CA) certificate
11	<code>Master_SSL_CA_Path</code>	The path to the Certificate Authority (CA) certificates
12	<code>Master_SSL_Cert</code>	The name of the SSL certificate file
13	<code>Master_SSL_Cipher</code>	The list of possible ciphers used in the handshake for the SSL connection
14	<code>Master_SSL_Key</code>	The name of the SSL key file
15	<code>Master_SSL_Verify_Server_Cert</code>	Whether to verify the server certificate
17	<code>Replicate_Ignore_Server_Ids</code>	The number of server IDs to be ignored, followed by the actual server IDs

The slave SQL thread updates the relay log info log. The following table shows the correspondence between the lines in the `relay-log.info` file and the columns displayed by `SHOW SLAVE STATUS`.

Line in <code>relay-log.info</code>	<code>SHOW SLAVE STATUS</code> Column	Description
1	<code>Relay_Log_File</code>	The name of the current relay log file
2	<code>Relay_Log_Pos</code>	The current position within the relay log file; events up to this position have been executed on the slave database
3	<code>Relay_Master_Log_File</code>	The name of the master binary log file from which the events in the relay log file were read
4	<code>Exec_Master_Log_Pos</code>	The equivalent position within the master's binary log file of events that have already been executed

The contents of the `relay-log.info` file and the states shown by the `SHOW SLAVE STATUS` statement might not match if the `relay-log.info` file has not been flushed to disk. Ideally, you should only view `relay-log.info` on a slave that is offline (that is, `mysqld` is not running). For a running system, `SHOW SLAVE STATUS` should be used.

When you back up the slave's data, you should back up these two status logs, along with the relay log files. The status logs are needed to resume replication after you restore the data from the slave. If you lose the relay logs but still have the relay log info log, you can check it to determine how far the SQL thread has

executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. Of course, this requires that the binary logs still exist on the master.

16.2.3. How Servers Evaluate Replication Filtering Rules

If a master server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all slaves and each slave determines whether to execute it or ignore it.

On the master, you can control which databases to log changes for by using the `--binlog-do-db` [1894] and `--binlog-ignore-db` [1896] options to control binary logging. For a description of the rules that servers use in evaluating these options, see [Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”](#). You should not use these options to control which databases and tables are replicated. Instead, use filtering on the slave to control the events that are executed on the slave.

On the slave side, decisions about whether to execute or ignore statements received from the master are made according to the `--replicate-*` options that the slave was started with. (See [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).)

In the simplest case, when there are no `--replicate-*` options, the slave executes all statements that it receives from the master. Otherwise, the result depends on the particular options given.

Database-level options (`--replicate-do-db` [1875], `--replicate-ignore-db` [1877]) are checked first; see [Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”](#), for a description of this process. If no matching database-level options are found, option checking proceeds to any table-level options that may be in use, as discussed in [Section 16.2.3.2, “Evaluation of Table-Level Replication Options”](#).

To make it easier to determine what effect an option set will have, it is recommended that you avoid mixing “do” and “ignore” options, or wildcard and nonwildcard options. An example of the latter that may have unintended effects is the use of `--replicate-do-db` [1875] and `--replicate-wild-do-table` [1879] together, where `--replicate-wild-do-table` [1879] uses a pattern for the database name that matches the name given for `--replicate-do-db` [1875]. Suppose a replication slave is started with `--replicate-do-db=dbx` [1875] `--replicate-wild-do-table=db%.t1` [1879]. Then, suppose that on the master, you issue the statement `CREATE DATABASE dbx`. Although you might expect it, this statement is not replicated because it does not reference a table named `t1`.

If any `--replicate-rewrite-db` [1878] options were specified, they are applied before the `--replicate-*` filtering rules are tested.



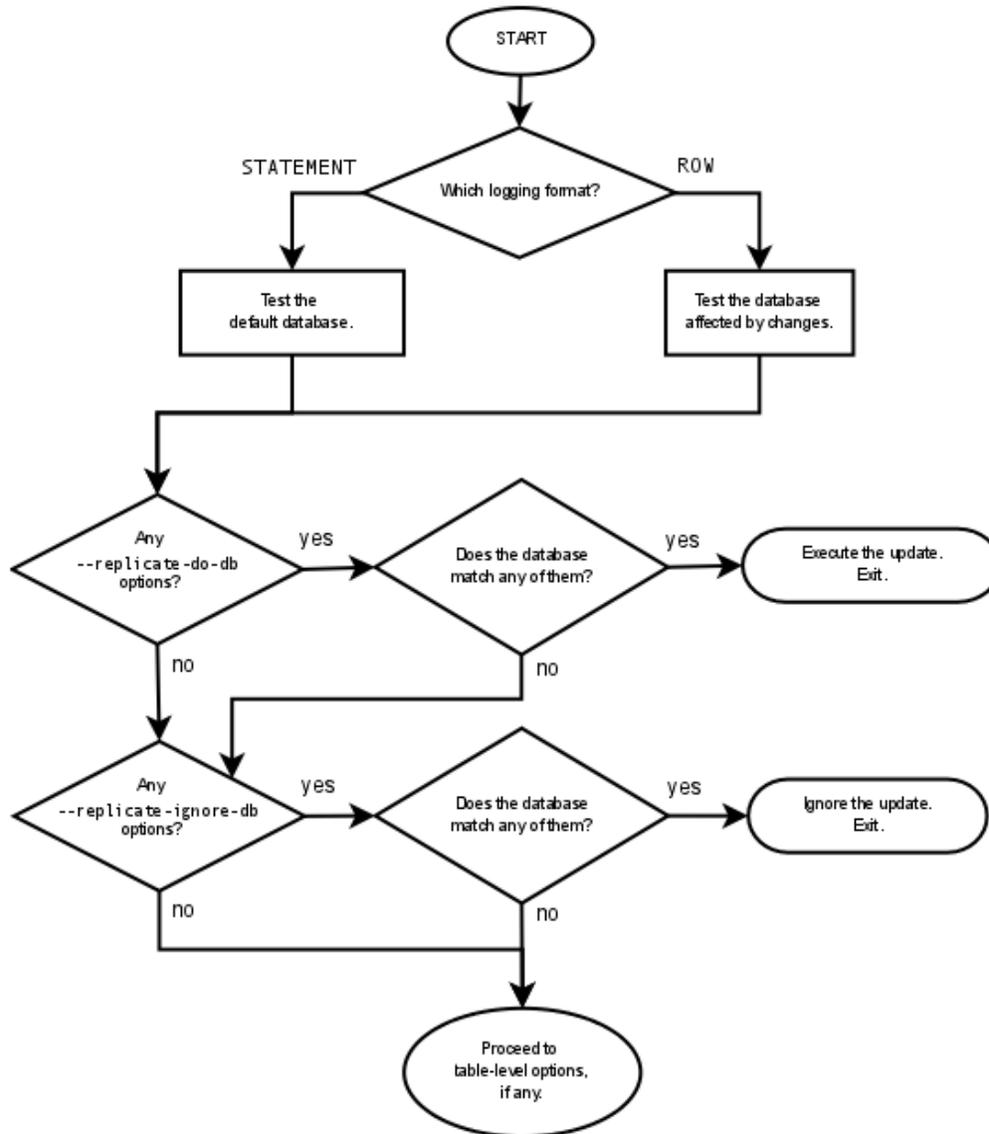
Note

In MySQL 5.5, database-level filtering options are case-sensitive on platforms supporting case sensitivity in filenames, whereas table-level filtering options are not (regardless of platform). This is true regardless of the value of the `lower_case_table_names` [518] system variable.

16.2.3.1. Evaluation of Database-Level Replication and Binary Logging Options

When evaluating replication options, the slave begins by checking to see whether there are any `--replicate-do-db` [1875] or `--replicate-ignore-db` [1877] options that apply. When using `--binlog-do-db` [1894] or `--binlog-ignore-db` [1896], the process is similar, but the options are checked on the master.

With statement-based replication, the default database is checked for a match. With row-based replication, the database where data is to be changed is the database that is checked. Regardless of the binary logging format, checking of database-level options proceeds as shown in the following diagram.



The steps involved are listed here:

1. Are there any `--replicate-do-db [1875]` options?

- **Yes.** Do any of them match the database?
 - **Yes.** Execute the statement and exit.
 - **No.** Continue to step 2.
- **No.** Continue to step 2.

2. Are there any `--replicate-ignore-db [1877]` options?

- **Yes.** Do any of them match the database?

- **Yes.** Ignore the statement and exit.
 - **No.** Continue to step 3.
 - **No.** Continue to step 3.
3. Proceed to checking the table-level replication options, if there are any. For a description of how these options are checked, see [Section 16.2.3.2, “Evaluation of Table-Level Replication Options”](#).



Important

A statement that is still permitted at this stage is not yet actually executed. The statement is not executed until all table-level options (if any) have also been checked, and the outcome of that process permits execution of the statement.

For binary logging, the steps involved are listed here:

1. Are there any `--binlog-do-db [1894]` or `--binlog-ignore-db [1896]` options?
 - **Yes.** Continue to step 2.
 - **No.** Log the statement and exit.
2. Is there a default database (has any database been selected by `USE`)?
 - **Yes.** Continue to step 3.
 - **No.** Ignore the statement and exit.
3. There is a default database. Are there any `--binlog-do-db [1894]` options?
 - **Yes.** Do any of them match the database?
 - **Yes.** Log the statement and exit.
 - **No.** Ignore the statement and exit.
 - **No.** Continue to step 4.
4. Do any of the `--binlog-ignore-db [1896]` options match the database?
 - **Yes.** Ignore the statement and exit.
 - **No.** Log the statement and exit.



Important

For statement-based logging, an exception is made in the rules just given for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. In those cases, the database being *created*, *altered*, or *dropped* replaces the default database when determining whether to log or ignore updates.

`--binlog-do-db [1894]` can sometimes mean “ignore other databases”. For example, when using statement-based logging, a server running with only `--binlog-do-db=sales [1894]` does not write to the binary log statements for which the default database differs from `sales`. When using row-based logging with the same option, the server logs only those updates that change data in `sales`.

16.2.3.2. Evaluation of Table-Level Replication Options

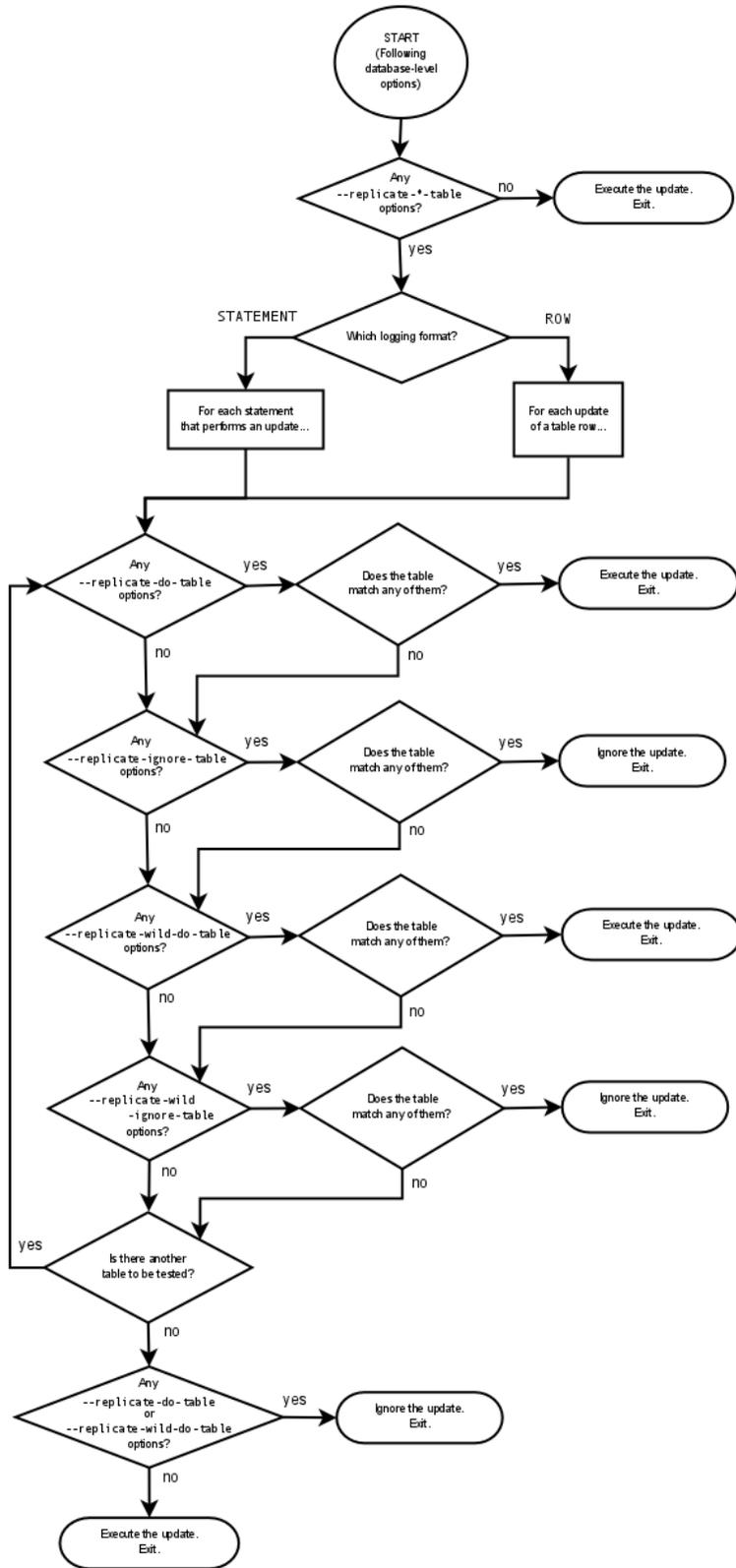
The slave checks for and evaluates table options only if no matching database options were found (see [Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”](#)).

First, as a preliminary condition, the slave checks whether statement-based replication is enabled. If so, and the statement occurs within a stored function, the slave executes the statement and exits. If row-based replication is enabled, the slave does not know whether a statement occurred within a stored function on the master, so this condition does not apply.

**Note**

For statement-based replication, replication events represent statements (all changes making up a given event are associated with a single SQL statement); for row-based replication, each event represents a change in a single table row (thus a single statement such as `UPDATE mytable SET mycol = 1` may yield many row-based events). When viewed in terms of events, the process of checking table options is the same for both row-based and statement-based replication.

Having reached this point, if there are no table options, the slave simply executes all events. If there are any `--replicate-do-table` [\[1878\]](#) or `--replicate-wild-do-table` [\[1879\]](#) options, the event must match one of these if it is to be executed; otherwise, it is ignored. If there are any `--replicate-ignore-table` [\[1878\]](#) or `--replicate-wild-ignore-table` [\[1880\]](#) options, all events are executed except those that match any of these options. This process is illustrated in the following diagram.



The following steps describe this evaluation in more detail:

1. Are there any table options?
 - **Yes.** Continue to step 2.
 - **No.** Execute the event and exit.
2. Are there any `--replicate-do-table` [1878] options?
 - **Yes.** Does the table match any of them?
 - **Yes.** Execute the event and exit.
 - **No.** Continue to step 3.
 - **No.** Continue to step 3.
3. Are there any `--replicate-ignore-table` [1878] options?
 - **Yes.** Does the table match any of them?
 - **Yes.** Ignore the event and exit.
 - **No.** Continue to step 4.
 - **No.** Continue to step 4.
4. Are there any `--replicate-wild-do-table` [1879] options?
 - **Yes.** Does the table match any of them?
 - **Yes.** Execute the event and exit.
 - **No.** Continue to step 5.
 - **No.** Continue to step 5.
5. Are there any `--replicate-wild-ignore-table` [1880] options?
 - **Yes.** Does the table match any of them?
 - **Yes.** Ignore the event and exit.
 - **No.** Continue to step 6.
 - **No.** Continue to step 6.
6. Are there any `--replicate-do-table` [1878] or `--replicate-wild-do-table` [1879] options?
 - **Yes.** Ignore the event and exit.
 - **No.** Execute the event and exit.

16.2.3.3. Replication Rule Application

This section provides additional explanation and examples of usage for different combinations of replication filtering options.

Some typical combinations of replication filter rule types are given in the following table:

Condition (Types of Options)	Outcome
No <code>--replicate-*</code> options at all:	The slave executes all events that it receives from the master.
<code>--replicate-*-db</code> options, but no table options:	The slave accepts or ignores events using the database options. It executes all events permitted by those options because there are no table restrictions.
<code>--replicate-*-table</code> options, but no database options:	All events are accepted at the database-checking stage because there are no database conditions. The slave executes or ignores events based solely on the table options.
A combination of database and table options:	The slave accepts or ignores events using the database options. Then it evaluates all events permitted by those options according to the table options. This can sometimes lead to results that seem counterintuitive, and that may be different depending on whether you are using statement-based or row-based replication; see the text for an example.

A more complex example follows, in which we examine the outcomes for both statement-based and row-based settings.

Suppose that we have two tables `mytbl1` in database `db1` and `mytbl2` in database `db2` on the master, and the slave is running with the following options (and no other replication filtering options):

```
replicate-ignore-db = db1
replicate-do-table  = db2.tbl2
```

Now we execute the following statements on the master:

```
USE db1;
INSERT INTO db2.tbl2 VALUES (1);
```

The results on the slave vary considerably depending on the binary log format, and may not match initial expectations in either case.

Statement-based replication. The `USE` statement causes `db1` to be the default database. Thus the `--replicate-ignore-db [1877]` option matches, and the `INSERT` statement is ignored. The table options are not checked.

Row-based replication. The default database has no effect on how the slave reads database options when using row-based replication. Thus, the `USE` statement makes no difference in how the `--replicate-ignore-db [1877]` option is handled: the database specified by this option does not match the database where the `INSERT` statement changes data, so the slave proceeds to check the table options. The table specified by `--replicate-do-table [1878]` matches the table to be updated, and the row is inserted.

16.3. Replication Solutions

Replication can be used in many different environments for a range of purposes. This section provides general notes and advice on using replication for specific solution types.

For information on using replication in a backup environment, including notes on the setup, backup procedure, and files to back up, see [Section 16.3.1, “Using Replication for Backups”](#).

For advice and tips on using different storage engines on the master and slaves, see [Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”](#).

Using replication as a scale-out solution requires some changes in the logic and operation of applications that use the solution. See [Section 16.3.3, “Using Replication for Scale-Out”](#).

For performance or data distribution reasons, you may want to replicate different databases to different replication slaves. See [Section 16.3.4, “Replicating Different Databases to Different Slaves”](#)

As the number of replication slaves increases, the load on the master can increase and lead to reduced performance (because of the need to replicate the binary log to each slave). For tips on improving your replication performance, including using a single secondary server as an replication master, see [Section 16.3.5, “Improving Replication Performance”](#).

For guidance on switching masters, or converting slaves into masters as part of an emergency failover solution, see [Section 16.3.6, “Switching Masters During Failover”](#).

To secure your replication communication, you can use SSL to encrypt the communication channel. For step-by-step instructions, see [Section 16.3.7, “Setting Up Replication Using SSL”](#).

16.3.1. Using Replication for Backups

To use replication as a backup solution, replicate data from the master to a slave, and then back up the data slave. The slave can be paused and shut down without affecting the running operation of the master, so you can produce an effective snapshot of “live” data that would otherwise require the master to be shut down.

How you back up a database depends on its size and whether you are backing up only the data, or the data and the replication slave state so that you can rebuild the slave in the event of failure. There are therefore two choices:

- If you are using replication as a solution to enable you to back up the data on the master, and the size of your database is not too large, the `mysqldump` tool may be suitable. See [Section 16.3.1.1, “Backing Up a Slave Using `mysqldump`”](#).
- For larger databases, where `mysqldump` would be impractical or inefficient, you can back up the raw data files instead. Using the raw data files option also means that you can back up the binary and relay logs that will enable you to recreate the slave in the event of a slave failure. For more information, see [Section 16.3.1.2, “Backing Up Raw Data from a Slave”](#).

Another backup strategy, which can be used for either master or slave servers, is to put the server in a read-only state. The backup is performed against the read-only server, which then is changed back to its usual read/write operational status. See [Section 16.3.1.3, “Backing Up a Master or Slave by Making It Read Only”](#).

16.3.1.1. Backing Up a Slave Using `mysqldump`

Using `mysqldump` to create a copy of a database enables you to capture all of the data in the database in a format that enables the information to be imported into another instance of MySQL Server (see [Section 4.5.4, “`mysqldump` — A Database Backup Program”](#)). Because the format of the information is SQL statements, the file can easily be distributed and applied to running servers in the event that you need access to the data in an emergency. However, if the size of your data set is very large, `mysqldump` may be impractical.

When using `mysqldump`, you should stop replication on the slave before starting the dump process to ensure that the dump contains a consistent set of data:

1. Stop the slave from processing requests. You can stop replication completely on the slave using `mysqladmin`:

```
shell> mysqladmin stop-slave
```

Alternatively, you can stop only the slave SQL thread to pause event execution:

```
shell> mysql -e 'STOP SLAVE SQL_THREAD;'
```

This enables the slave to continue to receive data change events from the master's binary log and store them in the relay logs using the I/O thread, but prevents the slave from executing these events and changing its data. Within busy replication environments, permitting the I/O thread to run during backup may speed up the catch-up process when you restart the slave SQL thread.

2. Run `mysqldump` to dump your databases. You may either dump all databases or select databases to be dumped. For example, to dump all databases:

```
shell> mysqldump --all-databases > fulldb.dump
```

3. Once the dump has completed, start slave operations again:

```
shell> mysqladmin start-slave
```

In the preceding example, you may want to add login credentials (user name, password) to the commands, and bundle the process up into a script that you can run automatically each day.

If you use this approach, make sure you monitor the slave replication process to ensure that the time taken to run the backup does not affect the slave's ability to keep up with events from the master. See [Section 16.1.4.1, "Checking Replication Status"](#). If the slave is unable to keep up, you may want to add another slave and distribute the backup process. For an example of how to configure this scenario, see [Section 16.3.4, "Replicating Different Databases to Different Slaves"](#).

16.3.1.2. Backing Up Raw Data from a Slave

To guarantee the integrity of the files that are copied, backing up the raw data files on your MySQL replication slave should take place while your slave server is shut down. If the MySQL server is still running, background tasks may still be updating the database files, particularly those involving storage engines with background processes such as [InnoDB](#). With [InnoDB](#), these problems should be resolved during crash recovery, but since the slave server can be shut down during the backup process without affecting the execution of the master it makes sense to take advantage of this capability.

To shut down the server and back up the files:

1. Shut down the slave MySQL server:

```
shell> mysqladmin shutdown
```

2. Copy the data files. You can use any suitable copying or archive utility, including `cp`, `tar` or `WinZip`. For example, assuming that the data directory is located under the current directory, you can archive the entire directory as follows:

```
shell> tar cf /tmp/dbbackup.tar ./data
```

3. Start the MySQL server again. Under Unix:

```
shell> mysqld_safe &
```

Under Windows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld"
```

Normally you should back up the entire data directory for the slave MySQL server. If you want to be able to restore the data and operate as a slave (for example, in the event of failure of the slave), then in addition to the slave's data, you should also back up the slave status files, `master.info` and `relay-log.info`,

along with the relay log files. These files are needed to resume replication after you restore the slave's data.

If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. This requires that the binary logs still exist on the master server.

If your slave is replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations. The location of this directory is the value of the `--slave-load-tmpdir` [1882] option. If the server was not started with that option, the directory location is the value of the `tmpdir` [568] system variable.

16.3.1.3. Backing Up a Master or Slave by Making It Read Only

It is possible to back up either master or slave servers in a replication setup by acquiring a global read lock and manipulating the `read_only` [544] system variable to change the read-only state of the server to be backed up:

1. Make the server read-only, so that it processes only retrievals and blocks updates.
2. Perform the backup.
3. Change the server back to its normal read/write state.



Note

The instructions in this section place the server to be backed up in a state that is safe for backup methods that get the data from the server, such as `mysqldump` (see Section 4.5.4, “`mysqldump` — A Database Backup Program”). You should not attempt to use these instructions to make a binary backup by copying files directly because the server may still have modified data cached in memory and not flushed to disk.

The following instructions describe how to do this for a master server and for a slave server. For both scenarios discussed here, suppose that you have the following replication setup:

- A master server M1
- A slave server S1 that has M1 as its master
- A client C1 connected to M1
- A client C2 connected to S1

In either scenario, the statements to acquire the global read lock and manipulate the `read_only` [544] variable are performed on the server to be backed up and do not propagate to any slaves of that server.

Scenario 1: Backup with a Read-Only Master

Put the master M1 in a read-only state by executing these statements on it:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

While M1 is in a read-only state, the following properties are true:

- Requests for updates sent by C1 to M1 will block because the server is in read-only mode.

- Requests for query results sent by C1 to M1 will succeed.
- Making a backup on M1 is safe.
- Making a backup on S1 is not safe. This server is still running, and might be processing the binary log or update requests coming from client C2

While M1 is read only, perform the backup. For example, you can use `mysqldump`.

After the backup operation on M1 completes, restore M1 to its normal operational state by executing these statements:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

Although performing the backup on M1 is safe (as far as the backup is concerned), it is not optimal for performance because clients of M1 are blocked from executing updates.

This strategy applies to backing up a master server in a replication setup, but can also be used for a single server in a nonreplication setting.

Scenario 2: Backup with a Read-Only Slave

Put the slave S1 in a read-only state by executing these statements on it:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

While S1 is in a read-only state, the following properties are true:

- The master M1 will continue to operate, so making a backup on the master is not safe.
- The slave S1 is stopped, so making a backup on the slave S1 is safe.

These properties provide the basis for a popular backup scenario: Having one slave busy performing a backup for a while is not a problem because it does not affect the entire network, and the system is still running during the backup. In particular, clients can still perform updates on the master server, which remains unaffected by backup activity on the slave.

While S1 is read only, perform the backup. For example, you can use `mysqldump`.

After the backup operation on S1 completes, restore S1 to its normal operational state by executing these statements:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

After the slave is restored to normal operation, it again synchronizes to the master by catching up with any outstanding updates from the binary log of the master.

16.3.2. Using Replication with Different Master and Slave Storage Engines

It does not matter for the replication process whether the source table on the master and the replicated table on the slave use different engine types. In fact, the `default_storage_engine` [494] and `storage_engine` [559] system variables are not replicated.

This provides a number of benefits in the replication process in that you can take advantage of different engine types for different replication scenarios. For example, in a typical scale-out scenario (see [Section 16.3.3, “Using Replication for Scale-Out”](#)), you want to use `InnoDB` tables on the master to take advantage of the transactional functionality, but use `MyISAM` on the slaves where transaction support is

not required because the data is only read. When using replication in a data-logging environment you may want to use the [Archive](#) storage engine on the slave.

Configuring different engines on the master and slave depends on how you set up the initial replication process:

- If you used `mysqldump` to create the database snapshot on your master, you could edit the dump file text to change the engine type used on each table.

Another alternative for `mysqldump` is to disable engine types that you do not want to use on the slave before using the dump to build the data on the slave. For example, you can add the `--skip-innodb` [1546] option on your slave to disable the `InnoDB` engine. If a specific engine does not exist for a table to be created, MySQL will use the default engine type, usually `MyISAM`. (This requires that the `NO_ENGINE_SUBSTITUTION` [614] SQL mode is not enabled.) If you want to disable additional engines in this way, you may want to consider building a special binary to be used on the slave that only supports the engines you want.

- If you are using raw data files (a binary backup) to set up the slave, you will be unable to change the initial table format. Instead, use `ALTER TABLE` to change the table types after the slave has been started.
- For new master/slave replication setups where there are currently no tables on the master, avoid specifying the engine type when creating new tables.

If you are already running a replication solution and want to convert your existing tables to another engine type, follow these steps:

1. Stop the slave from running replication updates:

```
mysql> STOP SLAVE;
```

This will enable you to change engine types without interruptions.

2. Execute an `ALTER TABLE ... ENGINE='engine_type'` for each table to be changed.
3. Start the slave replication process again:

```
mysql> START SLAVE;
```

Although the `default_storage_engine` [494] variable is not replicated, be aware that `CREATE TABLE` and `ALTER TABLE` statements that include the engine specification will be correctly replicated to the slave. For example, if you have a CSV table and you execute:

```
mysql> ALTER TABLE csvtable Engine='MyISAM';
```

The above statement will be replicated to the slave and the engine type on the slave will be converted to `MyISAM`, even if you have previously changed the table type on the slave to an engine other than CSV. If you want to retain engine differences on the master and slave, you should be careful to use the `default_storage_engine` [494] variable on the master when creating a new table. For example, instead of:

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

Use this format:

```
mysql> SET default_storage_engine=MyISAM;  
mysql> CREATE TABLE tablea (columna int);
```

When replicated, the `default_storage_engine` [494] variable will be ignored, and the `CREATE TABLE` statement will execute on the slave using the slave's default engine.

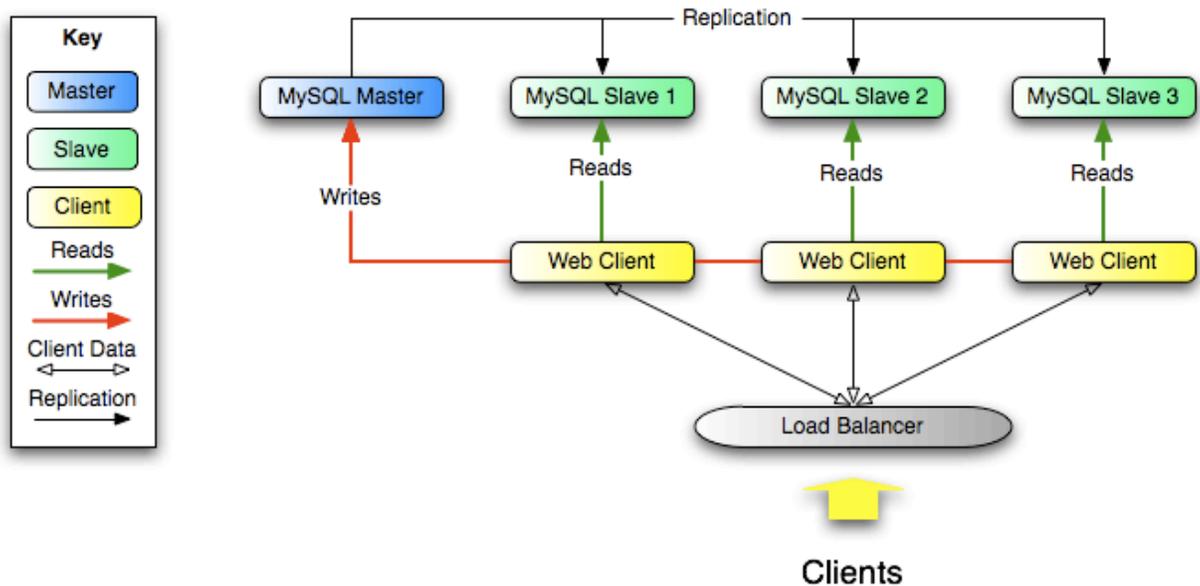
16.3.3. Using Replication for Scale-Out

You can use replication as a scale-out solution; that is, where you want to split up the load of database queries across multiple database servers, within some reasonable limitations.

Because replication works from the distribution of one master to one or more slaves, using replication for scale-out works best in an environment where you have a high number of reads and low number of writes/updates. Most Web sites fit into this category, where users are browsing the Web site, reading articles, posts, or viewing products. Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.

Replication in this situation enables you to distribute the reads over the replication slaves, while still enabling your web servers to communicate with the replication master when a write is required. You can see a sample replication layout for this scenario in [Figure 16.1, “Using Replication to Improve Performance During Scale-Out”](#).

Figure 16.1. Using Replication to Improve Performance During Scale-Out



If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to clean it up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

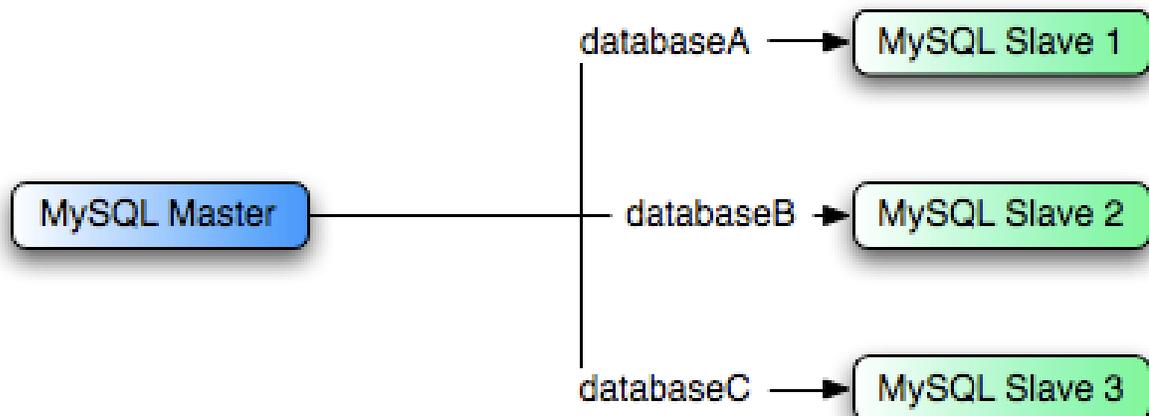
Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions; for example, to log how long each statement took, or which statement among those issued gave you an error.

If you have written a lot of code, you may want to automate the conversion task by using the `replace` utility that comes with standard MySQL distributions, or write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

16.3.4. Replicating Different Databases to Different Slaves

There may be situations where you have a single master and want to replicate different databases to different slaves. For example, you may want to distribute different sales data to different departments to help spread the load during data analysis. A sample of this layout is shown in [Figure 16.2, “Using Replication to Replicate Databases to Separate Replication Slaves”](#).

Figure 16.2. Using Replication to Replicate Databases to Separate Replication Slaves



You can achieve this separation by configuring the master and slaves as normal, and then limiting the binary log statements that each slave processes by using the `--replicate-wild-do-table` [\[1879\]](#) configuration option on each slave.



Important

You should *not* use `--replicate-do-db` [\[1875\]](#) for this purpose when using statement-based replication, since statement-based replication causes this option's affects to vary according to the database that is currently selected. This applies to mixed-format replication as well, since this enables some updates to be replicated using the statement-based format.

However, it should be safe to use `--replicate-do-db` [\[1875\]](#) for this purpose if you are using row-based replication only, since in this case the currently selected database has no effect on the option's operation.

For example, to support the separation as shown in [Figure 16.2, “Using Replication to Replicate Databases to Separate Replication Slaves”](#), you should configure each replication slave as follows, before executing `START SLAVE`:

- Replication slave 1 should use `--replicate-wild-do-table=databaseA.%`.
- Replication slave 2 should use `--replicate-wild-do-table=databaseB.%`.
- Replication slave 3 should use `--replicate-wild-do-table=databaseC.%`.

Each slave in this configuration receives the entire binary log from the master, but executes only those events from the binary log that apply to the databases and tables included by the `--replicate-wild-do-table` [1879] option in effect on that slave.

If you have data that must be synchronized to the slaves before replication starts, you have a number of choices:

- Synchronize all the data to each slave, and delete the databases, tables, or both that you do not want to keep.
- Use `mysqldump` to create a separate dump file for each database and load the appropriate dump file on each slave.
- Use a raw data file dump and include only the specific files and databases that you need for each slave.



Note

This does not work with `InnoDB` databases unless you use `innodb_file_per_table` [1555].

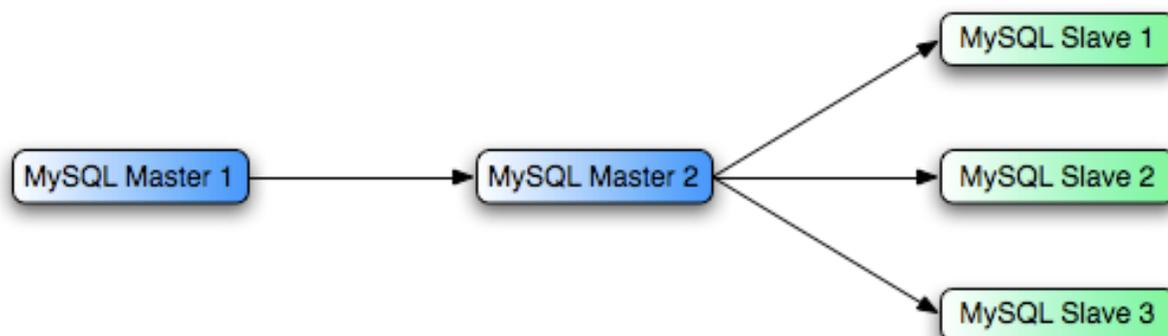
16.3.5. Improving Replication Performance

As the number of slaves connecting to a master increases, the load, although minimal, also increases, as each slave uses a client connection to the master. Also, as each slave must receive a full copy of the master binary log, the network load on the master may also increase and create a bottleneck.

If you are using a large number of slaves connected to one master, and that master is also busy processing requests (for example, as part of a scale-out solution), then you may want to improve the performance of the replication process.

One way to improve the performance of the replication process is to create a deeper replication structure that enables the master to replicate to only one slave, and for the remaining slaves to connect to this primary slave for their individual replication requirements. A sample of this structure is shown in [Figure 16.3, “Using an Additional Replication Host to Improve Performance”](#).

Figure 16.3. Using an Additional Replication Host to Improve Performance



For this to work, you must configure the MySQL instances as follows:

- Master 1 is the primary master where all changes and updates are written to the database. Binary logging should be enabled on this machine.
- Master 2 is the slave to the Master 1 that provides the replication functionality to the remainder of the slaves in the replication structure. Master 2 is the only machine permitted to connect to Master 1. Master 2 also has binary logging enabled, and the `--log-slave-updates` [1870] option so that replication instructions from Master 1 are also written to Master 2's binary log so that they can then be replicated to the true slaves.
- Slave 1, Slave 2, and Slave 3 act as slaves to Master 2, and replicate the information from Master 2, which actually consists of the upgrades logged on Master 1.

The above solution reduces the client load and the network interface load on the primary master, which should improve the overall performance of the primary master when used as a direct database solution.

If your slaves are having trouble keeping up with the replication process on the master, there are a number of options available:

- If possible, put the relay logs and the data files on different physical drives. To do this, use the `--relay-log` [1873] option to specify the location of the relay log.
- If the slaves are significantly slower than the master, you may want to divide up the responsibility for replicating different databases to different slaves. See [Section 16.3.4, “Replicating Different Databases to Different Slaves”](#).
- If your master makes use of transactions and you are not concerned about transaction support on your slaves, use `MyISAM` or another nontransactional engine on the slaves. See [Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”](#).
- If your slaves are not acting as masters, and you have a potential solution in place to ensure that you can bring up a master in the event of failure, then you can switch off `--log-slave-updates` [1870]. This prevents “dumb” slaves from also logging events they have executed into their own binary log.

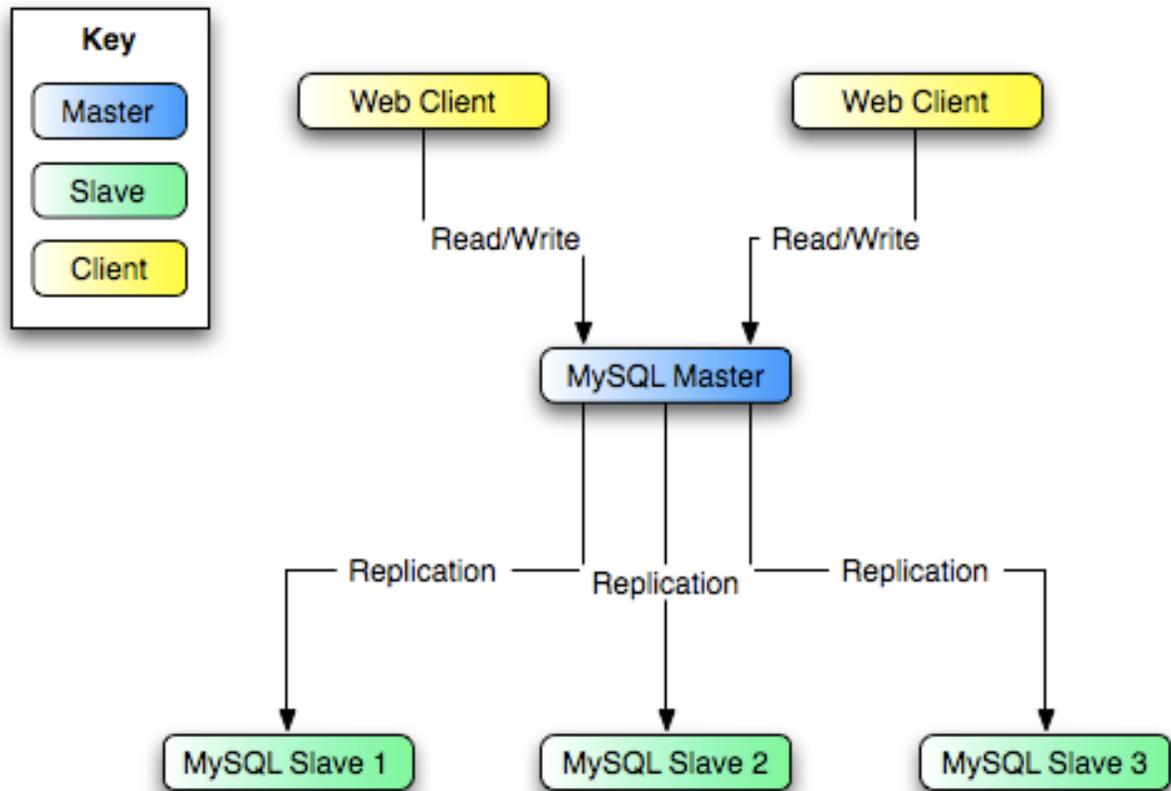
16.3.6. Switching Masters During Failover

There is currently no official solution for providing failover between master and slaves in the event of a failure. With the currently available features, you would have to set up a master and a slave (or several slaves), and to write a script that monitors the master to check whether it is up. Then instruct your applications and the slaves to change master in case of failure.

Remember that you can tell a slave to change its master at any time, using the `CHANGE MASTER TO` statement. The slave will not check whether the databases on the master are compatible with the slave, it will just start reading and executing events from the specified binary log coordinates on the new master. In a failover situation, all the servers in the group are typically executing the same events from the same binary log file, so changing the source of the events should not affect the database structure or integrity providing you are careful.

Run your slaves with the `--log-bin` [1892] option and without `--log-slave-updates` [1870]. In this way, the slave is ready to become a master as soon as you issue `STOP SLAVE; RESET MASTER`, and `CHANGE MASTER TO` statement on the other slaves. For example, assume that you have the structure shown in [Figure 16.4, “Redundancy Using Replication, Initial Structure”](#).

Figure 16.4. Redundancy Using Replication, Initial Structure



In this diagram, the `MySQL Master` holds the master database, the `MySQL Slave` hosts are replication slaves, and the `Web Client` machines are issuing database reads and writes. Web clients that issue only reads (and would normally be connected to the slaves) are not shown, as they do not need to switch to a new server in the event of a failure. For a more detailed example of a read/write scale-out replication structure, see [Section 16.3.3, “Using Replication for Scale-Out”](#).

Each `MySQL Slave` (`Slave 1`, `Slave 2`, and `Slave 3`) is a slave running with `--log-bin` [1892] and without `--log-slave-updates` [1870]. Because updates received by a slave from the master are not logged in the binary log unless `--log-slave-updates` [1870] is specified, the binary log on each slave is empty initially. If for some reason `MySQL Master` becomes unavailable, you can pick one of the slaves to become the new master. For example, if you pick `Slave 1`, all `Web Clients` should be redirected to `Slave 1`, which will log updates to its binary log. `Slave 2` and `Slave 3` should then replicate from `Slave 1`.

The reason for running the slave without `--log-slave-updates` [1870] is to prevent slaves from receiving updates twice in case you cause one of the slaves to become the new master. Suppose that `Slave 1` has `--log-slave-updates` [1870] enabled. Then it will write updates that it receives from `Master` to its own binary log. When `Slave 2` changes from `Master` to `Slave 1` as its master, it may receive updates from `Slave 1` that it has already received from `Master`.

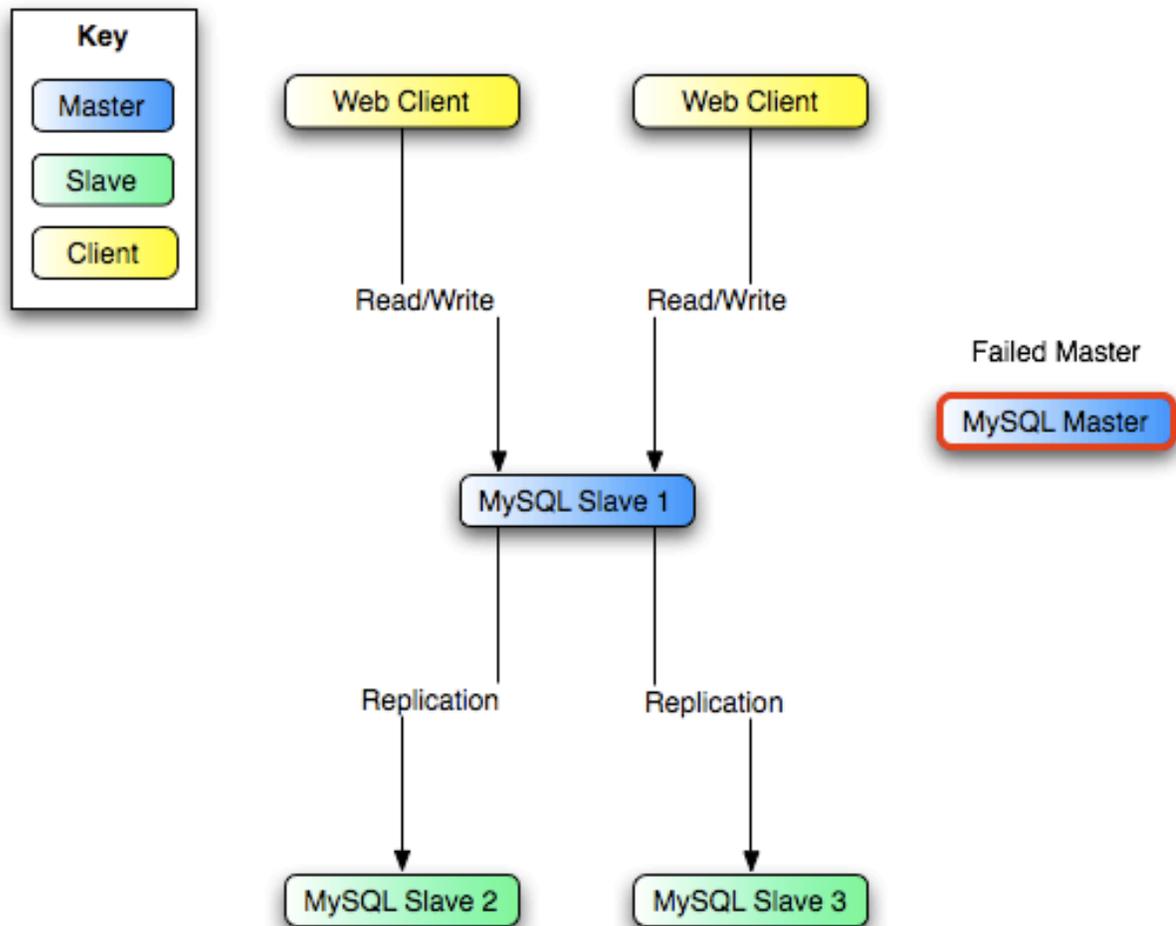
Make sure that all slaves have processed any statements in their relay log. On each slave, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all slaves, they can be reconfigured to the new setup. On the slave `Slave 1` being promoted to become the master, issue `STOP SLAVE` and `RESET MASTER`.

On the other slaves *Slave 2* and *Slave 3*, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='Slave1'` (where `'Slave1'` represents the real host name of *Slave 1*). To use `CHANGE MASTER TO`, add all information about how to connect to *Slave 1* from *Slave 2* or *Slave 3* (`user`, `password`, `port`). In `CHANGE MASTER TO`, there is no need to specify the name of the *Slave 1* binary log file or log position to read from: We know it is the first binary log file and position 4, which are the defaults for `CHANGE MASTER TO`. Finally, use `START SLAVE` on *Slave 2* and *Slave 3*.

Once the new replication is in place, you will then need to instruct each *Web Client* to direct its statements to *Slave 1*. From that point on, all updates statements sent by *Web Client* to *Slave 1* are written to the binary log of *Slave 1*, which then contains every update statement sent to *Slave 1* since *Master* died.

The resulting server structure is shown in [Figure 16.5, “Redundancy Using Replication, After Master Failure”](#).

Figure 16.5. Redundancy Using Replication, After Master Failure



When *Master* is up again, you must issue on it the same `CHANGE MASTER TO` as that issued on *Slave 2* and *Slave 3*, so that *Master* becomes a slave of *S1* and picks up each *Web Client* writes that it missed while it was down.

To make *Master* a master again (for example, because it is the most powerful machine), use the preceding procedure as if *Slave 1* was unavailable and *Master* was to be the new master. During this

procedure, do not forget to run `RESET MASTER` on `Master` before making `Slave 1`, `Slave 2`, and `Slave 3` slaves of `Master`. Otherwise, they may pick up old `Web Client` writes from before the point at which `Master` became unavailable.

Note that there is no synchronization between the different slaves to a master. Some slaves might be ahead of others. This means that the concept outlined in the previous example might not work. In practice, however, the relay logs of different slaves will most likely not be far behind the master, so it would work, anyway (but there is no guarantee).

A good way to keep your applications informed as to the location of the master is by having a dynamic DNS entry for the master. With `bind` you can use `nsupdate` to dynamically update your DNS.

16.3.7. Setting Up Replication Using SSL

To use SSL for encrypting the transfer of the binary log required during replication, both the master and the slave must support SSL network connections. If either host does not support SSL connections (because it has not been compiled or configured for SSL), replication through an SSL connection is not possible.

Setting up replication using an SSL connection is similar to setting up a server and client using SSL. You must obtain (or create) a suitable security certificate that you can use on the master, and a similar certificate (from the same certificate authority) on each slave.

For more information on setting up a server and client for SSL connectivity, see [Section 6.3.8.2, “Configuring MySQL for SSL”](#).

To enable SSL on the master you must create or obtain suitable certificates, and then add the following configuration options to the master's configuration within the `[mysqld]` section of the master's `my.cnf` file:

```
[mysqld]
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

The paths to the certificates may be relative or absolute; we recommend that you always use complete paths for this purpose.

The options are as follows:

- `ssl-ca` identifies the Certificate Authority (CA) certificate.
- `ssl-cert` identifies the server public key. This can be sent to the client and authenticated against the CA certificate that it has.
- `ssl-key` identifies the server private key.

On the slave, you have two options available for setting the SSL information. You can either add the slave certificates to the `[client]` section of the slave's `my.cnf` file, or you can explicitly specify the SSL information using the `CHANGE MASTER TO` statement:

- To add the slave certificates using an option file, add the following lines to the `[client]` section of the slave's `my.cnf` file:

```
[client]
ssl-ca=cacert.pem
ssl-cert=client-cert.pem
ssl-key=client-key.pem
```

Restart the slave server, using the `--skip-slave-start` [1882] option to prevent the slave from connecting to the master. Use `CHANGE MASTER TO` to specify the master configuration, using the `MASTER_SSL` option to enable SSL connectivity:

```
mysql> CHANGE MASTER TO
-> MASTER_HOST='master_hostname',
-> MASTER_USER='replicate',
-> MASTER_PASSWORD='password',
-> MASTER_SSL=1;
```

- To specify the SSL certificate options using the `CHANGE MASTER TO` statement, append the SSL options:

```
mysql> CHANGE MASTER TO
-> MASTER_HOST='master_hostname',
-> MASTER_USER='replicate',
-> MASTER_PASSWORD='password',
-> MASTER_SSL=1,
-> MASTER_SSL_CA = 'ca_file_name',
-> MASTER_SSL_CAPATH = 'ca_directory_name',
-> MASTER_SSL_CERT = 'cert_file_name',
-> MASTER_SSL_KEY = 'key_file_name';
```

After the master information has been updated, start the slave replication process:

```
mysql> START SLAVE;
```

You can use the `SHOW SLAVE STATUS` statement to confirm that the SSL connection was established successfully.

For more information on the `CHANGE MASTER TO` statement, see [Section 13.4.2.1, “CHANGE MASTER TO Syntax”](#).

If you want to enforce the use of SSL connections during replication, then create a user with the `REPLICATION SLAVE` [697] privilege and use the `REQUIRE SSL` option for that user. For example:

```
mysql> CREATE USER 'repl'@'%mydomain.com' IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.*
-> TO 'repl'@'%mydomain.com' REQUIRE SSL;
```

If the account already exists, you can add `REQUIRE SSL` to it with this statement:

```
mysql> GRANT USAGE ON *.*
-> TO 'repl'@'%mydomain.com' REQUIRE SSL;
```

16.3.8. Semisynchronous Replication

MySQL 5.5 supports an interface to semisynchronous replication in addition to the built-in asynchronous replication. This section discusses what semisynchronous replication is and how it works. The following sections cover the administrative interface to semisynchronous replication and how to install, configure, and monitor it.

MySQL replication by default is asynchronous. The master writes events to its binary log but does not know whether or when a slave has retrieved and processed them. With asynchronous replication, if the master crashes, transactions that it has committed might not have been transmitted to any slave. Consequently, failover from master to slave in this case may result in failover to a server that is missing transactions relative to the master.

Semisynchronous replication can be used as an alternative to asynchronous replication:

- A slave indicates whether it is semisynchronous-capable when it connects to the master.
- If semisynchronous replication is enabled on the master side and there is at least one semisynchronous slave, a thread that performs a transaction commit on the master blocks after the commit is done and waits until at least one semisynchronous slave acknowledges that it has received all events for the transaction, or until a timeout occurs.
- The slave acknowledges receipt of a transaction's events only after the events have been written to its relay log and flushed to disk.
- If a timeout occurs without any slave having acknowledged the transaction, the master reverts to asynchronous replication. When at least one semisynchronous slave catches up, the master returns to semisynchronous replication.
- Semisynchronous replication must be enabled on both the master and slave sides. If semisynchronous replication is disabled on the master, or enabled on the master but on no slaves, the master uses asynchronous replication.

While the master is blocking (waiting for acknowledgment from a slave after having performed a commit), it does not return to the session that performed the transaction. When the block ends, the master returns to the session, which then can proceed to execute other statements. At this point, the transaction has committed on the master side, and receipt of its events has been acknowledged by at least one slave.

Blocking also occurs after rollbacks that are written to the binary log, which occurs when a transaction that modifies nontransactional tables is rolled back. The rolled-back transaction is logged even though it has no effect for transactional tables because the modifications to the nontransactional tables cannot be rolled back and must be sent to slaves.

For statements that do not occur in transactional context (that is, when no transaction has been started with `START TRANSACTION` or `SET autocommit = 0`), autocommit is enabled and each statement commits implicitly. With semisynchronous replication, the master blocks after committing each such statement, just as it does for explicit transaction commits.

To understand what the “semi” in “semisynchronous replication” means, compare it with asynchronous and fully synchronous replication:

- With asynchronous replication, the master writes events to its binary log and slaves request them when they are ready. There is no guarantee that any event will ever reach any slave.
- With fully synchronous replication, when a master commits a transaction, all slaves also will have committed the transaction before the master returns to the session that performed the transaction. The drawback of this is that there might be a lot of delay to complete a transaction.
- Semisynchronous replication falls between asynchronous and fully synchronous replication. The master waits after commit only until at least one slave has received and logged the events. It does not wait for all slaves to acknowledge receipt, and it requires only receipt, not that the events have been fully executed and committed on the slave side.

Compared to asynchronous replication, semisynchronous replication provides improved data integrity. When a commit returns successfully, it is known that the data exists in at least two places (on the master and at least one slave). If the master commits but a crash occurs while the master is waiting for acknowledgment from a slave, it is possible that the transaction may not have reached any slave.

Semisynchronous replication also places a rate limit on busy sessions by constraining the speed at which binary log events can be sent from master to slave. When one user is too busy, this will slow it down, which is useful in some deployment situations.

Semisynchronous replication does have some performance impact because commits are slower due to the need to wait for slaves. This is the tradeoff for increased data integrity. The amount of slowdown is at least the TCP/IP roundtrip time to send the commit to the slave and wait for the acknowledgment of receipt by the slave. This means that semisynchronous replication works best for close servers communicating over fast networks, and worst for distant servers communicating over slow networks.

16.3.8.1. Semisynchronous Replication Administrative Interface

The administrative interface to semisynchronous replication has several components:

- Two plugins implement semisynchronous capability. There is one plugin for the master side and one for the slave side.

- System variables control plugin behavior. Some examples:

- `rpl_semi_sync_master_enabled` [548]

Controls whether semisynchronous replication is enabled on the master. To enable or disable the plugin, set this variable to 1 or 0, respectively. The default is 1.

- `rpl_semi_sync_master_timeout` [548]

A value in milliseconds that controls how long the master waits on a commit for acknowledgment from a slave before timing out and reverting to asynchronous replication. The default value is 10000 (10 seconds).

- `rpl_semi_sync_slave_enabled` [550]

Similar to `rpl_semi_sync_master_enabled` [548], but controls the slave plugin.

All `rpl_semi_sync_xxx` system variables are described at [Section 5.1.4, “Server System Variables”](#).

- Status variables enable semisynchronous replication monitoring. Some examples:

- `Rpl_semi_sync_master_clients` [605]

The number of semisynchronous slaves.

- `Rpl_semi_sync_master_status` [606]

Whether semisynchronous replication currently is operational on the master. The value is 1 if the plugin has been enabled and a commit acknowledgment has occurred. It is 0 if the plugin is not enabled or the master has fallen back to asynchronous replication due to commit acknowledgment timeout.

- `Rpl_semi_sync_master_no_tx` [606]

The number of commits that were not acknowledged successfully by a slave.

- `Rpl_semi_sync_master_yes_tx` [607]

The number of commits that were acknowledged successfully by a slave.

- `Rpl_semi_sync_slave_status` [607]

Whether semisynchronous replication currently is operational on the slave. This is 1 if the plugin has been enabled and the slave I/O thread is running, 0 otherwise.

All `Rpl_semi_sync_xxx` status variables are described at [Section 5.1.6, “Server Status Variables”](#).

The system and status variables are available only if the appropriate master or slave plugin has been installed with `INSTALL PLUGIN`.

16.3.8.2. Semisynchronous Replication Installation and Configuration

Semisynchronous replication is implemented using plugins, so the plugins must be installed into the server to make them available. After a plugin has been installed, you control it by means of the system variables associated with it. These system variables are unavailable until the associated plugin has been installed.

To use semisynchronous replication, the following requirements must be satisfied:

- MySQL 5.5 or higher must be installed.
- The capability of installing plugins requires a MySQL server that supports dynamic loading. To verify this, check that the value of the `have_dynamic_loading` [504] system variable is `YES`. Binary distributions should support dynamic loading.
- Replication must already be working. For information on creating a master/slave relationship, see [Section 16.1.1, “How to Set Up Replication”](#).

To set up semisynchronous replication, use the following instructions. The `INSTALL PLUGIN`, `SET GLOBAL`, `STOP SLAVE`, and `START SLAVE` statements mentioned here require the `SUPER` [697] privilege.

The semisynchronous replication plugins are included with MySQL distributions.

Unpack the component distribution, which contains files for the master side and the slave side.

Install the component files in the plugin directory of the appropriate server. Install the `semisync_master*` files in the plugin directory of the master server. Install the `semisync_slave*` files in the plugin directory of each slave server. The location of the plugin directory is available as the value of the server's `plugin_dir` [537] system variable.

To load the plugins, use the `INSTALL PLUGIN` statement on the master and on each slave that is to be semisynchronous.

On the master:

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
```

On each slave:

```
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

The preceding commands use a plugin file name suffix of `.so`. A different suffix might apply on your system. If you are not sure about the plugin file name, look for the plugins in the server's plugin directory.

If an attempt to install a plugin results in an error on Linux similar to that shown here, you will need to install `libimf`:

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
ERROR 1126 (HY000): Can't open shared library
'/usr/local/mysql/lib/plugin/semisync_master.so' (errno: 22 libimf.so: cannot open
shared object file: No such file or directory)
```

You can obtain `libimf` from <http://dev.mysql.com/downloads/os-linux.html>.

To see which plugins are installed, use the `SHOW PLUGINS` statement, or query the `INFORMATION_SCHEMA.PLUGINS` table.

After a semisynchronous replication plugin has been installed, it is disabled by default. The plugins must be enabled both on the master side and the slave side to enable semisynchronous replication. If only one side is enabled, replication will be asynchronous.

To control whether an installed plugin is enabled, set the appropriate system variables. You can set these variables at runtime using `SET GLOBAL`, or at server startup on the command line or in an option file.

At runtime, these master-side system variables are available:

```
mysql> SET GLOBAL rpl_semi_sync_master_enabled = {0|1};
mysql> SET GLOBAL rpl_semi_sync_master_timeout = N;
```

On the slave side, this system variable is available:

```
mysql> SET GLOBAL rpl_semi_sync_slave_enabled = {0|1};
```

For `rpl_semi_sync_master_enabled` [548] or `rpl_semi_sync_slave_enabled` [550], the value should be 1 to enable semisynchronous replication or 0 to disable it. By default, these variables are set to 1.

For `rpl_semi_sync_master_timeout` [548], the value `N` is given in milliseconds. The default value is 10000 (10 seconds).

If you enable semisynchronous replication on a slave at runtime, you must also start the slave I/O thread (stopping it first if it is already running) to cause the slave to connect to the master and register as a semisynchronous slave:

```
mysql> STOP SLAVE IO_THREAD; START SLAVE IO_THREAD;
```

If the I/O thread is already running and you do not restart it, the slave continues to use asynchronous replication.

At server startup, the variables that control semisynchronous replication can be set as command-line options or in an option file. A setting listed in an option file takes effect each time the server starts. For example, you can set the variables in `my.cnf` files on the master and slave sides as follows.

On the master:

```
[mysqld]
rpl_semi_sync_master_enabled=1
rpl_semi_sync_master_timeout=1000 # 1 second
```

On each slave:

```
[mysqld]
rpl_semi_sync_slave_enabled=1
```

16.3.8.3. Semisynchronous Replication Monitoring

The plugins for the semisynchronous replication capability expose several system and status variables that you can examine to determine its configuration and operational state.

The system variable reflect how semisynchronous replication is configured. To check their values, use `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'rpl_semi_sync%';
```

The status variables enable you to monitor the operation of semisynchronous replication. To check their values, use `SHOW STATUS`:

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync%';
```

When the master switches between asynchronous or semisynchronous replication due to commit-blocking timeout or a slave catching up, it sets the value of the `Rpl_semi_sync_master_status` [606] status variable appropriately. Automatic fallback from semisynchronous to asynchronous replication on the master means that it is possible for the `rpl_semi_sync_master_enabled` [548] system variable to have a value of 1 on the master side even when semisynchronous replication is in fact not operational at the moment. You can monitor the `Rpl_semi_sync_master_status` [606] status variable to determine whether the master currently is using asynchronous or semisynchronous replication.

To see how many semisynchronous slaves are connected, check `Rpl_semi_sync_master_clients` [605].

The number of commits that have been acknowledged successfully or unsuccessfully by slaves are indicated by the `Rpl_semi_sync_master_yes_tx` [607] and `Rpl_semi_sync_master_no_tx` [606] variables.

On the slave side, `Rpl_semi_sync_slave_status` [607] indicates whether semisynchronous replication currently is operational.

16.4. Replication Notes and Tips

16.4.1. Replication Features and Issues

The following sections provide information about what is supported and what is not in MySQL replication, and about specific issues and situations that may occur when replicating certain statements.

Statement-based replication depends on compatibility at the SQL level between the master and slave. In others, successful SBR requires that any SQL features used be supported by both the master and the slave servers. For example, if you use a feature on the master server that is available only in MySQL 5.5 (or later), you cannot replicate to a slave that uses MySQL 5.1 (or earlier).

Such incompatibilities also can occur within a release series when using pre-production releases of MySQL. For example, the `SLEEP()` [1201] function is available beginning with MySQL 5.0.12. If you use this function on the master, you cannot replicate to a slave that uses MySQL 5.0.11 or earlier.

For this reason, use Generally Available (GA) releases of MySQL for statement-based replication in a production setting, since we do not introduce new SQL statements or change their behavior within a given release series once that series reaches GA release status.

If you are planning to use statement-based replication between MySQL 5.5 and a previous MySQL release series, it is also a good idea to consult the edition of the *MySQL Reference Manual* corresponding to the earlier release series for information regarding the replication characteristics of that series.

With MySQL's statement-based replication, there may be issues with replicating stored routines or triggers. You can avoid these issues by using MySQL's row-based replication instead. For a detailed list of issues, see [Section 19.7, “Binary Logging of Stored Programs”](#). For more information about row-based logging and row-based replication, see [Section 5.2.4.1, “Binary Logging Formats”](#), and [Section 16.1.2, “Replication Formats”](#).

For additional information specific to replication and [InnoDB](#), see [Section 14.3.5.5, “InnoDB and MySQL Replication”](#). For information relating to replication with MySQL Cluster, see [Section 17.6, “MySQL Cluster Replication”](#).

16.4.1.1. Replication and `AUTO_INCREMENT`

Statement-based replication of `AUTO_INCREMENT`, `LAST_INSERT_ID()` [1194], and `TIMESTAMP` values is done correctly, subject to the following exceptions:

- When using statement-based replication prior to MySQL 5.5.30, `AUTO_INCREMENT` columns in tables on the slave must match the same columns on the master; that is, `AUTO_INCREMENT` columns must be replicated to `AUTO_INCREMENT` columns. (Bug #12669186)
- A statement invoking a trigger or function that causes an update to an `AUTO_INCREMENT` column is not replicated correctly using statement-based replication. In MySQL 5.5, such statements are marked as unsafe. (Bug #45677)
- An `INSERT` into a table that has a composite primary key that includes an `AUTO_INCREMENT` column that is not the first column of this composite key is not safe for statement-based logging or replication. Beginning with MySQL 5.5.25, such statements are marked as unsafe. (Bug #11754117, Bug #45670)

This issue does not affect tables using the `InnoDB` storage engine, since `InnoDB` does not allow the creation of a composite key that includes an `AUTO_INCREMENT` column that is not the first column in the key.

- Adding an `AUTO_INCREMENT` column to a table with `ALTER TABLE` might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to a table `t1` that has columns `col1` and `col2`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```



Important

To guarantee the same ordering on both master and slave, the `ORDER BY` clause must name *all* columns of `t1`.

The instructions just given are subject to the limitations of `CREATE TABLE ... LIKE`: Foreign key definitions are ignored, as are the `DATA DIRECTORY` and `INDEX DIRECTORY` table options. If a table definition includes any of those characteristics, create `t2` using a `CREATE TABLE` statement that is identical to the one used to create `t1`, but with the addition of the `AUTO_INCREMENT` column.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

See also [Section C.5.7.1, “Problems with `ALTER TABLE`”](#).

16.4.1.2. Replication and Character Sets

The following applies to replication between MySQL servers that use different character sets:

- If the master uses MySQL 4.1, you must *always* use the same *global* character set and collation on the master and the slave, regardless of the slave MySQL version. (These are controlled by the `--character-set-server` [438] and `--collation-server` [438] options.) Otherwise, you may get

duplicate-key errors on the slave, because a key that is unique in the master character set might not be unique in the slave character set. Note that this is not a cause for concern when master and slave are both MySQL 5.0 or later.

- If the master is older than MySQL 4.1.3, the character set of any client should never be made different from its global value because this character set change is not known to the slave. In other words, clients should not use `SET NAMES`, `SET CHARACTER SET`, and so forth. If both the master and the slave are 4.1.3 or newer, clients can freely set session values for character set variables because these settings are written to the binary log and so are known to the slave. That is, clients can use `SET NAMES` or `SET CHARACTER SET` or can set variables such as `collation_client` or `collation_server` [489]. However, clients are prevented from changing the *global* value of these variables; as stated previously, the master and slave must always have identical global character set values. This is true whether you are using statement-based or row-based replication.
- If the master has databases with a character set different from the global `character_set_server` [488] value, you should design your `CREATE TABLE` statements so that they do not implicitly rely on the database default character set. A good workaround is to state the character set and collation explicitly in `CREATE TABLE` statements.

16.4.1.3. Replication of `CREATE ... IF NOT EXISTS` Statements

MySQL applies these rules when various `CREATE ... IF NOT EXISTS` statements are replicated:

- Every `CREATE DATABASE IF NOT EXISTS` statement is replicated, whether or not the database already exists on the master.
- Similarly, every `CREATE TABLE IF NOT EXISTS` statement without a `SELECT` is replicated, whether or not the table already exists on the master. This includes `CREATE TABLE IF NOT EXISTS ... LIKE`. Replication of `CREATE TABLE IF NOT EXISTS ... SELECT` follows somewhat different rules; see [Section 16.4.1.4, “Replication of `CREATE TABLE ... SELECT` Statements”](#), for more information.
- `CREATE EVENT IF NOT EXISTS` is always replicated in MySQL 5.5, whether or not the event named in the statement already exists on the master.

See also Bug #45574.

16.4.1.4. Replication of `CREATE TABLE ... SELECT` Statements

This section discusses how MySQL replicates `CREATE TABLE ... SELECT` statements.

These behaviors are not dependent on MySQL version:

- `CREATE TABLE ... SELECT` always performs an implicit commit ([Section 13.3.3, “Statements That Cause an Implicit Commit”](#)).
- If destination table does not exist, logging occurs as follows. It does not matter whether `IF NOT EXISTS` is present.
 - `STATEMENT` or `MIXED` format: The statement is logged as written.
 - `ROW` format: The statement is logged as a `CREATE TABLE` statement followed by a series of insert-row events.
- If the statement fails, nothing is logged. This includes the case that the destination table exists and `IF NOT EXISTS` is not given.

When the destination table exists and `IF NOT EXISTS` is given, MySQL handles the statement in a version-dependent way.

In MySQL 5.1 before 5.1.51 and in MySQL 5.5 before 5.5.6 (this is the original behavior):

- `STATEMENT` or `MIXED` format: The statement is logged as written.
- `ROW` format: The statement is logged as a `CREATE TABLE` statement followed by a series of insert-row events.

In MySQL 5.1 as of 5.1.51:

- `STATEMENT` or `MIXED` format: The statement is logged as the equivalent pair of `CREATE TABLE` and `INSERT INTO ... SELECT` statements.
- `ROW` format: The statement is logged as a `CREATE TABLE` statement followed by a series of insert-row events.

In MySQL 5.5 as of 5.5.6:

- Nothing is inserted or logged.

These version dependencies arise due to a change in MySQL 5.5.6 in handling of `CREATE TABLE ... SELECT` not to insert rows if the destination table already exists, and a change made in MySQL 5.1.51 to preserve forward compatibility in replication of such statements from a 5.1 master to a 5.5 slave. For details, see [Section 13.1.17.1, “CREATE TABLE ... SELECT Syntax”](#).

When using statement-based replication between a MySQL 5.6 or later slave and a master running a previous version of MySQL, a `CREATE TABLE ... SELECT` statement causing changes in other tables on the master fails on the slave, causing replication to stop. This is due to the fact that MySQL 5.6 does not allow a `CREATE TABLE ... SELECT` statement to make any changes in tables other than the table that is created by the statement—a change in behavior from previous versions of MySQL, which permitted these statements to do so. To keep this from happening, you should use row-based replication, rewrite the offending statement before running it on the master, or upgrade the master to MySQL 5.6 (or later). (If you choose to upgrade the master, keep in mind that such a `CREATE TABLE ... SELECT` statement will fail there as well, following the upgrade, unless the statement is rewritten to remove any side effects on other tables.) This is not an issue when using row-based replication, because the statement is logged as a `CREATE TABLE` statement with any changes to table data logged as row-insert events (or possibly row-update events), rather than as the entire `CREATE TABLE ... SELECT` statement.

16.4.1.5. Replication of `CREATE SERVER`, `ALTER SERVER`, and `DROP SERVER`

In MySQL 5.5, the statements `CREATE SERVER`, `ALTER SERVER`, and `DROP SERVER` are not written to the binary log, regardless of the binary logging format that is in use.

16.4.1.6. Replication of `CURRENT_USER()`

The following statements support use of the `CURRENT_USER()` [1192] function to take the place of the name of (and, possibly, the host for) an affected user or a definer; in such cases, `CURRENT_USER()` [1192] is expanded where and as needed:

- `DROP USER`
- `RENAME USER`
- `GRANT`
- `REVOKE`
- `CREATE FUNCTION`

- `CREATE PROCEDURE`
- `CREATE TRIGGER`
- `CREATE EVENT`
- `CREATE VIEW`
- `ALTER EVENT`
- `ALTER VIEW`
- `SET PASSWORD`

When `CURRENT_USER()` [1192] or `CURRENT_USER` [1192] is used as the definer in any of the statements `CREATE FUNCTION`, `CREATE PROCEDURE`, `CREATE TRIGGER`, `CREATE EVENT`, `CREATE VIEW`, or `ALTER VIEW` when binary logging is enabled, the function reference is expanded before it is written to the binary log, so that the statement refers to the same user on both the master and the slave when the statement is replicated. `CURRENT_USER()` [1192] or `CURRENT_USER` [1192] is also expanded prior to being written to the binary log when used in `DROP USER`, `RENAME USER`, `GRANT`, `REVOKE`, or `ALTER EVENT`.

16.4.1.7. Replication of `DROP ... IF EXISTS` Statements

The `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, and `DROP VIEW IF EXISTS` statements are always replicated, even if the database, table, or view to be dropped does not exist on the master. This is to ensure that the object to be dropped no longer exists on either the master or the slave, once the slave has caught up with the master.

`DROP ... IF EXISTS` statements for stored programs (stored procedures and functions, triggers, and events) are also replicated, even if the stored program to be dropped does not exist on the master.

16.4.1.8. Replication with Differing Table Definitions on Master and Slave

Source and target tables for replication do not have to be identical. A table on the master can have more or fewer columns than the slave's copy of the table. In addition, corresponding table columns on the master and the slave can use different data types, subject to certain conditions.

In all cases where the source and target tables do not have identical definitions, the database and table names must be the same on both the master and the slave. Additional conditions are discussed, with examples, in the following two sections.

16.4.1.8.1. Replication with More Columns on Master or Slave

You can replicate a table from the master to the slave such that the master and slave copies of the table have differing numbers of columns, subject to the following conditions:

- Columns common to both versions of the table must be defined in the same order on the master and the slave.

(This is true even if both tables have the same number of columns.)

- Columns common to both versions of the table must be defined before any additional columns.

This means that executing an `ALTER TABLE` statement on the slave where a new column is inserted into the table within the range of columns common to both tables causes replication to fail, as shown in the following example:

Suppose that a table `t`, existing on the master and the slave, is defined by the following `CREATE TABLE` statement:

```
CREATE TABLE t (
  c1 INT,
  c2 INT,
  c3 INT
);
```

Suppose that the `ALTER TABLE` statement shown here is executed on the slave:

```
ALTER TABLE t ADD COLUMN cnew1 INT AFTER c3;
```

The previous `ALTER TABLE` is permitted on the slave because the columns `c1`, `c2`, and `c3` that are common to both versions of table `t` remain grouped together in both versions of the table, before any columns that differ.

However, the following `ALTER TABLE` statement cannot be executed on the slave without causing replication to break:

```
ALTER TABLE t ADD COLUMN cnew2 INT AFTER c3;
```

Replication fails after execution on the slave of the `ALTER TABLE` statement just shown, because the new column `cnew2` comes between columns common to both versions of `t`.

- Each “extra” column in the version of the table having more columns must have a default value.



Note

A column's default value is determined by a number of factors, including its type, whether it is defined with a `DEFAULT` option, whether it is declared as `NULL`, and the server SQL mode in effect at the time of its creation; for more information, see [Section 11.5, “Data Type Default Values”](#).

In addition, when the slave's copy of the table has more columns than the master's copy, each column common to the tables must use the same data type in both tables.

Examples. The following examples illustrate some valid and invalid table definitions:

More columns on the master. The following table definitions are valid and replicate correctly:

```
master> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
slave> CREATE TABLE t1 (c1 INT, c2 INT);
```

The following table definitions would raise Error 1532 ([ER_BINLOG_ROW_RBR_TO_SBR \[4015\]](#)) because the definitions of the columns common to both versions of the table are in a different order on the slave than they are on the master:

```
master> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
slave> CREATE TABLE t1 (c2 INT, c1 INT);
```

The following table definitions would also raise Error 1532 because the definition of the extra column on the master appears before the definitions of the columns common to both versions of the table:

```
master> CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
slave> CREATE TABLE t1 (c1 INT, c2 INT);
```

More columns on the slave. The following table definitions are valid and replicate correctly:

```

master> CREATE TABLE t1 (c1 INT, c2 INT);
slave> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);

```

The following definitions raise Error 1532 because the columns common to both versions of the table are not defined in the same order on both the master and the slave:

```

master> CREATE TABLE t1 (c1 INT, c2 INT);
slave> CREATE TABLE t1 (c2 INT, c1 INT, c3 INT);

```

The following table definitions also raise Error 1532 because the definition for the extra column in the slave's version of the table appears before the definitions for the columns which are common to both versions of the table:

```

master> CREATE TABLE t1 (c1 INT, c2 INT);
slave> CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);

```

The following table definitions fail because the slave's version of the table has additional columns compared to the master's version, and the two versions of the table use different data types for the common column `c2`:

```

master> CREATE TABLE t1 (c1 INT, c2 BIGINT);
slave> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);

```

16.4.1.8.2. Replication of Columns Having Different Data Types

Corresponding columns on the master's and the slave's copies of the same table ideally should have the same data type. However, beginning with MySQL 5.1.21, this is not always strictly enforced, as long as certain conditions are met.

All other things being equal, it is always possible to replicate from a column of a given data type to another column of the same type and same size or width, where applicable, or larger. For example, you can replicate from a `CHAR(10)` column to another `CHAR(10)`, or from a `CHAR(10)` column to a `CHAR(25)` column without any problems. In certain cases, it is also possible to replicate from a column having one data type (on the master) to a column having a different data type (on the slave); when the data type of the master's version of the column is promoted to a type that is the same size or larger on the slave, this is known as *attribute promotion*.

Attribute promotion can be used with both statement-based and row-based replication, and is not dependent on the storage engine used by either the master or the slave. However, the choice of logging format does have an effect on the type conversions that are permitted; the particulars are discussed later in this section.



Important

Whether you use statement-based or row-based replication, the slave's copy of the table cannot contain more columns than the master's copy if you wish to employ attribute promotion.

Statement-based replication. When using statement-based replication, a simple rule of thumb to follow is, “If the statement run on the master would also execute successfully on the slave, it should also replicate successfully”. In other words, if the statement uses a value that is compatible with the type of a given column on the slave, the statement can be replicated. For example, you can insert any value that fits in a `TINYINT` column into a `BIGINT` column as well; it follows that, even if you change the type of a `TINYINT` column in the slave's copy of a table to `BIGINT`, any insert into that column on the master that succeeds should also succeed on the slave, since it is impossible to have a legal `TINYINT` value that is large enough to exceed a `BIGINT` column.

Prior to MySQL 5.5.30, when using statement-based replication, `AUTO_INCREMENT` columns were required to be the same on both the master and the slave; otherwise, updates could be applied to the wrong table on the slave. (Bug #12669186)

Row-based replication: attribute promotion and demotion. Formerly, due to the fact that in row-based replication changes rather than statements are replicated, and that these changes are transmitted using formats that do not always map directly to MySQL server column data types, you could not replicate between different subtypes of the same general type (for example, from `TINYINT` to `BIGINT`, both `INT` subtypes). However, beginning with MySQL 5.5.3, MySQL Replication supports attribute promotion and demotion between smaller data types and larger types. It is also possible to specify whether or not to permit lossy (truncated) or non-lossy conversions of demoted column values, as explained later in this section.

Lossy and non-lossy conversions. In the event that the target type cannot represent the value being inserted, a decision must be made on how to handle the conversion. If we permit the conversion but truncate (or otherwise modify) the source value to achieve a “fit” in the target column, we make what is known as a *lossy conversion*. A conversion which does not require truncation or similar modifications to fit the source column value in the target column is a *non-lossy conversion*.

Type conversion modes (`slave_type_conversions` variable). The setting of the `slave_type_conversions` global server variable controls the type conversion mode used on the slave. This variable takes a set of values from the following table, which shows the effects of each mode on the slave's type-conversion behavior:

Mode	Effect
<code>ALL_LOSSY</code>	<p>In this mode, type conversions that would mean loss of information are permitted.</p> <p>This does not imply that non-lossy conversions are permitted, merely that only cases requiring either lossy conversions or no conversion at all are permitted; for example, enabling <i>only</i> this mode permits an <code>INT</code> column to be converted to <code>TINYINT</code> (a lossy conversion), but not a <code>TINYINT</code> column to an <code>INT</code> column (non-lossy). Attempting the latter conversion in this case would cause replication to stop with an error on the slave.</p>
<code>ALL_NON_LOSSY</code>	<p>This mode permits conversions that do not require truncation or other special handling of the source value; that is, it permits conversions where the target type has a wider range than the source type.</p> <p>Setting this mode has no bearing on whether lossy conversions are permitted; this is controlled with the <code>ALL_LOSSY</code> mode. If only <code>ALL_NON_LOSSY</code> is set, but not <code>ALL_LOSSY</code>, then attempting a conversion that would result in the loss of data (such as <code>INT</code> to <code>TINYINT</code>, or <code>CHAR(25)</code> to <code>VARCHAR(20)</code>) causes the slave to stop with an error.</p>
<code>ALL_LOSSY, ALL_NON_LOSSY</code>	<p>When this mode is set, all supported type conversions are permitted, whether or not they are lossy conversions.</p>
[empty]	<p>When <code>slave_type_conversions</code> is not set, no attribute promotion or demotion is permitted; this means that all columns in the source and target tables must be of the same types.</p> <p>This mode is the default.</p>

Changing the type conversion mode requires restarting the slave with the new `slave_type_conversions` setting.

Supported conversions. Supported conversions between different but similar data types are shown in the following list:

- Between any of the integer types `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, and `BIGINT`.

This includes conversions between the signed and unsigned versions of these types.

Lossy conversions are made by truncating the source value to the maximum (or minimum) permitted by the target column. For insuring non-lossy conversions when going from unsigned to signed types, the target column must be large enough to accommodate the range of values in the source column. For example, you can demote `TINYINT UNSIGNED` non-lossily to `SMALLINT`, but not to `TINYINT`.

- Between any of the decimal types `DECIMAL`, `FLOAT`, `DOUBLE`, and `NUMERIC`.

`FLOAT` to `DOUBLE` is a non-lossy conversion; `DOUBLE` to `FLOAT` can only be handled lossily. A conversion from `DECIMAL(M,D)` to `DECIMAL(M',D')` where $M' \Rightarrow M$ and $D' \Rightarrow D$ is non-lossy; for any case where $M' < M$, $D' < D$, or both, only a lossy conversion can be made.

For any of the decimal types, if a value to be stored cannot be fit in the target type, the value is rounded down according to the rounding rules defined for the server elsewhere in the documentation. See [Section 12.18.4, “Rounding Behavior”](#), for information about how this is done for decimal types.

- Between any of the string types `CHAR`, `VARCHAR`, and `TEXT`, including conversions between different widths.

Conversion of a `CHAR`, `VARCHAR`, or `TEXT` to a `CHAR`, `VARCHAR`, or `TEXT` column the same size or larger is never lossy. Lossy conversion is handled by inserting only the first N characters of the string on the slave, where N is the width of the target column.



Important

Replication between columns using different character sets is not supported.

- Between any of the binary data types `BINARY`, `VARBINARY`, and `BLOB`, including conversions between different widths.

Conversion of a `BINARY`, `VARBINARY`, or `BLOB` to a `BINARY`, `VARBINARY`, or `BLOB` column the same size or larger is never lossy. Lossy conversion is handled by inserting only the first N bytes of the string on the slave, where N is the width of the target column.

- Between any 2 `BIT` columns of any 2 sizes.

When inserting a value from a `BIT(M)` column into a `BIT(M')` column, where $M' > M$, the most significant bits of the `BIT(M')` columns are cleared (set to zero) and the M bits of the `BIT(M)` value are set as the least significant bits of the `BIT(M')` column.

When inserting a value from a source `BIT(M)` column into a target `BIT(M')` column, where $M' < M$, the maximum possible value for the `BIT(M')` column is assigned; in other words, an “all-set” value is assigned to the target column.

Conversions between types not in the previous list are not permitted.

Replication type conversions in MySQL 5.5.3 and earlier. Prior to MySQL 5.5.3, with row-based binary logging, you could not replicate between different `INT` subtypes, such as from `TINYINT` to `BIGINT`,

because changes to columns of these types were represented differently from one another in the binary log when using row-based logging. (However, you could replicate from `BLOB` to `TEXT` using row-based replication because changes to `BLOB` and `TEXT` columns were represented using the same format in the binary log.)

Supported conversions for attribute promotion when using row-based replication prior to MySQL 5.5.3 are shown in the following table:

From (Master)	To (Slave)
<code>BINARY</code>	<code>CHAR</code>
<code>BLOB</code>	<code>TEXT</code>
<code>CHAR</code>	<code>BINARY</code>
<code>DECIMAL</code>	<code>NUMERIC</code>
<code>NUMERIC</code>	<code>DECIMAL</code>
<code>TEXT</code>	<code>BLOB</code>
<code>VARBINARY</code>	<code>VARCHAR</code>
<code>VARCHAR</code>	<code>VARBINARY</code>



Note

In all cases, the size or width of the column on the slave must be equal to or greater than that of the column on the master. For example, you could replicate from a `CHAR(10)` column on the master to a column that used `BINARY(10)` or `BINARY(25)` on the slave, but you could not replicate from a `CHAR(10)` column on the master to `BINARY(5)` column on the slave.

Any unique index (including primary keys) having a prefix must use a prefix of the same length on both master and slave; in such cases, differing prefix lengths are disallowed. It is possible to use a nonunique index whose prefix length differs between master and slave, but this can cause serious performance issues, particularly when the prefix used on the master is longer. This is due to the fact that 2 unique prefixes of a given length may no longer be unique at a shorter length; for example, the words *catalogue* and *catamount* have the 5-character prefixes *catal* and *catam*, respectively, but share the same 4-character prefix (*cata*). This can lead to queries that use such indexes executing less efficiently on the slave, when a shorter prefix is employed in the slave' definition of the same index than on the master.

For `DECIMAL` and `NUMERIC` columns, both the mantissa (*M*) and the number of decimals (*D*) must be the same size or larger on the slave as compared with the master. For example, replication from a `NUMERIC(5,4)` to a `DECIMAL(6,4)` worked, but not from a `NUMERIC(5,4)` to a `DECIMAL(5,3)`.

Prior to MySQL 5.5.3, MySQL replication did not support attribute promotion of any of the following data types to or from any other data type when using row-based replication:

- `INT` (including `TINYINT`, `SMALLINT`, `MEDIUMINT`, `BIGINT`).

Promotion between `INT` subtypes—for example, from `SMALLINT` to `BIGINT`—was also not supported prior to MySQL 5.5.3.

- `SET` or `ENUM`.

- `FLOAT` or `DOUBLE`.
- All of the data types relating to dates, times, or both: `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, and `YEAR`.

16.4.1.9. Replication and `DIRECTORY` Table Options

If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the master server, the table option is also used on the slave. This can cause problems if no corresponding directory exists in the slave host file system or if it exists but is not accessible to the slave server. This can be overridden by using the `NO_DIR_IN_CREATE` [614] server SQL mode on the slave, which causes the slave to ignore the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that `MyISAM` data and index files are created in the table's database directory.

For more information, see [Section 5.1.7, “Server SQL Modes”](#).

16.4.1.10. Replication of Invoked Features

Replication of invoked features such as user-defined functions (UDFs) and stored programs (stored procedures and functions, triggers, and events) provides the following characteristics:

- The effects of the feature are always replicated.
- The following statements are replicated using statement-based replication:
 - `CREATE EVENT`
 - `ALTER EVENT`
 - `DROP EVENT`
 - `CREATE PROCEDURE`
 - `DROP PROCEDURE`
 - `CREATE FUNCTION`
 - `DROP FUNCTION`
 - `CREATE TRIGGER`
 - `DROP TRIGGER`

However, the *effects* of features created, modified, or dropped using these statements are replicated using row-based replication.



Note

Attempting to replicate invoked features using statement-based replication produces the warning `Statement is not safe to log in statement format`. For example, trying to replicate a UDF with statement-based replication generates this warning because it currently cannot be determined by the MySQL server whether the UDF is deterministic. If you are absolutely certain that the invoked feature's effects are deterministic, you can safely disregard such warnings.

- In the case of `CREATE EVENT` and `ALTER EVENT`:

- The status of the event is set to `SLAVESIDE_DISABLED` on the slave regardless of the state specified (this does not apply to `DROP EVENT`).
- The master on which the event was created is identified on the slave by its server ID. The `ORIGINATOR` column in `INFORMATION_SCHEMA.EVENTS` and the `originator` column in `mysql.event` store this information. See [Section 20.7, “The INFORMATION_SCHEMA EVENTS Table”](#), and [Section 13.7.5.19, “SHOW EVENTS Syntax”](#), for more information.
- The feature implementation resides on the slave in a renewable state so that if the master fails, the slave can be used as the master without loss of event processing.

To determine whether there are any scheduled events on a MySQL server that were created on a different server (that was acting as a replication master), query the `INFORMATION_SCHEMA.EVENTS` table in a manner similar to what is shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME
FROM INFORMATION_SCHEMA.EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED';
```

Alternatively, you can use the `SHOW EVENTS` statement, like this:

```
SHOW EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED';
```

When promoting a replication slave having such events to a replication master, you must enable each event using `ALTER EVENT event_name ENABLED`, where `event_name` is the name of the event.

If more than one master was involved in creating events on this slave, and you wish to identify events that were created only on a given master having the server ID `master_id`, modify the previous query on the `EVENTS` table to include the `ORIGINATOR` column, as shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME, ORIGINATOR
FROM INFORMATION_SCHEMA.EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED'
AND ORIGINATOR = 'master_id'
```

You can employ `ORIGINATOR` with the `SHOW EVENTS` statement in a similar fashion:

```
SHOW EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED'
AND ORIGINATOR = 'master_id'
```

Before enabling events that were replicated from the master, you should disable the MySQL Event Scheduler on the slave (using a statement such as `SET GLOBAL event_scheduler = OFF;`), run any necessary `ALTER EVENT` statements, restart the server, then re-enable the Event Scheduler on the slave afterward (using a statement such as `SET GLOBAL event_scheduler = ON;`)-

If you later demote the new master back to being a replication slave, you must disable manually all events enabled by the `ALTER EVENT` statements. You can do this by storing in a separate table the event names from the `SELECT` statement shown previously, or using `ALTER EVENT` statements to rename the events with a common prefix such as `replicated_` to identify them.

If you rename the events, then when demoting this server back to being a replication slave, you can identify the events by querying the `EVENTS` table, as shown here:

```
SELECT CONCAT(EVENT_SCHEMA, '.', EVENT_NAME) AS 'Db.Event'
FROM INFORMATION_SCHEMA.EVENTS
WHERE INSTR(EVENT_NAME, 'replicated_') = 1;
```

16.4.1.11. Replication and Floating-Point Values

With statement-based replication, values are converted from decimal to binary. Because conversions between decimal and binary representations of them may be approximate, comparisons involving floating-point values are inexact. This is true for operations that use floating-point values explicitly, or that use values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on master and slave servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See [Section 12.2, “Type Conversion in Expression Evaluation”](#), and [Section C.5.5.8, “Problems with Floating-Point Values”](#).

16.4.1.12. Replication and `FLUSH`

Some forms of the `FLUSH` statement are not logged because they could cause problems if replicated to a slave: `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK`. For a syntax example, see [Section 13.7.6.3, “FLUSH Syntax”](#). The `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements are written to the binary log and thus replicated to slaves. This is not normally a problem because these statements do not modify table data.

However, this behavior can cause difficulties under certain circumstances. If you replicate the privilege tables in the `mysql` database and update those tables directly without using `GRANT`, you must issue a `FLUSH PRIVILEGES` on the slaves to put the new privileges into effect. In addition, if you use `FLUSH TABLES` when renaming a `MyISAM` table that is part of a `MERGE` table, you must issue `FLUSH TABLES` manually on the slaves. These statements are written to the binary log unless you specify `NO_WRITE_TO_BINLOG` or its alias `LOCAL`.

16.4.1.13. Replication and System Functions

Certain functions do not replicate well under some conditions:

- The `USER()` [1197], `CURRENT_USER()` [1192] (or `CURRENT_USER` [1192]), `UUID()` [1201], `VERSION()` [1198], and `LOAD_FILE()` [1105] functions are replicated without change and thus do not work reliably on the slave unless row-based replication is enabled. (See [Section 16.1.2, “Replication Formats”](#).)

`USER()` [1197] and `CURRENT_USER()` [1192] are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode. (Bug #28086) (See also [Section 16.4.1.6, “Replication of `CURRENT_USER\(\)`”](#).)

Beginning with MySQL 5.5.1, `VERSION()` [1198] is also automatically replicated using row-based replication when using `MIXED` mode, and generates a warning in `STATEMENT` mode. (Bug #47995) Beginning with MySQL 5.5.2, this is also true with regard to the `RAND()` [1128] function. (Bug #49222)

- For `NOW()` [1143], the binary log includes the timestamp. This means that the value *as returned by the call to this function on the master* is replicated to the slave. This can lead to a possibly unexpected result when replicating between MySQL servers in different time zones. Suppose that the master is located in New York, the slave is located in Stockholm, and both servers are using local time. Suppose further that, on the master, you create a table `mytable`, perform an `INSERT` statement on this table, and then select from the table, as shown here:

```
mysql> CREATE TABLE mytable (mycol TEXT);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO mytable VALUES ( NOW() );
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM mytable;
+-----+
| mycol |
+-----+
| 2009-09-01 12:00:00 |
```

```
+-----+
1 row in set (0.00 sec)
```

Local time in Stockholm is 6 hours later than in New York; so, if you issue `SELECT NOW()` on the slave at that exact same instant, the value `2009-09-01 18:00:00` is returned. For this reason, if you select from the slave's copy of `mytable` after the `CREATE TABLE` and `INSERT` statements just shown have been replicated, you might expect `mycol` to contain the value `2009-09-01 18:00:00`. However, this is not the case; when you select from the slave's copy of `mytable`, you obtain exactly the same result as on the master:

```
mysql> SELECT * FROM mytable;
+-----+
| mycol |
+-----+
| 2009-09-01 12:00:00 |
+-----+
1 row in set (0.00 sec)
```

Unlike `NOW()` [1143], the `SYSDATE()` [1146] function is not replication-safe because it is not affected by `SET TIMESTAMP` statements in the binary log and is nondeterministic if statement-based logging is used. This is not a problem if row-based logging is used.

An alternative is to use the `--sysdate-is-now` [465] option to cause `SYSDATE()` [1146] to be an alias for `NOW()` [1143]. This must be done on the master and the slave to work correctly. In such cases, a warning is still issued by this function, but can safely be ignored as long as `--sysdate-is-now` [465] is used on both the master and the slave.

Beginning with MySQL 5.5.1, `SYSDATE()` [1146] is automatically replicated using row-based replication when using `MIXED` mode, and generates a warning in `STATEMENT` mode. (Bug #47995)

See also [Section 16.4.1.30, "Replication and Time Zones"](#).

- *The following restriction applies to statement-based replication only, not to row-based replication.* The `GET_LOCK()` [1199], `RELEASE_LOCK()` [1201], `IS_FREE_LOCK()` [1200], and `IS_USED_LOCK()` [1200] functions that handle user-level locks are replicated without the slave knowing the concurrency context on the master. Therefore, these functions should not be used to insert into a master table because the content on the slave would differ. For example, do not issue a statement such as `INSERT INTO mytable VALUES(GET_LOCK(...))`.

Beginning with MySQL 5.5.1, these functions are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode. (Bug #47995)

As a workaround for the preceding limitations when statement-based replication is in effect, you can use the strategy of saving the problematic function result in a user variable and referring to the variable in a later statement. For example, the following single-row `INSERT` is problematic due to the reference to the `UUID()` [1201] function:

```
INSERT INTO t VALUES(UUID());
```

To work around the problem, do this instead:

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

That sequence of statements replicates because the value of `@my_uuid` is stored in the binary log as a user-variable event prior to the `INSERT` statement and is available for use in the `INSERT`.

The same idea applies to multiple-row inserts, but is more cumbersome to use. For a two-row insert, you can do this:

```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

However, if the number of rows is large or unknown, the workaround is difficult or impracticable. For example, you cannot convert the following statement to one in which a given individual user variable is associated with each row:

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

Within a stored function, `RAND()` [1128] replicates correctly as long as it is invoked only once during the execution of the function. (You can consider the function execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

The `FOUND_ROWS()` [1193] and `ROW_COUNT()` [1196] functions are not replicated reliably using statement-based replication. A workaround is to store the result of the function call in a user variable, and then use that in the `INSERT` statement. For example, if you wish to store the result in a table named `mytable`, you might normally do so like this:

```
SELECT SQL_CALC_FOUND_ROWS FROM mytable LIMIT 1;
INSERT INTO mytable VALUES( FOUND_ROWS() );
```

However, if you are replicating `mytable`, you should use `SELECT ... INTO`, and then store the variable in the table, like this:

```
SELECT SQL_CALC_FOUND_ROWS INTO @found_rows FROM mytable LIMIT 1;
INSERT INTO mytable VALUES(@found_rows);
```

In this way, the user variable is replicated as part of the context, and applied on the slave correctly.

These functions are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode. (Bug #12092, Bug #30244)

16.4.1.14. Replication and `LIMIT`

Statement-based replication of `LIMIT` clauses in `DELETE`, `UPDATE`, and `INSERT ... SELECT` statements is unsafe since the order of the rows affected is not defined. (Such statements can be replicated correctly with statement-based replication only if they also contain an `ORDER BY` clause.) When such a statement is encountered:

- When using `STATEMENT` mode, a warning that the statement is not safe for statement-based replication is now issued.

Currently, when using `STATEMENT` mode, warnings are issued for DML statements containing `LIMIT` even when they also have an `ORDER BY` clause (and so are made deterministic). This is a known issue. (Bug #42851)

- When using `MIXED` mode, the statement is now automatically replicated using row-based mode.

16.4.1.15. Replication and `LOAD DATA INFILE`

The `LOAD DATA INFILE` statement was not always replicated correctly to a slave running MySQL 5.5.0 or earlier from a master running MySQL 4.0 or earlier. When using statement-based replication, the `LOAD DATA INFILE` statement `CONCURRENT` option was not replicated. This issue was fixed in MySQL 5.5.0. This issue does not have any impact on `CONCURRENT` option handling when using row-based replication in MySQL 5.1 or later. (Bug #34628)

In MySQL 5.5.6 and later, `LOAD DATA INFILE` is considered unsafe (see [Section 16.1.2.3, "Determination of Safe and Unsafe Statements in Binary Logging"](#)). It causes a warning when using statement-based logging format, and is logged using row-based format when using mixed-format logging.

16.4.1.16. Replication and the Slow Query Log

Replication slaves do not write replicated queries to the slow query log, even if the same queries were written to the slow query log on the master. This is a known issue. (Bug #23300)

16.4.1.17. Replication and `REPAIR TABLE`

When used on a corrupted or otherwise damaged table, it is possible for the `REPAIR TABLE` statement to delete rows that cannot be recovered. However, any such modifications of table data performed by this statement are not replicated, which can cause master and slave to lose synchronization. For this reason, in the event that a table on the master becomes damaged and you use `REPAIR TABLE` to repair it, you should first stop replication (if it is still running) before using `REPAIR TABLE`, then afterward compare the master's and slave's copies of the table and be prepared to correct any discrepancies manually, before restarting replication.

16.4.1.18. Replication and Master or Slave Shutdowns

It is safe to shut down a master server and restart it later. When a slave loses its connection to the master, the slave tries to reconnect immediately and retries periodically if that fails. The default is to retry every 60 seconds. This may be changed with the `CHANGE MASTER TO` statement. A slave also is able to deal with network connectivity outages. However, the slave notices the network outage only after receiving no data from the master for `slave_net_timeout` [1888] seconds. If your outages are short, you may want to decrease `slave_net_timeout` [1888]. See Section 5.1.4, “Server System Variables”.

An unclean shutdown (for example, a crash) on the master side can result in the master binary log having a final position less than the most recent position read by the slave, due to the master binary log file not being flushed. This can cause the slave not to be able to replicate when the master comes back up. Setting `sync_binlog=1` [1904] in the master `my.cnf` file helps to minimize this problem because it causes the master to flush its binary log more frequently.

Shutting down a slave cleanly is safe because it keeps track of where it left off. However, be careful that the slave does not have temporary tables open; see Section 16.4.1.21, “Replication and Temporary Tables”. Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the problem occurred:

- For transactions, the slave commits and then updates `relay-log.info`. If a crash occurs between these two operations, relay log processing will have proceeded further than the information file indicates and the slave will re-execute the events from the last transaction in the relay log after it has been restarted.
- A similar problem can occur if the slave updates `relay-log.info` but the server host crashes before the write has been flushed to disk. To minimize the chance of this occurring, set `sync_relay_log_info=1` [1891] in the slave `my.cnf` file. The default value of `sync_relay_log_info` [1891] is 0, which does not cause writes to be forced to disk; the server relies on the operating system to flush the file from time to time.

The fault tolerance of your system for these types of problems is greatly increased if you have a good uninterruptible power supply.

16.4.1.19. Replication and `max_allowed_packet`

`max_allowed_packet` [519] sets an upper limit on the size of any single message between the MySQL server and clients, including replication slaves. If you are replicating large column values (such as might be found in `TEXT` or `BLOB` columns) and `max_allowed_packet` [519] is too small on the master, the master fails with an error, and the slave shuts down the I/O thread. If `max_allowed_packet` [519] is too small on the slave, this also causes the slave to stop the I/O thread.

Row-based replication currently sends all columns and column values for updated rows from the master to the slave, including values of columns that were not actually changed by the update. This means that, when you are replicating large column values using row-based replication, you must take care to set `max_allowed_packet` [519] large enough to accommodate the largest row in any table to be replicated, even if you are replicating updates only, or you are inserting only relatively small values.

16.4.1.20. Replication and MEMORY Tables

When a master server shuts down and restarts, its MEMORY tables become empty. To replicate this effect to slaves, the first time that the master uses a given MEMORY table after startup, it logs an event that notifies slaves that the table must to be emptied by writing a `DELETE` statement for that table to the binary log.

When a slave server shuts down and restarts, its MEMORY tables become empty. This causes the slave to be out of synchrony with the master and may lead to other failures or cause the slave to stop:

- Row-format updates and deletes received from the master may fail with `Can't find record in 'memory_table'`.
- Statements such as `INSERT INTO ... SELECT FROM memory_table` may insert a different set of rows on the master and slave.

The safe way to restart a slave that is replicating MEMORY tables is to first drop or delete all rows from the MEMORY tables on the master and wait until those changes have replicated to the slave. Then it is safe to restart the slave.

An alternative restart method may apply in some cases. When `binlog_format=ROW` [1899], you can prevent the slave from stopping if you set `slave_exec_mode=IDEMPOTENT` [1887] before you start the slave again. This allows the slave to continue to replicate, but its MEMORY tables will still be different from those on the master. This can be okay if the application logic is such that the contents of MEMORY tables can be safely lost (for example, if the MEMORY tables are used for caching). `slave_exec_mode=IDEMPOTENT` [1887] applies globally to all tables, so it may hide other replication errors in non-MEMORY tables.

The size of MEMORY tables is limited by the value of the `max_heap_table_size` [521] system variable, which is not replicated (see Section 16.4.1.35, “Replication and Variables”). A change in `max_heap_table_size` takes effect for MEMORY tables that are created or updated using `ALTER TABLE ... ENGINE = MEMORY` or `TRUNCATE TABLE` following the change, or for all MEMORY tables following a server restart. If you increase the value of this variable on the master without doing so on the slave, it becomes possible for a table on the master to grow larger than its counterpart on the slave, leading to inserts that succeed on the master but fail on the slave with `Table is full` errors. This is a known issue (Bug #48666). In such cases, you must set the global value of `max_heap_table_size` [521] on the slave as well as on the master, then restart replication. It is also recommended that you restart both the master and slave MySQL servers, to insure that the new value takes complete (global) effect on each of them.

See Section 14.6, “The MEMORY Storage Engine”, for more information about MEMORY tables.

16.4.1.21. Replication and Temporary Tables

This section does not apply when row-based replication is in use because in that case temporary tables are not replicated. It does apply with mixed-format replication for statements involving temporary tables that can be logged safely using statement-based format. For information about row-based replication and temporary tables, see [RBL, RBR, and temporary tables](#).

Safe slave shutdown when using temporary tables. Temporary tables are replicated except in the case where you stop the slave server (not just the slave threads) and you have replicated temporary

tables that are open for use in updates that have not yet been executed on the slave. If you stop the slave server, the temporary tables needed by those updates are no longer available when the slave is restarted. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use the following procedure:

1. Issue a `STOP SLAVE SQL_THREAD` statement.
2. Use `SHOW STATUS` to check the value of the `Slave_open_temp_tables` [607] variable.
3. If the value is not 0, restart the slave SQL thread with `START SLAVE SQL_THREAD` and repeat the procedure later.
4. When the value is 0, issue a `mysqladmin shutdown` command to stop the slave.

Temporary tables and replication options. By default, all temporary tables are replicated; this happens whether or not there are any matching `--replicate-do-db` [1875], `--replicate-do-table` [1878], or `--replicate-wild-do-table` [1879] options in effect. However, the `--replicate-ignore-table` [1878] and `--replicate-wild-ignore-table` [1880] options are honored for temporary tables.

A recommended practice when using statement-based or mixed-format replication is to designate a prefix for exclusive use in naming temporary tables that you do not want replicated, then employ a `--replicate-wild-ignore-table` [1880] option to match that prefix. For example, you might give all such tables names beginning with `norep` (such as `norepmytable`, `norepyourtable`, and so on), then use `--replicate-wild-ignore-table=norep%` [1880] to prevent them from being replicated.

16.4.1.22. Replication of the `mysql` System Database

Data modification statements made to tables in the `mysql` database are replicated according to the value of `binlog_format` [1899]; if this value is `MIXED`, these statements are replicated using row-based format. However, statements that would normally update this information indirectly—such as `GRANT`, `REVOKE`, and statements manipulating triggers, stored routines, and views—are replicated to slaves using statement-based replication.

16.4.1.23. Replication and the Query Optimizer

It is possible for the data on the master and slave to become different if a statement is written in such a way that the data modification is nondeterministic; that is, left up to the query optimizer. (In general, this is not a good practice, even outside of replication.) Examples of nondeterministic statements include `DELETE` or `UPDATE` statements that use `LIMIT` with no `ORDER BY` clause; see Section 16.4.1.14, “Replication and `LIMIT`”, for a detailed discussion of these.

16.4.1.24. Replication and Reserved Words

You can encounter problems when you attempt to replicate from an older master to a newer slave and you make use of identifiers on the master that are reserved words in the newer MySQL version running on the slave. An example of this is using a table column named `current_user` on a 4.0 master that is replicating to a 4.1 or higher slave because `CURRENT_USER` is a reserved word beginning in MySQL 4.1. Replication can fail in such cases with Error 1064 *You have an error in your SQL syntax..., even if a database or table named using the reserved word or a table having a column named using the reserved word is excluded from replication*. This is due to the fact that each SQL event must be parsed by the slave prior to execution, so that the slave knows which database object or objects would be affected; only after the event is parsed can the slave apply any filtering rules defined by `--replicate-do-db` [1875], `--replicate-do-table` [1878], `--replicate-ignore-db` [1877], and `--replicate-ignore-table` [1878].

To work around the problem of database, table, or column names on the master which would be regarded as reserved words by the slave, do one of the following:

- Use one or more `ALTER TABLE` statements on the master to change the names of any database objects where these names would be considered reserved words on the slave, and change any SQL statements that use the old names to use the new names instead.
- In any SQL statements using these database object names, write the names as quoted identifiers using backtick characters (```).

For listings of reserved words by MySQL version, see [Reserved Words](#), in the *MySQL Server Version Reference*. For identifier quoting rules, see [Section 9.2, “Schema Object Names”](#).

16.4.1.25. `SET PASSWORD` and Row-Based Replication

Row-based replication of `SET PASSWORD` statements from a MySQL 5.1 master to a MySQL 5.5 slave did not work correctly prior to MySQL 5.1.53 on the master and MySQL 5.5.7 on the slave (see Bug #57098, Bug #57357).

16.4.1.26. Slave Errors During Replication

If a statement produces the same error (identical error code) on both the master and the slave, the error is logged, but replication continues.

If a statement produces different errors on the master and the slave, the slave SQL thread terminates, and the slave writes a message to its error log and waits for the database administrator to decide what to do about the error. This includes the case that a statement produces an error on the master or the slave, but not both. To address the issue, connect to the slave manually and determine the cause of the problem. `SHOW SLAVE STATUS` is useful for this. Then fix the problem and run `START SLAVE`. For example, you might need to create a nonexistent table before you can start the slave again.

If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the `--slave-skip-errors [1883]` option.

For nontransactional storage engines such as `MyISAM`, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the master, the slave expects execution of the statement to result in the same error code. If it does not, the slave SQL thread stops as described previously.

If you are replicating between tables that use different storage engines on the master and slave, keep in mind that the same statement might produce a different error when run against one version of the table, but not the other, or might cause an error for one version of the table, but not the other. For example, since `MyISAM` ignores foreign key constraints, an `INSERT` or `UPDATE` statement accessing an `InnoDB` table on the master might cause a foreign key violation but the same statement performed on a `MyISAM` version of the same table on the slave would produce no such error, causing replication to stop.

16.4.1.27. Replication and Server SQL Mode

Using different server SQL mode settings on the master and the slave may cause the same `INSERT` statements to be handled differently on the master and the slave, leading the master and slave to diverge. For best results, you should always use the same server SQL mode on the master and on the slave. This advice applies whether you are using statement-based or row-based replication.

If you are replicating partitioned tables, using different SQL modes on the master and the slave is likely to cause issues. At a minimum, this is likely to cause the distribution of data among partitions to be different

in the master's and slave's copies of a given table. It may also cause inserts into partitioned tables that succeed on the master to fail on the slave.

For more information, see [Section 5.1.7, “Server SQL Modes”](#).

16.4.1.28. Replication Retries and Timeouts

The global system variable `slave_transaction_retries` [1889] affects replication as follows: If the slave SQL thread fails to execute a transaction because of an InnoDB deadlock or because it exceeded the InnoDB `innodb_lock_wait_timeout` [1560] value, or the `NDBCLUSTER TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout` value, the slave automatically retries the transaction `slave_transaction_retries` [1889] times before stopping with an error. The default value is 10. The total retry count can be seen in the output of `SHOW STATUS`; see [Section 5.1.6, “Server Status Variables”](#).

16.4.1.29. Replication and `TIMESTAMP`

Older versions of MySQL (prior to 4.1) differed significantly in several ways in their handling of the `TIMESTAMP` data type from what is supported in MySQL versions 5.5 and newer; these include syntax extensions which are deprecated in MySQL 5.1, and that no longer supported in MySQL 5.5. This can cause problems (including replication failures) when replicating between MySQL Server versions, if you are using columns that are defined using the old `TIMESTAMP(N)` syntax. See [Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”](#), for more information about the differences, how they can impact MySQL replication, and what you can do if you encounter such problems.

16.4.1.30. Replication and Time Zones

The same system time zone should be set for both master and slave. Otherwise, statements depending on the local time on the master are not replicated properly, such as statements that use the `NOW()` [1143] or `FROM_UNIXTIME()` [1140] functions. You can set the time zone in which MySQL server runs by using the `--timezone=timezone_name` [258] option of the `mysqld_safe` script or by setting the `TZ` environment variable. See also [Section 16.4.1.13, “Replication and System Functions”](#).

If the master is MySQL 4.1 or earlier, both master and slave should also use the same default connection time zone. That is, the `--default-time-zone` [441] parameter should have the same value for both master and slave.

`CONVERT_TZ(..., ..., @@session.time_zone)` [1134] is properly replicated only if both master and slave are running MySQL 5.0.4 or newer.

16.4.1.31. Replication and Transactions

Mixing transactional and nontransactional statements within the same transaction. In general, you should avoid transactions that update both transactional and nontransactional tables in a replication environment. You should also avoid using any statement that accesses both transactional (or temporary) and nontransactional tables and writes to any of them.

As of MySQL 5.5.2, the server uses these rules for binary logging:

- If the initial statements in a transaction are nontransactional, they are written to the binary log immediately. The remaining statements in the transaction are cached and not written to the binary log until the transaction is committed. (If the transaction is rolled back, the cached statements are written to the binary log only if they make nontransactional changes that cannot be rolled back. Otherwise, they are discarded.)
- For statement-based logging, logging of nontransactional statements is affected by the `binlog_direct_non_transactional_updates` [1898] system variable. When this variable is

`OFF` (the default), logging is as just described. When this variable is `ON`, logging occurs immediately for nontransactional statements occurring anywhere in the transaction (not just initial nontransactional statements). Other statements are kept in the transaction cache and logged when the transaction commits. `binlog_direct_non_transactional_updates` [1898] has no effect for row-format or mixed-format binary logging.

Transactional, nontransactional, and mixed statements.

To apply those rules, the server considers a statement nontransactional if it changes only nontransactional tables, and transactional if it changes only transactional tables. Prior to MySQL 5.5.6, a statement that changed both nontransactional and transactional tables was considered “mixed”. Beginning with MySQL 5.5.6, a statement that references both nontransactional and transactional tables and updates *any* of the tables involved, is considered a mixed statement. Mixed statements, like transactional statements, are cached and logged when the transaction commits.

Beginning with MySQL 5.5.6, a mixed statement that updates a transactional table is considered unsafe if the statement also performs either of the following actions:

- Updates or reads a transactional table
- Reads a nontransactional table and the transaction isolation level is less than `REPEATABLE_READ`

Also beginning with MySQL 5.5.6, any mixed statement following the update of a transactional table within a transaction is considered unsafe if it performs either of the following actions:

- Updates any table and reads from any temporary table
- Updates a nontransactional table and `binlog_direct_non_trans_update` is `OFF`

For more information, see [Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”](#).



Note

A mixed statement is unrelated to mixed binary logging format.

Before MySQL 5.5.2, the rules for binary logging are similar to those just described, except that there is no `binlog_direct_non_transactional_updates` [1898] system variable to affect logging of transactional statements. Thus, the server immediately logs only the initial nontransactional statements in a transaction and caches the rest until commit time.

In situations where transactions mix updates to transactional and nontransactional tables, the order of statements in the binary log is correct, and all needed statements are written to the binary log even in case of a `ROLLBACK`. However, when a second connection updates the nontransactional table before the first connection transaction is complete, statements can be logged out of order because the second connection update is written immediately after it is performed, regardless of the state of the transaction being performed by the first connection.

Using different storage engines on master and slave. It is possible to replicate transactional tables on the master using nontransactional tables on the slave. For example, you can replicate an `InnoDB` master table as a `MyISAM` slave table. However, if you do this, there are problems if the slave is stopped in the middle of a `BEGIN ... COMMIT` block because the slave restarts at the beginning of the `BEGIN` block.

Beginning with MySQL 5.5.0, it is also safe to replicate transactions from `MyISAM` tables on the master to transactional tables—such as tables that use the `InnoDB` storage engine—on the slave. In such cases (beginning with MySQL 5.5.0), an `AUTOCOMMIT=1` [484] statement issued on the master is replicated, thus enforcing `AUTOCOMMIT` mode on the slave.

When the storage engine type of the slave is nontransactional, transactions on the master that mix updates of transactional and nontransactional tables should be avoided because they can cause inconsistency of the data between the master transactional table and the slave nontransactional table. That is, such transactions can lead to master storage engine-specific behavior with the possible effect of replication going out of synchrony. MySQL does not issue a warning about this currently, so extra care should be taken when replicating transactional tables from the master to nontransactional tables on the slaves.

Changing the binary logging format within transactions. Beginning with MySQL 5.5.3, the `binlog_format` [1899] system variable is read-only as long as a transaction is in progress. (Bug #47863)

Every transaction (including `autocommit` [484] transactions) is recorded in the binary log as though it starts with a `BEGIN` statement, and ends with either a `COMMIT` or a `ROLLBACK` statement. In MySQL 5.5, this true is even for statements affecting tables that use a nontransactional storage engine (such as `MyISAM`).

16.4.1.32. Replication and Triggers

With statement-based replication, triggers executed on the master also execute on the slave. With row-based replication, triggers executed on the master do not execute on the slave. Instead, the row changes on the master resulting from trigger execution are replicated and applied on the slave.

This behavior is by design. If under row-based replication the slave applied the triggers as well as the row changes caused by them, the changes would in effect be applied twice on the slave, leading to different data on the master and the slave.

If you want triggers to execute on both the master and the slave—perhaps because you have different triggers on the master and slave—you must use statement-based replication. However, to enable slave-side triggers, it is not necessary to use statement-based replication exclusively. It is sufficient to switch to statement-based replication only for those statements where you want this effect, and to use row-based replication the rest of the time.

A statement invoking a trigger (or function) that causes an update to an `AUTO_INCREMENT` column is not replicated correctly using statement-based replication. MySQL 5.5 marks such statements as unsafe. (Bug #45677)

16.4.1.33. Replication and Views

Views are always replicated to slaves. Views are filtered by their own name, not by the tables they refer to. This means that a view can be replicated to the slave even if the view contains a table that would normally be filtered out by `replication-ignore-table` rules. Care should therefore be taken to ensure that views do not replicate table data that would normally be filtered for security reasons.

16.4.1.34. Replication and `TRUNCATE TABLE`

`TRUNCATE TABLE` is normally regarded as a DML statement, and so would be expected to be logged and replicated using row-based format when the binary logging mode is `ROW` or `MIXED`. However this caused issues when logging or replicating, in `STATEMENT` or `MIXED` mode, tables that used transactional storage engines such as `InnoDB` when the transaction isolation level was `READ COMMITTED` or `READ UNCOMMITTED`, which precludes statement-based logging.

`TRUNCATE TABLE` is treated for purposes of logging and replication as DDL rather than DML so that it can be logged and replicated as a statement. However, the effects of the statement as applicable to `InnoDB` and other transactional tables on replication slaves still follow the rules described in [Section 13.1.33](#), “`TRUNCATE TABLE Syntax`” governing such tables. (Bug #36763)

16.4.1.35. Replication and Variables

System variables are not replicated correctly when using `STATEMENT` mode, except for the following variables when they are used with session scope:

- `auto_increment_increment` [1865]
- `auto_increment_offset` [1868]
- `character_set_client` [486]
- `character_set_connection` [487]
- `character_set_database` [487]
- `character_set_server` [488]
- `collation_connection` [489]
- `collation_database` [489]
- `collation_server` [489]
- `foreign_key_checks` [500]
- `identity` [505]
- `last_insert_id` [511]
- `lc_time_names` [512]
- `pseudo_thread_id` [539]
- `sql_auto_is_null` [554]
- `time_zone` [567]
- `timestamp` [567]
- `unique_checks` [571]

When `MIXED` mode is used, the variables in the preceding list, when used with session scope, cause a switch from statement-based to row-based logging. See [Section 5.2.4.3, “Mixed Binary Logging Format”](#).

`sql_mode` [556] is also replicated except for the `NO_DIR_IN_CREATE` [614] mode; the slave always preserves its own value for `NO_DIR_IN_CREATE` [614], regardless of changes to it on the master. This is true for all replication formats.

However, when `mysqlbinlog` parses a `SET @@sql_mode = mode` statement, the full `mode` value, including `NO_DIR_IN_CREATE` [614], is passed to the receiving server. For this reason, replication of such a statement may not be safe when `STATEMENT` mode is in use.

The `default_storage_engine` [494] and `storage_engine` [559] system variables are not replicated, regardless of the logging mode; this is intended to facilitate replication between different storage engines.

The `read_only` [544] system variable is not replicated. In addition, the enabling this variable has different effects with regard to temporary tables, table locking, and the `SET PASSWORD` statement in different MySQL versions.

The `max_heap_table_size` [521] system variable is not replicated. Increasing the value of this variable on the master without doing so on the slave can lead eventually to `Table is full` errors on the slave when trying to execute `INSERT` statements on a `MEMORY` table on the master that is thus permitted to grow larger than its counterpart on the slave. For more information, see [Section 16.4.1.20, “Replication and MEMORY Tables”](#).

In statement-based replication, session variables are not replicated properly when used in statements that update tables. For example, the following sequence of statements will not insert the same data on the master and the slave:

```
SET max_join_size=1000;
INSERT INTO mytable VALUES(@@max_join_size);
```

This does not apply to the common sequence:

```
SET time_zone=...;
INSERT INTO mytable VALUES(CONVERT_TZ(..., ..., @@time_zone));
```

Replication of session variables is not a problem when row-based replication is being used, in which case, session variables are always replicated safely. See [Section 16.1.2, “Replication Formats”](#).

In MySQL 5.5, the following session variables are written to the binary log and honored by the replication slave when parsing the binary log, regardless of the logging format:

- `sql_mode` [556]
- `foreign_key_checks` [500]
- `unique_checks` [571]
- `character_set_client` [486]
- `collation_connection` [489]
- `collation_database` [489]
- `collation_server` [489]
- `sql_auto_is_null` [554]



Important

Even though session variables relating to character sets and collations are written to the binary log, replication between different character sets is not supported.

It is strongly recommended that you always use the same setting for the `lower_case_table_names` [518] system variable on both master and slave. In particular, when a case-sensitive file system is used, setting this variable to 1 on the slave, but to a different value on the master, can cause two types of problems: Names of databases are not converted to lowercase; in addition, when using row-based replication names of tables are also not converted. Either of these problems can cause replication to fail. This is a known issue, which is fixed in MySQL 5.6.

16.4.2. Replication Compatibility Between MySQL Versions

MySQL supports replication from one major version to the next higher major version. For example, you can replicate from a master running MySQL 4.1 to a slave running MySQL 5.0, from a master running MySQL 5.0 to a slave running MySQL 5.1, and so on.

However, one may encounter difficulties when replicating from an older master to a newer slave if the master uses statements or relies on behavior no longer supported in the version of MySQL used on the slave. For example, in MySQL 5.5, `CREATE TABLE ... SELECT` statements are permitted to change tables other than the one being created, but are no longer allowed to do so in MySQL 5.6 (see [Section 16.4.1.4, “Replication of `CREATE TABLE ... SELECT` Statements”](#)).

The use of more than 2 MySQL Server versions is not supported in replication setups involving multiple masters, regardless of the number of master or slave MySQL servers. This restriction applies not only to major versions, but to minor versions within the same major version as well. For example, if you are using a chained or circular replication setup, you cannot use MySQL 5.5.1, MySQL 5.5.2, and MySQL 5.5.4 concurrently, although you could use any 2 of these releases together.

In some cases, it is also possible to replicate between a master and a slave that is more than one major version newer than the master. However, there are known issues with trying to replicate from a master running MySQL 4.1 or earlier to a slave running MySQL 5.1 or later. To work around such problems, you can insert a MySQL server running an intermediate version between the two; for example, rather than replicating directly from a MySQL 4.1 master to a MySQL 5.1 slave, it is possible to replicate from a MySQL 4.1 server to a MySQL 5.0 server, and then from the MySQL 5.0 server to a MySQL 5.1 server.



Important

It is strongly recommended to use the most recent release available within a given MySQL major version because replication (and other) capabilities are continually being improved. It is also recommended to upgrade masters and slaves that use early releases of a major version of MySQL to GA (production) releases when the latter become available for that major version.

Replication from newer masters to older slaves may be possible, but is generally not supported. This is due to a number of factors:

- **Binary log format changes.** The binary log format can change between major releases. While we attempt to maintain backward compatibility, this is not always possible. For example, the binary log format implemented in MySQL 5.0 changed considerably from that used in previous versions, especially with regard to handling of character sets, `LOAD DATA INFILE`, and time zones. This means that replication from a MySQL 5.0 (or later) master to a MySQL 4.1 (or earlier) slave is generally not supported.

This also has significant implications for upgrading replication servers; see [Section 16.4.3, “Upgrading a Replication Setup”](#), for more information.

- **Use of row-based replication.** Row-based replication was implemented in MySQL 5.1.5, so you cannot replicate using row-based replication from any MySQL 5.5 or later master to a slave older than MySQL 5.1.5.

For more information about row-based replication, see [Section 16.1.2, “Replication Formats”](#).

- **SQL incompatibilities.** You cannot replicate from a newer master to an older slave using statement-based replication if the statements to be replicated use SQL features available on the master but not on the slave.

However, if both the master and the slave support row-based replication, and there are no data definition statements to be replicated that depend on SQL features found on the master but not on the slave, you can use row-based replication to replicate the effects of data modification statements even if the DDL run on the master is not supported on the slave.

For more information on potential replication issues, see [Section 16.4.1, “Replication Features and Issues”](#).

16.4.3. Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading.

This section applies to upgrading replication from older versions of MySQL to MySQL 5.5. A 4.0 server should be 4.0.3 or newer.

When you upgrade a master to 5.5 from an earlier MySQL release series, you should first ensure that all the slaves of this master are using the same 5.5.x release. If this is not the case, you should first upgrade the slaves. To upgrade each slave, shut it down, upgrade it to the appropriate 5.5.x version, restart it, and restart replication. The 5.5 slave is able to read the old relay logs written prior to the upgrade and to execute the statements they contain. Relay logs created by the slave after the upgrade are in 5.5 format.

After the slaves have been upgraded, shut down the master, upgrade it to the same 5.5.x release as the slaves, and restart it. The 5.5 master is able to read the old binary logs written prior to the upgrade and to send them to the 5.5 slaves. The slaves recognize the old format and handle it properly. Binary logs created by the master subsequent to the upgrade are in 5.5 format. These too are recognized by the 5.5 slaves.

In other words, when upgrading to MySQL 5.5, the slaves must be MySQL 5.5 before you can upgrade the master to 5.5. Note that downgrading from 5.5 to older versions does not work so simply: You must ensure that any 5.5 binary log or relay log has been fully processed, so that you can remove it before proceeding with the downgrade.

Downgrading a replication setup to a previous version cannot be done once you have switched from statement-based to row-based replication, and after the first row-based statement has been written to the binlog. See [Section 16.1.2, “Replication Formats”](#).

Some upgrades may require that you drop and re-create database objects when you move from one MySQL series to the next. For example, collation changes might require that table indexes be rebuilt. Such operations, if necessary, will be detailed at [Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”](#). It is safest to perform these operations separately on the slaves and the master, and to disable replication of these operations from the master to the slave. To achieve this, use the following procedure:

1. Stop all the slaves and upgrade them. Restart them with the `--skip-slave-start` [\[1882\]](#) option so that they do not connect to the master. Perform any table repair or rebuilding operations needed to re-create database objects, such as use of `REPAIR TABLE` or `ALTER TABLE`, or dumping and reloading tables or triggers.
2. Disable the binary log on the master. To do this without restarting the master, execute a `SET sql_log_bin = 0` statement. Alternatively, stop the master and restart it without the `--log-bin` [\[1892\]](#) option. If you restart the master, you might also want to disallow client connections. For example, if all clients connect using TCP/IP, use the `--skip-networking` [\[461\]](#) option when you restart the master.
3. With the binary log disabled, perform any table repair or rebuilding operations needed to re-create database objects. The binary log must be disabled during this step to prevent these operations from being logged and sent to the slaves later.
4. Re-enable the binary log on the master. If you set `sql_log_bin` [\[555\]](#) to 0 earlier, execute a `SET sql_log_bin = 1` statement. If you restarted the master to disable the binary log, restart it with `--log-bin` [\[1892\]](#), and without `--skip-networking` [\[461\]](#) so that clients and slaves can connect.
5. Restart the slaves, this time without the `--skip-slave-start` [\[1882\]](#) option.

16.4.4. Troubleshooting Replication

If you have followed the instructions but your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the master has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. If logging is enabled, `Position` is nonzero. If binary logging is not enabled, verify that you are running the master with the `--log-bin` [1892] option.
- Verify that the master and slave both were started with the `--server-id` [1858] option and that the ID value is unique on each server.
- Verify that the slave is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the slave server. For example, `--skip-slave-start` [1882] prevents the slave threads from starting until you issue a `START SLAVE` statement.
- If the slave is running, check whether it established a connection to the master. Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see what they display. See Section 16.2.1, “Replication Implementation Details”. If the I/O thread state says `Connecting to master`, check the following:
 - Verify the privileges for the user being used for replication on the master.
 - Check that the host name of the master is correct and that you are using the correct port to connect to the master. The port used for replication is the same as used for client network communication (the default is 3306). For the host name, ensure that the name resolves to the correct IP address.
 - Check that networking has not been disabled on the master or slave. Look for the `skip-networking` [461] option in the configuration file. If present, comment it out or remove it.
 - If the master has a firewall or IP filtering configuration, ensure that the network port being used for MySQL is not being filtered.
 - Check that you can reach the master by using `ping` or `traceroute/tracert` to reach the host.
- If the slave was running previously but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modified the data on the slave outside of the slave thread. If the slave stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in Section 16.4.1, “Replication Features and Issues”. If it is a bug, see Section 16.4.5, “How to Report Replication Bugs or Problems”, for instructions on how to report it.
- If a statement that succeeded on the master refuses to run on the slave, try the following procedure if it is not feasible to do a full database resynchronization by deleting the slave’s databases and copying a new snapshot from the master:
 1. Determine whether the affected table on the slave is different from the master table. Try to understand how this happened. Then make the slave’s table identical to the master’s and run `START SLAVE`.
 2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.

3. If you decide that the slave can skip the next statement from the master, issue the following statements:

```
mysql> SET GLOBAL sql_slave_skip_counter = N;
mysql> START SLAVE;
```

The value of *N* should be 1 if the next statement from the master does not use `AUTO_INCREMENT` or `LAST_INSERT_ID()` [1194]. Otherwise, the value should be 2. The reason for using a value of 2 for statements that use `AUTO_INCREMENT` or `LAST_INSERT_ID()` [1194] is that they take two events in the binary log of the master.

See also [Section 13.4.2.4](#), “`SET GLOBAL sql_slave_skip_counter Syntax`”.

4. If you are sure that the slave started out perfectly synchronized with the master, and that no one has updated the tables involved outside of the slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

16.4.5. How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in [Section 1.7](#), “[How to Report Bugs or Problems](#)”. If you have a “phantom” problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the slave thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*
2. Run the slave with the `--log-slave-updates` [1870] and `--log-bin` [1892] options. These options cause the slave to log the updates that it receives from the master into its own binary logs.
3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:
 - All binary log files from the master
 - All binary log files from the slave
 - The output of `SHOW MASTER STATUS` from the master at the time you discovered the problem
 - The output of `SHOW SLAVE STATUS` from the slave at the time you discovered the problem
 - Error logs from the master and the slave
4. Use `mysqlbinlog` to examine the binary logs. The following should be helpful to find the problem statement. `log_file` and `log_pos` are the `Master_Log_File` and `Read_Master_Log_Pos` values from `SHOW SLAVE STATUS`.

```
shell> mysqlbinlog --start-position=log_pos log_file | head
```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at [Section 1.7, "How to Report Bugs or Problems"](#).

Chapter 17. MySQL Cluster NDB 7.2

Table of Contents

17.1. MySQL Cluster Overview	1970
17.1.1. MySQL Cluster Core Concepts	1972
17.1.2. MySQL Cluster Nodes, Node Groups, Replicas, and Partitions	1975
17.1.3. MySQL Cluster Hardware, Software, and Networking Requirements	1977
17.1.4. MySQL Cluster Development History	1979
17.1.5. MySQL Server using InnoDB Compared with MySQL Cluster	1981
17.1.6. Known Limitations of MySQL Cluster	1986
17.2. MySQL Cluster Installation	1997
17.2.1. Installing MySQL Cluster on Linux	2000
17.2.2. Installing MySQL Cluster on Windows	2005
17.2.3. Initial Configuration of MySQL Cluster	2014
17.2.4. Initial Startup of MySQL Cluster	2016
17.2.5. MySQL Cluster Example with Tables and Data	2017
17.2.6. Safe Shutdown and Restart of MySQL Cluster	2021
17.2.7. Upgrading and Downgrading MySQL Cluster NDB 7.2	2022
17.3. MySQL Cluster Configuration	2022
17.3.1. Quick Test Setup of MySQL Cluster	2023
17.3.2. MySQL Cluster Configuration Files	2025
17.3.3. Overview of MySQL Cluster Configuration Parameters	2104
17.3.4. MySQL Server Options and Variables for MySQL Cluster	2126
17.3.5. Using High-Speed Interconnects with MySQL Cluster	2174
17.4. MySQL Cluster Programs	2176
17.4.1. ndbd — The MySQL Cluster Data Node Daemon	2176
17.4.2. ndbinfo_select_all — Select From ndbinfo Tables	2183
17.4.3. ndbmt — The MySQL Cluster Data Node Daemon (Multi-Threaded)	2184
17.4.4. ndb_mgmd — The MySQL Cluster Management Server Daemon	2186
17.4.5. ndb_mgm — The MySQL Cluster Management Client	2194
17.4.6. ndb_config — Extract MySQL Cluster Configuration Information	2195
17.4.7. ndb_cpctest — Automate Testing for NDB Development	2203
17.4.8. ndb_delete_all — Delete All Rows from an NDB Table	2203
17.4.9. ndb_desc — Describe NDB Tables	2204
17.4.10. ndb_drop_index — Drop Index from an NDB Table	2208
17.4.11. ndb_drop_table — Drop an NDB Table	2209
17.4.12. ndb_error_reporter — NDB Error-Reporting Utility	2209
17.4.13. ndb_index_stat — NDB Index Statistics Utility	2210
17.4.14. ndb_print_backup_file — Print NDB Backup File Contents	2215
17.4.15. ndb_print_schema_file — Print NDB Schema File Contents	2215
17.4.16. ndb_print_sys_file — Print NDB System File Contents	2215
17.4.17. ndbd_redo_log_reader — Check and Print Content of Cluster Redo Log	2216
17.4.18. ndb_restore — Restore a MySQL Cluster Backup	2217
17.4.19. ndb_select_all — Print Rows from an NDB Table	2229
17.4.20. ndb_select_count — Print Row Counts for NDB Tables	2232
17.4.21. ndb_show_tables — Display List of NDB Tables	2233
17.4.22. ndb_size.pl — NDBCLUSTER Size Requirement Estimator	2234
17.4.23. ndb_waiter — Wait for MySQL Cluster to Reach a Given Status	2237
17.4.24. Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs	2240

17.5. Management of MySQL Cluster	2243
17.5.1. Summary of MySQL Cluster Start Phases	2244
17.5.2. Commands in the MySQL Cluster Management Client	2246
17.5.3. Online Backup of MySQL Cluster	2249
17.5.4. MySQL Server Usage for MySQL Cluster	2254
17.5.5. Performing a Rolling Restart of a MySQL Cluster	2255
17.5.6. Event Reports Generated in MySQL Cluster	2258
17.5.7. MySQL Cluster Log Messages	2267
17.5.8. MySQL Cluster Single User Mode	2278
17.5.9. Quick Reference: MySQL Cluster SQL Statements	2279
17.5.10. The <code>ndbinfo</code> MySQL Cluster Information Database	2282
17.5.11. MySQL Cluster Security Issues	2302
17.5.12. MySQL Cluster Disk Data Tables	2310
17.5.13. Adding MySQL Cluster Data Nodes Online	2316
17.5.14. Distributed MySQL Privileges for MySQL Cluster	2327
17.5.15. NDB API Statistics Counters and Variables	2330
17.6. MySQL Cluster Replication	2342
17.6.1. MySQL Cluster Replication: Abbreviations and Symbols	2343
17.6.2. General Requirements for MySQL Cluster Replication	2344
17.6.3. Known Issues in MySQL Cluster Replication	2345
17.6.4. MySQL Cluster Replication Schema and Tables	2352
17.6.5. Preparing the MySQL Cluster for Replication	2355
17.6.6. Starting MySQL Cluster Replication (Single Replication Channel)	2356
17.6.7. Using Two Replication Channels for MySQL Cluster Replication	2358
17.6.8. Implementing Failover with MySQL Cluster Replication	2359
17.6.9. MySQL Cluster Backups With MySQL Cluster Replication	2360
17.6.10. MySQL Cluster Replication: Multi-Master and Circular Replication	2366
17.6.11. MySQL Cluster Replication Conflict Resolution	2371
17.7. MySQL Cluster Release Notes	2380

This chapter contains information about *MySQL Cluster*, which is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. Recent releases of MySQL Cluster use version 7 of the `NDBCLUSTER` storage engine (also known as `NDB`) to enable running several computers with MySQL servers and other software in a cluster; the latest releases available for production use incorporate `NDB` version 7.2.

Support for the `NDBCLUSTER` storage engine is not included in the standard MySQL Server 5.5 binaries built by Oracle. Instead, users of MySQL Cluster binaries from Oracle should upgrade to the most recent binary release of MySQL Cluster for supported platforms—these include RPMs that should work with most Linux distributions. MySQL Cluster users who build from source should use the sources provided for MySQL Cluster. (Locations where the sources can be obtained are listed later in this section.)

This chapter contains information about MySQL Cluster NDB 7.2 releases through 5.5.29-ndb-7.2.10. Currently, the MySQL Cluster NDB 7.2 release series is Generally Available (GA), as is MySQL Cluster NDB 7.1. MySQL Cluster NDB 7.0 and MySQL Cluster NDB 6.3 are previous GA release series; although they are still supported, we recommend that new deployments use MySQL Cluster NDB 7.2. For information about MySQL Cluster NDB 7.1, MySQL Cluster NDB 7.0, and previous versions of MySQL Cluster, see [MySQL Cluster NDB 6.1 - 7.1](#), in the *MySQL 5.1 Manual*.

Supported Platforms. MySQL Cluster is currently available and supported on a number of platforms. For exact levels of support available for on specific combinations of operating system versions, operating system distributions, and hardware platforms, please refer to <http://www.mysql.com/support/supportedplatforms/cluster.html>.

Availability. MySQL Cluster binary and source packages are available for supported platforms from <http://dev.mysql.com/downloads/cluster/>.

MySQL Cluster release numbers. MySQL Cluster follows a somewhat different release pattern from the mainline MySQL Server 5.5 series of releases. In this *Manual* and other MySQL documentation, we identify these and later MySQL Cluster releases employing a version number that begins with “NDB”. This version number is that of the `NDBCLUSTER` storage engine used in the release, and not of the MySQL server version on which the MySQL Cluster release is based.

Version strings used in MySQL Cluster software. The version string displayed by MySQL Cluster programs uses this format:

```
mysql-mysql_server_version-ndb-ndb_engine_version
```

`mysql_server_version` represents the version of the MySQL Server on which the MySQL Cluster release is based. For all MySQL Cluster NDB 6.x and 7.x releases, this is “5.1”. `ndb_engine_version` is the version of the `NDB` storage engine used by this release of the MySQL Cluster software. You can see this format used in the `mysql` client, as shown here:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.29-ndb-7.2.10 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT VERSION()\G
***** 1. row *****
VERSION(): 5.5.29-ndb-7.2.10
1 row in set (0.00 sec)
```

This version string is also displayed in the output of the `SHOW` command in the `ndb_mgm` client:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[nbdb(NDB)]      2 node(s)
id=1      @10.0.10.6  (5.5.29-ndb-7.2.10, Nodegroup: 0, Master)
id=2      @10.0.10.8  (5.5.29-ndb-7.2.10, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=3      @10.0.10.2  (5.5.29-ndb-7.2.10)

[mysqld(API)]   2 node(s)
id=4      @10.0.10.10 (5.5.29-ndb-7.2.10)
id=5      (not connected, accepting connect from any host)
```

The version string identifies the mainline MySQL version from which the MySQL Cluster release was branched and the version of the `NDBCLUSTER` storage engine used. For example, the full version string for MySQL Cluster NDB 7.2.4 (the first MySQL Cluster production release based on MySQL Server 5.5) is `mysql-5.5.19-ndb-7.2.4`. From this we can determine the following:

- Since the portion of the version string preceding “-ndb-” is the base MySQL Server version, this means that MySQL Cluster NDB 7.2.4 derives from the MySQL 5.5.19, and contains all feature enhancements and bugfixes from MySQL 5.5 up to and including MySQL 5.5.19.
- Since the portion of the version string following “-ndb-” represents the version number of the `NDB` (or `NDBCLUSTER`) storage engine, MySQL Cluster NDB 7.2.4 uses version 7.2.4 of the `NDBCLUSTER` storage engine.

New MySQL Cluster releases are numbered according to updates in the [NDB](#) storage engine, and do not necessarily correspond in a one-to-one fashion with mainline MySQL Server releases. For example, MySQL Cluster NDB 7.2.4 (as previously noted) is based on MySQL 5.5.19, while MySQL Cluster NDB 7.2.0 was based on MySQL 5.1.51 (version string: `mysql-5.1.51-ndb-7.2.0`).

Compatibility with standard MySQL 5.5 releases. While many standard MySQL schemas and applications can work using MySQL Cluster, it is also true that unmodified applications and database schemas may be slightly incompatible or have suboptimal performance when run using MySQL Cluster (see [Section 17.1.6, “Known Limitations of MySQL Cluster”](#)). Most of these issues can be overcome, but this also means that you are very unlikely to be able to switch an existing application datastore—that currently uses, for example, [MyISAM](#) or [InnoDB](#)—to use the [NDB](#) storage engine without allowing for the possibility of changes in schemas, queries, and applications. In addition, the MySQL Server and MySQL Cluster codebases diverge considerably, so that the standard `mysqld` cannot function as a drop-in replacement for the version of `mysqld` supplied with MySQL Cluster.

MySQL Cluster development source trees. MySQL Cluster development trees can also be accessed from <https://code.launchpad.net/~mysql/>:

The MySQL Cluster development sources maintained at <https://code.launchpad.net/~mysql/> are licensed under the GPL. For information about obtaining MySQL sources using Bazaar and building them yourself, see [Section 2.10.3, “Installing MySQL from a Development Source Tree”](#).

**Note**

As with MySQL Server 5.5, MySQL Cluster NDB 7.2 is built using [CMake](#).

Currently, MySQL Cluster NDB 7.0, MySQL Cluster NDB 7.1, and MySQL Cluster NDB 7.2 releases are all Generally Available (GA), although we recommend that new deployments use MySQL Cluster NDB 7.2. MySQL Cluster NDB 6.1, MySQL Cluster NDB 6.2, and MySQL Cluster NDB 6.3, are no longer in active development. For an overview of major features added in MySQL Cluster NDB 7.2, see [Section 17.1.4, “MySQL Cluster Development History”](#). For an overview of major features added in past MySQL Cluster releases through MySQL Cluster NDB 7.1, see [MySQL Cluster Development History](#).

This chapter represents a work in progress, and its contents are subject to revision as MySQL Cluster continues to evolve. Additional information regarding MySQL Cluster can be found on the MySQL Web site at <http://www.mysql.com/products/cluster/>.

Additional Resources. More information about MySQL Cluster can be found in the following places:

- For answers to some commonly asked questions about MySQL Cluster, see [Section B.10, “MySQL FAQ: MySQL 5.5 and MySQL Cluster”](#).
- The MySQL Cluster mailing list: <http://lists.mysql.com/cluster>.
- The MySQL Cluster Forum: <http://forums.mysql.com/list.php?25>.
- Many MySQL Cluster users and developers blog about their experiences with MySQL Cluster, and make feeds of these available through [PlanetMySQL](#).

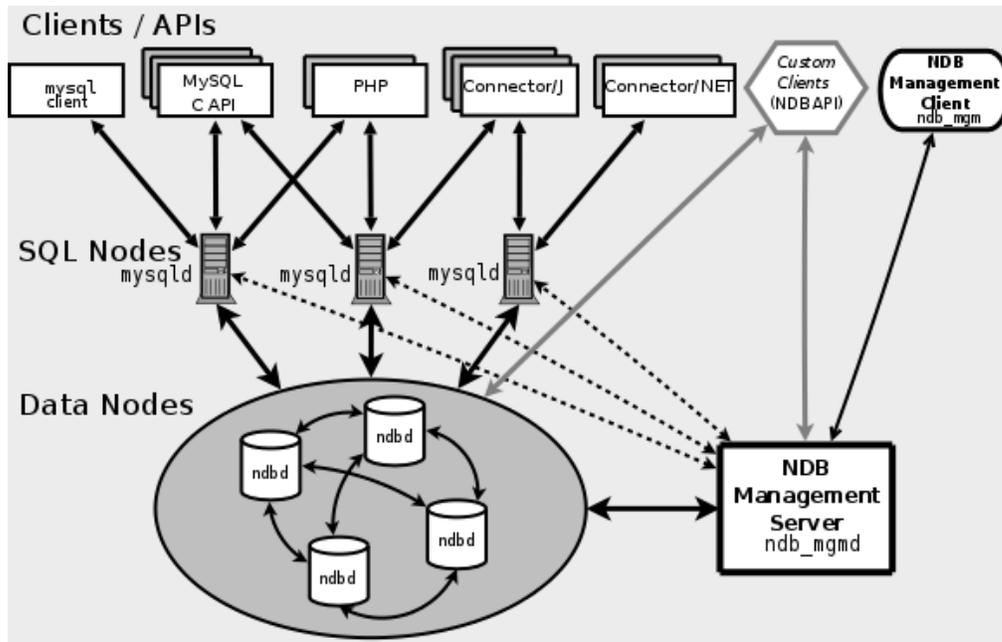
17.1. MySQL Cluster Overview

MySQL Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture enables the system to work with very inexpensive hardware, and with a minimum of specific requirements for hardware or software.

MySQL Cluster is designed not to have any single point of failure. In a shared-nothing system, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as network shares, network file systems, and SANs is not recommended or supported.

MySQL Cluster integrates the standard MySQL server with an in-memory clustered storage engine called **NDB** (which stands for “*Network DataBase*”). In our documentation, the term **NDB** refers to the part of the setup that is specific to the storage engine, whereas “MySQL Cluster” refers to the combination of one or more MySQL servers with the **NDB** storage engine.

A MySQL Cluster consists of a set of computers, known as *hosts*, each running one or more processes. These processes, known as *nodes*, may include MySQL servers (for access to NDB data), data nodes (for storage of the data), one or more management servers, and possibly other specialized data access programs. The relationship of these components in a MySQL Cluster is shown here:



All these programs work together to form a MySQL Cluster (see [Section 17.4, “MySQL Cluster Programs”](#)). When data is stored by the **NDB** storage engine, the tables (and table data) are stored in the data nodes. Such tables are directly accessible from all other MySQL servers (SQL nodes) in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

Although a MySQL Cluster SQL node uses the `mysqld` server daemon, it differs in a number of critical respects from the `mysqld` binary supplied with the MySQL 5.5 distributions, and the two versions of `mysqld` are not interchangeable.

In addition, a MySQL server that is not connected to a MySQL Cluster cannot use the **NDB** storage engine and cannot access any MySQL Cluster data.

The data stored in the data nodes for MySQL Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

Individual nodes can be stopped and restarted, and can then rejoin the system (cluster). Rolling restarts (in which all nodes are restarted in turn) are used in making configuration changes and software upgrades (see [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#)). Rolling restarts are also used as

part of the process of adding new data nodes online (see [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#)). For more information about data nodes, how they are organized in a MySQL Cluster, and how they handle and store MySQL Cluster data, see [Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#).

Backing up and restoring MySQL Cluster databases can be done using the NDB-native functionality found in the MySQL Cluster management client and the `ndb_restore` program included in the MySQL Cluster distribution. For more information, see [Section 17.5.3, “Online Backup of MySQL Cluster”](#), and [Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”](#). You can also use the standard MySQL functionality provided for this purpose in `mysqldump` and the MySQL server. See [Section 4.5.4, “mysqldump — A Database Backup Program”](#), for more information.

MySQL Cluster nodes can use a number of different transport mechanisms for inter-node communications, including TCP/IP using standard 100 Mbps or faster Ethernet hardware. It is also possible to use the high-speed *Scalable Coherent Interface* (SCI) protocol with MySQL Cluster, although this is not required to use MySQL Cluster. SCI requires special hardware and software; see [Section 17.3.5, “Using High-Speed Interconnects with MySQL Cluster”](#), for more about SCI and using it with MySQL Cluster.

17.1.1. MySQL Cluster Core Concepts

`NDBCLUSTER` (also known as `NDB`) is an in-memory storage engine offering high-availability and data-persistence features.

The `NDBCLUSTER` storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. MySQL Cluster's `NDB` storage engine contains a complete set of data, dependent only on other data within the cluster itself.

The “Cluster” portion of MySQL Cluster is configured independently of the MySQL servers. In a MySQL Cluster, each part of the cluster is considered to be a *node*.



Note

In many contexts, the term “node” is used to indicate a computer, but when discussing MySQL Cluster it means a *process*. It is possible to run multiple nodes on a single computer; for a computer on which one or more cluster nodes are being run we use the term *cluster host*.

There are three types of cluster nodes, and in a minimal MySQL Cluster configuration, there will be at least three nodes, one of each of these types:

- *Management node*: The role of this type of node is to manage the other nodes within the MySQL Cluster, performing such functions as providing configuration data, starting and stopping nodes, running backup, and so forth. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. An MGM node is started with the command `ndb_mgmd`.
- *Data node*: This type of node stores cluster data. There are as many data nodes as there are replicas, times the number of fragments (see [Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#)). For example, with two replicas, each having two fragments, you need four data nodes. One replica is sufficient for data storage, but provides no redundancy; therefore, it is recommended to have 2 (or more) replicas to provide redundancy, and thus high availability. A data node is started with the command `ndbd` (see [Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”](#)) or `ndbmtdd` (see [Section 17.4.3, “ndbmtdd — The MySQL Cluster Data Node Daemon \(Multi-Threaded\)”](#)).

MySQL Cluster tables are normally stored completely in memory rather than on disk (this is why we refer to MySQL Cluster as an *in-memory* database). However, some MySQL Cluster data can be stored on disk; see [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#), for more information.

- *SQL node*: This is a node that accesses the cluster data. In the case of MySQL Cluster, an SQL node is a traditional MySQL server that uses the `NDBCLUSTER` storage engine. An SQL node is a `mysqld` process started with the `--ndbcluster` [2142] and `--ndb-connectstring` options, which are explained elsewhere in this chapter, possibly with additional MySQL server options as well.

An SQL node is actually just a specialized type of *API node*, which designates any application which accesses MySQL Cluster data. Another example of an API node is the `ndb_restore` utility that is used to restore a cluster backup. It is possible to write such applications using the NDB API. For basic information about the NDB API, see [Getting Started with the NDB API](#).



Important

It is not realistic to expect to employ a three-node setup in a production environment. Such a configuration provides no redundancy; to benefit from MySQL Cluster's high-availability features, you must use multiple data and SQL nodes. The use of multiple management nodes is also highly recommended.

For a brief introduction to the relationships between nodes, node groups, replicas, and partitions in MySQL Cluster, see [Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#).

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. MySQL Cluster is currently designed with the intention that data nodes are homogeneous in terms of processor power, memory space, and bandwidth. In addition, to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the data nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These include standard MySQL clients, NDB-specific API programs, and management clients. These are described in the next few paragraphs.

Standard MySQL clients. MySQL Cluster can be used with existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on. Such client applications send SQL statements to and receive responses from MySQL servers acting as MySQL Cluster SQL nodes in much the same way that they interact with standalone MySQL servers.

MySQL clients using a MySQL Cluster as a data source can be modified to take advantage of the ability to connect with multiple MySQL servers to achieve load balancing and failover. For example, Java clients using Connector/J 5.0.6 and later can use `jdbc:mysql:loadbalance://` URLs (improved in Connector/J 5.1.7) to achieve load balancing transparently; for more information about using Connector/J with MySQL Cluster, see [Using Connector/J with MySQL Cluster](#).

NDB client programs. Client programs can be written that access MySQL Cluster data directly from the `NDBCLUSTER` storage engine, bypassing any MySQL Servers that may be connected to the cluster, using the *NDB API*, a high-level C++ API. Such applications may be useful for specialized purposes where an SQL interface to the data is not needed. For more information, see [The NDB API](#).

NDB-specific Java applications can also be written for MySQL Cluster using the *MySQL Cluster Connector for Java*. This MySQL Cluster Connector includes *ClusterJ*, a high-level database API similar to object-relational mapping persistence frameworks such as Hibernate and JPA that connect directly to

[NDBCLUSTER](#), and so does not require access to a MySQL Server. Support is also provided in MySQL Cluster NDB 7.1 and later for *ClusterJPA*, an OpenJPA implementation for MySQL Cluster that leverages the strengths of ClusterJ and JDBC; ID lookups and other fast operations are performed using ClusterJ (bypassing the MySQL Server), while more complex queries that can benefit from MySQL's query optimizer are sent through the MySQL Server, using JDBC. See [Java and MySQL Cluster](#), and [The ClusterJ API and Data Object Model](#), for more information.

The Memcache API for MySQL Cluster, implemented as the loadable *ndbmemcache* storage engine for memcached version 1.6 and later, is available beginning with MySQL Cluster NDB 7.2.2. This API can be used to provide a persistent MySQL Cluster data store, accessed using the memcache protocol.

The standard `memcached` caching engine is included in the MySQL Cluster NDB 7.2 distribution (7.2.2 and later). Each `memcached` server has direct access to data stored in MySQL Cluster, but is also able to cache data locally and to serve (some) requests from this local cache.

For more information, see [ndbmemcache—Memcache API for MySQL Cluster](#).

Management clients. These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on. An example of this type of program is the `ndb_mgm` management client supplied with MySQL Cluster (see [Section 17.4.5, “ndb_mgm — The MySQL Cluster Management Client”](#)). Such applications can be written using the *MGM API*, a C-language API that communicates directly with one or more MySQL Cluster management servers. For more information, see [The MGM API](#).

Oracle also makes available MySQL Cluster Manager, which provides an advanced command-line interface simplifying many complex MySQL Cluster management tasks, such as restarting a MySQL Cluster with a large number of nodes. The MySQL Cluster Manager client also supports commands for getting and setting the values of most node configuration parameters as well as `mysqld` server options and variables relating to MySQL Cluster. MySQL Cluster Manager 1.1 provides support for adding data nodes online. See the [MySQL Cluster Manager User Manual](#), for more information.

Event logs. MySQL Cluster logs events by category (startup, shutdown, errors, checkpoints, and so on), priority, and severity. A complete listing of all reportable events may be found in [Section 17.5.6, “Event Reports Generated in MySQL Cluster”](#). Event logs are of the two types listed here:

- *Cluster log*: Keeps a record of all desired reportable events for the cluster as a whole.
- *Node log*: A separate log which is also kept for each individual node.

**Note**

Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be consulted only for application development and debugging purposes.

Checkpoint. Generally speaking, when data is saved to disk, it is said that a *checkpoint* has been reached. More specific to MySQL Cluster, a checkpoint is a point in time where all committed transactions are stored on disk. With regard to the [NDB](#) storage engine, there are two types of checkpoints which work together to ensure that a consistent view of the cluster's data is maintained. These are shown in the following list:

- *Local Checkpoint (LCP)*: This is a checkpoint that is specific to a single node; however, LCP's take place for all nodes in the cluster more or less concurrently. An LCP involves saving all of a node's data to disk, and so usually occurs every few minutes. The precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.

- *Global Checkpoint (GCP)*: A GCP occurs every few seconds, when transactions for all nodes are synchronized and the redo-log is flushed to disk.

17.1.2. MySQL Cluster Nodes, Node Groups, Replicas, and Partitions

This section discusses the manner in which MySQL Cluster divides and duplicates data for storage.

A number of concepts central to an understanding of this topic are discussed in the next few paragraphs.

(Data) Node. An `ndbd` process, which stores a *replica*—that is, a copy of the *partition* (see below) assigned to the node group of which the node is a member.

Each data node should be located on a separate computer. While it is also possible to host multiple `ndbd` processes on a single computer, such a configuration is not supported.

It is common for the terms “node” and “data node” to be used interchangeably when referring to an `ndbd` process; where mentioned, management nodes (`ndb_mgmd` processes) and SQL nodes (`mysqld` processes) are specified as such in this discussion.

Node Group. A node group consists of one or more nodes, and stores partitions, or sets of *replicas* (see next item).

The number of node groups in a MySQL Cluster is not directly configurable; it is a function of the number of data nodes and of the number of replicas (`NoOfReplicas` [2041] configuration parameter), as shown here:

```
[number_of_node_groups] = number_of_data_nodes / NoOfReplicas
```

Thus, a MySQL Cluster with 4 data nodes has 4 node groups if `NoOfReplicas` [2041] is set to 1 in the `config.ini` file, 2 node groups if `NoOfReplicas` [2041] is set to 2, and 1 node group if `NoOfReplicas` [2041] is set to 4. Replicas are discussed later in this section; for more information about `NoOfReplicas` [2041], see [Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”](#).



Note

All node groups in a MySQL Cluster must have the same number of data nodes.

You can add new node groups (and thus new data nodes) online, to a running MySQL Cluster; see [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#), for more information.

Partition. This is a portion of the data stored by the cluster. There are as many cluster partitions as nodes participating in the cluster. Each node is responsible for keeping at least one copy of any partitions assigned to it (that is, at least one replica) available to the cluster.

A replica belongs entirely to a single node; a node can (and usually does) store several replicas.

NDB and user-defined partitioning. MySQL Cluster normally partitions `NDBCLUSTER` tables automatically. However, in MySQL 5.1 and later MySQL Cluster releases, it is possible to employ user-defined partitioning with `NDBCLUSTER` tables. This is subject to the following limitations:

1. Only `KEY` and `LINEAR KEY` partitioning schemes can be used with `NDBCLUSTER` tables.
2. When using `ndbd`, the maximum number of partitions that may be defined explicitly for any `NDBCLUSTER` table is $8 * [number\ of\ node\ groups]$. (The number of node groups in a MySQL Cluster is determined as discussed previously in this section.)

When using `ndbmt`, this maximum is also affected by the number of local query handler threads, which is determined by the value of the `MaxNoOfExecutionThreads` [2079] configuration

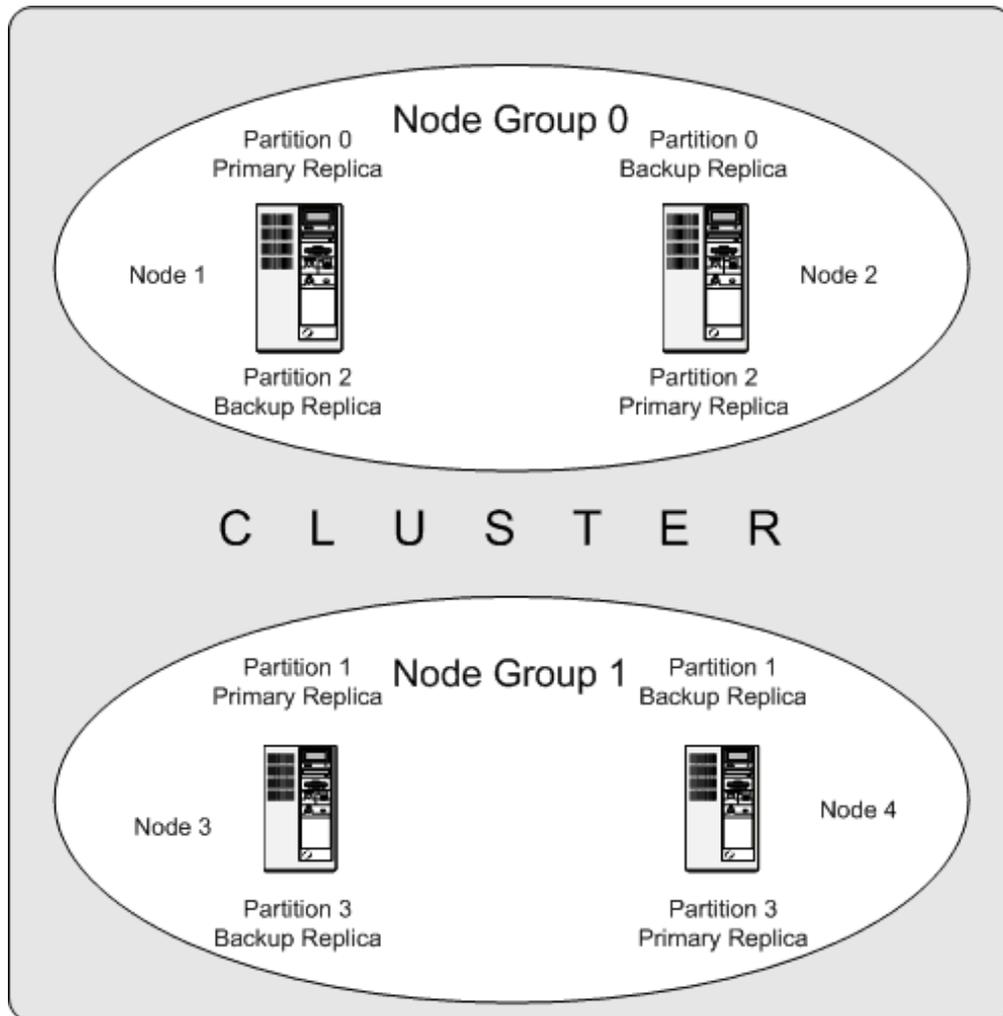
parameter. In such cases, the maximum number of partitions that may be defined explicitly for an `NDB` table is equal to $4 * \text{MaxNoOfExecutionThreads} * [\text{number of node groups}]$.

See [Section 17.4.3, “ndbmtbd — The MySQL Cluster Data Node Daemon \(Multi-Threaded\)”](#), for more information.

For more information relating to MySQL Cluster and user-defined partitioning, see [Section 17.1.6, “Known Limitations of MySQL Cluster”](#), and [Section 18.5.2, “Partitioning Limitations Relating to Storage Engines”](#).

Replica. This is a copy of a cluster partition. Each node in a node group stores a replica. Also sometimes known as a *partition replica*. The number of replicas is equal to the number of nodes per node group.

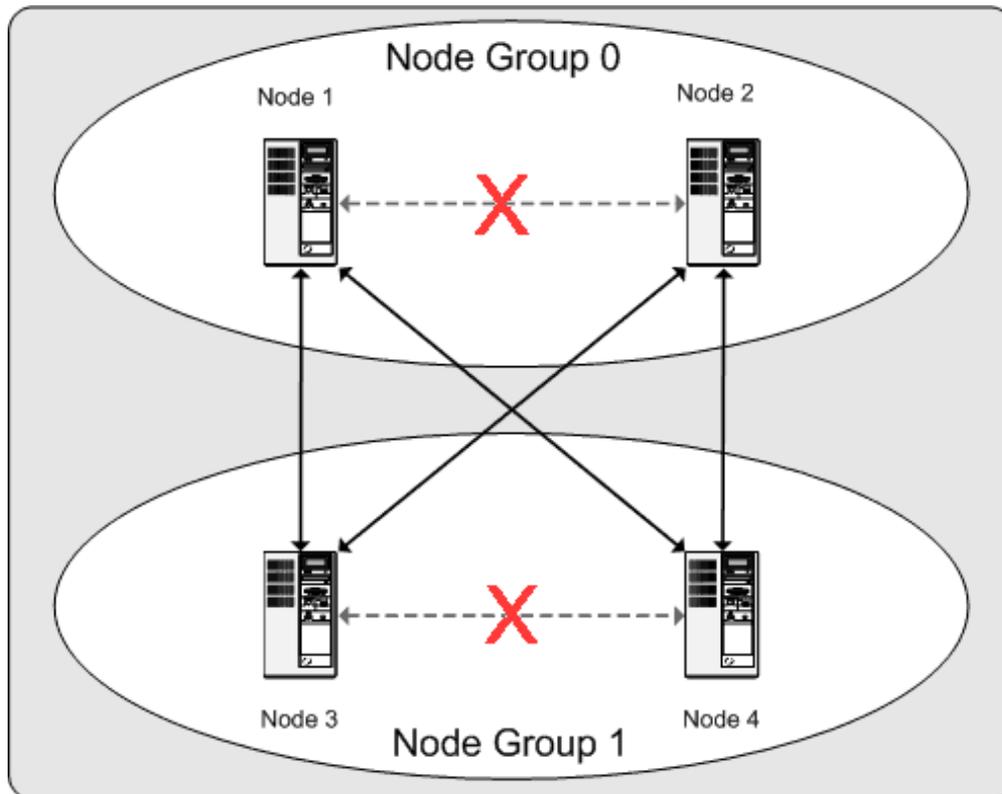
The following diagram illustrates a MySQL Cluster with four data nodes, arranged in two node groups of two nodes each; nodes 1 and 2 belong to node group 0, and nodes 3 and 4 belong to node group 1. Note that only data (`ndbd`) nodes are shown here; although a working cluster requires an `ndb_mgm` process for cluster management and at least one SQL node to access the data stored by the cluster, these have been omitted in the figure for clarity.



The data stored by the cluster is divided into four partitions, numbered 0, 1, 2, and 3. Each partition is stored—in multiple copies—on the same node group. Partitions are stored on alternate node groups as follows:

- Partition 0 is stored on node group 0; a *primary replica* (primary copy) is stored on node 1, and a *backup replica* (backup copy of the partition) is stored on node 2.
- Partition 1 is stored on the other node group (node group 1); this partition's primary replica is on node 3, and its backup replica is on node 4.
- Partition 2 is stored on node group 0. However, the placing of its two replicas is reversed from that of Partition 0; for Partition 2, the primary replica is stored on node 2, and the backup on node 1.
- Partition 3 is stored on node group 1, and the placement of its two replicas are reversed from those of partition 1. That is, its primary replica is located on node 4, with the backup on node 3.

What this means regarding the continued operation of a MySQL Cluster is this: so long as each node group participating in the cluster has at least one node operating, the cluster has a complete copy of all data and remains viable. This is illustrated in the next diagram.



In this example, where the cluster consists of two node groups of two nodes each, any combination of at least one node in node group 0 and at least one node in node group 1 is sufficient to keep the cluster “alive” (indicated by arrows in the diagram). However, if *both* nodes from *either* node group fail, the remaining two nodes are not sufficient (shown by the arrows marked out with an **X**); in either case, the cluster has lost an entire partition and so can no longer provide access to a complete set of all cluster data.

17.1.3. MySQL Cluster Hardware, Software, and Networking Requirements

One of the strengths of MySQL Cluster is that it can be run on commodity hardware and has no unusual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (It is possible to reduce this requirement using Disk Data tables—see [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#), for more information about these.) Naturally, multiple and faster CPUs can enhance performance. Memory requirements for other MySQL Cluster processes are relatively small.

The software requirements for MySQL Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support MySQL Cluster. For supported operating systems, a standard installation should be sufficient. The MySQL software requirements are simple: all that is needed is a production release of MySQL Cluster. It is not strictly necessary to compile MySQL yourself merely to be able to use MySQL Cluster. We assume that you are using the binaries appropriate to your platform, available from the MySQL Cluster software downloads page at <http://dev.mysql.com/downloads/cluster/>.

For communication between nodes, MySQL Cluster supports TCP/IP networking in any standard topology, and the minimum expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or router to provide network connectivity for the cluster as a whole. We strongly recommend that a MySQL Cluster be run on its own subnet which is not shared with machines not forming part of the cluster for the following reasons:

- **Security.** Communications between MySQL Cluster nodes are not encrypted or shielded in any way. The only means of protecting transmissions within a MySQL Cluster is to run your MySQL Cluster on a protected network. If you intend to use MySQL Cluster for Web applications, the cluster should definitely reside behind your firewall and not in your network's De-Militarized Zone (DMZ) or elsewhere.

See [Section 17.5.11.1, "MySQL Cluster Security and Networking Issues"](#), for more information.

- **Efficiency.** Setting up a MySQL Cluster on a private or protected network enables the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch for your MySQL Cluster not only helps protect against unauthorized access to MySQL Cluster data, it also ensures that MySQL Cluster nodes are shielded from interference caused by transmissions between other computers on the network. For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

Network communication and latency. MySQL Cluster requires communication between data nodes and API nodes (including SQL nodes), as well as between data nodes and other data nodes, to execute queries and updates. Communication latency between these processes can directly affect the observed performance and latency of user queries. In addition, to maintain consistency and service despite the silent failure of nodes, MySQL Cluster uses heartbeating and timeout mechanisms which treat an extended loss of communication from a node as node failure. This can lead to reduced redundancy. Recall that, to maintain data consistency, a MySQL Cluster shuts down when the last node in a node group fails. Thus, to avoid increasing the risk of a forced shutdown, breaks in communication between nodes should be avoided wherever possible.

The failure of a data or API node results in the abort of all uncommitted transactions involving the failed node. Data node recovery requires synchronization of the failed node's data from a surviving data node, and re-establishment of disk-based redo and checkpoint logs, before the data node returns to service. This recovery can take some time, during which the Cluster operates with reduced redundancy.

Heartbeating relies on timely generation of heartbeat signals by all nodes. This may not be possible if the node is overloaded, has insufficient machine CPU due to sharing with other programs, or is experiencing delays due to swapping. If heartbeat generation is sufficiently delayed, other nodes treat the node that is slow to respond as failed.

This treatment of a slow node as a failed one may or may not be desirable in some circumstances, depending on the impact of the node's slowed operation on the rest of the cluster. When setting timeout values such as `HeartbeatIntervalDbDb` [2062] and `HeartbeatIntervalDbApi` [2062] for MySQL Cluster, care must be taken to achieve quick detection, failover, and return to service, while avoiding potentially expensive false positives.

Where communication latencies between data nodes are expected to be higher than would be expected in a LAN environment (on the order of 100 μ s), timeout parameters must be increased to ensure that any

allowed periods of latency periods are well within configured timeouts. Increasing timeouts in this way has a corresponding effect on the worst-case time to detect failure and therefore time to service recovery.

LAN environments can typically be configured with stable low latency, and such that they can provide redundancy with fast failover. Individual link failures can be recovered from with minimal and controlled latency visible at the TCP level (where MySQL Cluster normally operates). WAN environments may offer a range of latencies, as well as redundancy with slower failover times. Individual link failures may require route changes to propagate before end-to-end connectivity is restored. At the TCP level this can appear as large latencies on individual channels. The worst-case observed TCP latency in these scenarios is related to the worst-case time for the IP layer to reroute around the failures.

SCI support. It is also possible to use the high-speed Scalable Coherent Interface (SCI) with MySQL Cluster, but this is not a requirement. See [Section 17.3.5, “Using High-Speed Interconnects with MySQL Cluster”](#), for more about this protocol and its use with MySQL Cluster.

17.1.4. MySQL Cluster Development History

In this section, we discuss changes in the implementation of MySQL Cluster in MySQL MySQL Cluster NDB 7.2, as compared to MySQL Cluster NDB 7.1 and earlier releases. Changes and features most likely to be of interest are shown in the following table:

MySQL Cluster NDB 7.2
MySQL Cluster NDB 7.2.1 and later MySQL Cluster NDB 7.2 releases are based on MySQL 5.5. For more information about new features in MySQL Server 5.5, see Section 1.4, “What Is New in MySQL 5.5” .
Version 2 binary log row events, to provide support for improvements in MySQL Cluster Replication conflict detection (see next item). A given <code>mysqld</code> can be made to use Version 1 or Version 2 binary logging row events with the <code>--log-bin-use-v1-row-events [1893]</code> option.
Two new “primary wins” conflict detection and resolution functions <code>NDB\$EPOCH()</code> and <code>NDB\$EPOCH_TRANS()</code> for use in replication setups with 2 MySQL Clusters. For more information, see Section 17.6, “MySQL Cluster Replication” .
Distribution of MySQL users and privileges across MySQL Cluster SQL nodes is now supported—see Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster” .
Improved support for distributed pushed-down joins, which greatly improve performance for many joins that can be executed in parallel on the data nodes.
Default values for a number of data node configuration parameters such as <code>HeartbeatIntervalDbDb [2062]</code> and <code>ArbitrationTimeout [2068]</code> have been improved.
Support for the Memcache API using the loadable <code>ndbmemcache</code> storage engine. See ndbmemcache—Memcache API for MySQL Cluster .

This section contains information about MySQL Cluster NDB 7.2 releases through 5.5.29-ndb-7.2.10, which is currently available for use in production beginning with MySQL Cluster NDB 7.2.4. MySQL Cluster NDB 7.1, MySQL Cluster NDB 7.0, and MySQL Cluster NDB 6.3 are previous GA release series; although these are still supported, we recommend that new deployments use MySQL Cluster NDB 7.2. For information about MySQL Cluster NDB 7.1 and previous releases, see [MySQL Cluster NDB 6.1 - 7.1](#), in the *MySQL 5.1 Manual*.

17.1.4.1. MySQL Cluster Development in MySQL Cluster NDB 7.2

The following improvements to MySQL Cluster have been made in MySQL Cluster NDB 7.2:

- **Based on MySQL Server 5.5.** Previous MySQL Cluster release series, including MySQL Cluster NDB 7.1, used MySQL 5.1 as a base. Beginning with MySQL Cluster NDB 7.2.1, MySQL Cluster

NDB 7.2 is based on MySQL Server 5.5, so that MySQL Cluster users can benefit from MySQL 5.5's improvements in scalability and performance monitoring. As with MySQL 5.5, MySQL Cluster NDB 7.2.1 and later use `CMake` for configuring and building from source in place of GNU autotools (used in MySQL 5.1 and MySQL Cluster releases based on MySQL 5.1). For more information about changes and improvements in MySQL 5.5, see [Section 1.4, "What Is New in MySQL 5.5"](#).

- **Conflict detection using GCI Reflection.** MySQL Cluster Replication implements a new "primary wins" conflict detection and resolution mechanism. *GCI Reflection* applies in two-cluster circulation "active-active" replication setups, tracking the order in which changes are applied on the MySQL Cluster designated as primary relative to changes originating on the other MySQL Cluster (referred to as the secondary). This relative ordering is used to determine whether changes originating on the slave are concurrent with any changes that originate locally, and are therefore potentially in conflict. Two new conflict detection functions are added: When using `NDB$EPOCH()`, rows that are out of sync on the secondary are realigned with those on the primary; with `NDB$EPOCH_TRANS()`, this realignment is applied to transactions. For more information, see [Section 17.6.11, "MySQL Cluster Replication Conflict Resolution"](#).
- **Version 2 binary log row events.** A new format for binary log row events, known as Version 2 binary log row events, provides support for improvements in MySQL Cluster Replication conflict detection (see previous item) and is intended to facilitate further improvements in MySQL Replication. You can cause a given `mysqld` use Version 1 or Version 2 binary logging row events with the `--log-bin-use-v1-row-events` [\[1893\]](#) option. For backwards compatibility, Version 2 binary log row events are also available in MySQL Cluster NDB 7.0 (7.0.27 and later) and MySQL Cluster NDB 7.1 (7.1.16 and later). However, MySQL Cluster NDB 7.0 and MySQL Cluster NDB 7.1 continue to use Version 1 binary log row events as the default, whereas the default in MySQL Cluster NDB 7.2.1 and later is use Version 2 row events for binary logging.
- **Distribution of MySQL users and privileges.** Automatic distribution of MySQL users and privileges across all SQL nodes in a given MySQL Cluster is now supported. To enable this support, you must first import an SQL script `share/mysql/ndb_dist_priv.sql` that is included with the MySQL Cluster NDB 7.2 distribution. This script creates several stored procedures which you can use to enable privilege distribution and perform related tasks.

When a new MySQL Server joins a MySQL Cluster where privilege distribution is in effect, it also participates in the privilege distribution automatically.

Once privilege distribution is enabled, all changes to the grant tables made on any `mysqld` attached to the cluster are immediately available on any other attached MySQL Servers. This is true whether the changes are made using `CREATE USER`, `GRANT`, or any of the other statements described elsewhere in this Manual (see [Section 13.7.1, "Account Management Statements"](#).) This includes privileges relating to stored routines and views; however, automatic distribution of the views or stored routines themselves is not currently supported.

For more information, see [Section 17.5.14, "Distributed MySQL Privileges for MySQL Cluster"](#).

- **Distributed pushed-down joins.** Many joins can now be pushed down to the NDB kernel for processing on MySQL Cluster data nodes. Previously, a join was handled in MySQL Cluster by means of repeated accesses of `NDB` by the SQL node; however, when pushed-down joins are enabled, a pushable join is sent in its entirety to the data nodes, where it can be distributed among the data nodes and executed in parallel on multiple copies of the data, with a single, merged result being returned to `mysqld`. This can reduce greatly the number of round trips between an SQL node and the data nodes required to handle such a join, leading to greatly improved performance of join processing.

It is possible to determine when joins can be pushed down to the data nodes by examining the join with `EXPLAIN`. A number of new system status variables (`Ndb_pushed_queries_defined` [\[2173\]](#), `Ndb_pushed_queries_dropped` [\[2173\]](#), `Ndb_pushed_queries_executed` [\[2174\]](#), and

`Ndb_pushed_reads` [2174]) and additions to the `counters` table (in the `ndbinfo` information database) can also be helpful in determining when and how well joins are being pushed down.

More information and examples are available in the description of the `ndb_join_pushdown` [2151] server system variable. See also the description of the status variables referenced in the previous paragraph, as well as [Section 17.5.10.7, “The `ndbinfo` counters Table”](#).

- **Improved default values for data node configuration parameters.** In order to provide more resiliency to environmental issues and better handling of some potential failure scenarios, and to perform more reliably with increases in memory and other resource requirements brought about by recent improvements in join handling by NDB, the default values for a number of MySQL Cluster data node configuration parameters have been changed. The parameters and changes are described in the following list:

- `HeartbeatIntervalDbDb` [2062]: Default increased from 1500 ms to 5000 ms.
- `ArbitrationTimeout` [2068]: Default increased from 3000 ms to 7500 ms.
- `TimeBetweenEpochsTimeout` [2065]: Now effectively disabled by default (default changed from 4000 ms to 0).
- `SharedGlobalMemory` [2083]: Default increased from 20 MB to 128 MB.
- `MaxParallelScansPerFragment` [2051]: Default increased from 32 to 256.
- `CrashOnCorruptedTuple` [2058] changed from `FALSE` to `TRUE`.
- Beginning with MySQL Cluster NDB 7.2.10, `DefaultOperationRedoProblemAction` [2094] changed from `ABORT` to `QUEUE`.

In addition, the value computed for `MaxNoOfLocalScans` [2050] when this parameter is not set in `config.ini` has been increased by a factor of 4.

- **Fail-fast data nodes.** Beginning with MySQL Cluster NDB 7.2.1, data nodes handle corrupted tuples in a fail-fast manner by default. This is a change from previous versions of MySQL Cluster where this behavior had to be enabled explicitly by enabling the `CrashOnCorruptedTuple` [2058] configuration parameter. In MySQL Cluster NDB 7.2.1 and later, this parameter is enabled by default and must be explicitly disabled, in which case data nodes merely log a warning whenever they detect a corrupted tuple.
- **Memcache API support (`ndbmemcache`).** The Memcached server is a distributed in-memory caching server that uses a simple text-based protocol. It is often employed with key-value stores. The Memcache API for MySQL Cluster, available beginning with MySQL Cluster NDB 7.2.2, is implemented as a loadable storage engine for memcached version 1.6 and later. This API can be used to access a persistent MySQL Cluster data store employing the memcache protocol. It is also possible for the `memcached` server to provide a strictly defined interface to existing MySQL Cluster tables.

Each memcache server can both cache data locally and access data stored in MySQL Cluster directly. Caching policies are configurable. For more information, see [`ndbmemcache`—Memcache API for MySQL Cluster](#), in the *MySQL Cluster API Developers Guide*.

- **Rows per partition limit removed.** Previously it was possible to store a maximum of 46137488 rows in a single MySQL Cluster partition—that is, per data node. Beginning with MySQL Cluster NDB 7.2.9, this limitation has been lifted, and there is no longer any practical upper limit to this number.

17.1.5. MySQL Server using InnoDB Compared with MySQL Cluster

MySQL Server offers a number of choices in storage engines. Since both [NDBCLUSTER](#) and [InnoDB](#) can serve as transactional MySQL storage engines, users of MySQL Server sometimes become interested in MySQL Cluster. They see [NDB](#) as a possible alternative or upgrade to the default [InnoDB](#) storage engine in MySQL 5.5. While [NDB](#) and [InnoDB](#) share common characteristics, there are differences in architecture and implementation, so that some existing MySQL Server applications and usage scenarios can be a good fit for MySQL Cluster, but not all of them.

In this section, we discuss and compare some characteristics of the [NDB](#) storage engine used by MySQL Cluster NDB 7.2 with [InnoDB](#) used in MySQL 5.5. The next few sections provide a technical comparison. In many instances, decisions about when and where to use MySQL Cluster must be made on a case-by-case basis, taking all factors into consideration. While it is beyond the scope of this documentation to provide specifics for every conceivable usage scenario, we also attempt to offer some very general guidance on the relative suitability of some common types of applications for [NDB](#) as opposed to [InnoDB](#) backends.

Recent MySQL Cluster NDB 7.2 releases use a `mysqld` based on MySQL 5.5, including support for [InnoDB](#) 1.1. While it is possible to use [InnoDB](#) tables with MySQL Cluster, such tables are not clustered. It is also not possible to use programs or libraries from a MySQL Cluster NDB 7.2 distribution with MySQL Server 5.5, or the reverse.

While it is also true that some types of common business applications can be run either on MySQL Cluster or on MySQL Server (most likely using the [InnoDB](#) storage engine), there are some important architectural and implementation differences. [Section 17.1.5.1, “Differences Between the NDB and InnoDB Storage Engines”](#), provides a summary of the these differences. Due to the differences, some usage scenarios are clearly more suitable for one engine or the other; see [Section 17.1.5.2, “NDB and InnoDB Workloads”](#). This in turn has an impact on the types of applications that better suited for use with [NDB](#) or [InnoDB](#). See [Section 17.1.5.3, “NDB and InnoDB Feature Usage Summary”](#), for a comparison of the relative suitability of each for use in common types of database applications.

For information about the relative characteristics of the [NDB](#) and [MEMORY](#) storage engines, see [When to Use MEMORY or MySQL Cluster](#).

See [Chapter 14, Storage Engines](#), for additional information about MySQL storage engines.

17.1.5.1. Differences Between the [NDB](#) and [InnoDB](#) Storage Engines

The MySQL Cluster [NDB](#) storage engine is implemented using a distributed, shared-nothing architecture, which causes it to behave differently from [InnoDB](#) in a number of ways. For those unaccustomed to working with [NDB](#), unexpected behaviors can arise due to its distributed nature with regard to transactions, foreign keys, table limits, and other characteristics. These are shown in the following table:

Feature	InnoDB 1.1	MySQL Cluster NDB 7.2
<i>MySQL Server Version</i>	5.5	5.5
<i>InnoDB Version</i>	InnoDB 1.1	InnoDB 1.1
<i>MySQL Cluster Version</i>	N/A	NDB 7.2
<i>Storage Limits</i>	64TB	3TB (Practical upper limit based on 48 data nodes with 64GB RAM each; can be increased with disk-based data and BLOBs)
<i>Foreign Keys</i>	Yes	No

Feature	InnoDB 1.1	MySQL Cluster NDB 7.2
		<p>(Ignored, as with MyISAM)</p> <div style="display: flex; align-items: flex-start;"> <div style="margin-right: 10px;">  </div> <div> <p>Note</p> <p>A MySQL Cluster NDB 7.3 Development Milestone Release that includes a preview of foreign key support for the InnoDB storage engine is now available. See https://dev.mysql.com/tech-resources/articles/cluster-7.3-dmr.html, for more information.</p> </div> </div>
<i>Transactions</i>	All standard types	READ COMMITTED [1402]
<i>MVCC</i>	Yes	No
<i>Data Compression</i>	Yes	<p>No</p> <p>(MySQL Cluster checkpoint and backup files can be compressed)</p>
<i>Large Row Support (> 14K)</i>	Supported for VARBINARY, VARCHAR, BLOB, and TEXT columns	<p>Supported for BLOB and TEXT columns only</p> <p>(Using these types to store very large amounts of data can lower MySQL Cluster performance)</p>
<i>Replication Support</i>	Asynchronous and semi-synchronous replication using MySQL Replication	<p>Automatic synchronous replication within a MySQL Cluster.</p> <p>Asynchronous replication between MySQL Clusters, using MySQL Replication</p>
<i>Scaleout for Read Operations</i>	Yes (MySQL Replication)	Yes (Automatic partitioning in MySQL Cluster; MySQL Replication)

Feature	InnoDB 1.1	MySQL Cluster NDB 7.2
<i>Scaleout for Write Operations</i>	Requires application-level partitioning (sharding)	Yes (Automatic partitioning in MySQL Cluster is transparent to applications)
<i>High Availability (HA)</i>	Requires additional software	Yes (Designed for 99.999% uptime)
<i>Node Failure Recovery and Failover</i>	Requires additional software	Automatic (Key element in MySQL Cluster architecture)
<i>Time for Node Failure Recovery</i>	30 seconds or longer	Typically < 1 second
<i>Real-Time Performance</i>	No	Yes
<i>In-Memory Tables</i>	No	Yes (Some data can optionally be stored on disk; both in-memory and disk data storage are durable)
<i>NoSQL Access to Storage Engine</i>	Native memcached interface in development (see the MySQL Dev Zone article NoSQL to MySQL with Memcached)	Yes Multiple APIs, including Memcached, Node.js/JavaScript, Java, JPA, C++, and HTTP/REST
<i>Concurrent and Parallel Writes</i>	Not supported	Up to 48 writers, optimized for concurrent writes
<i>Conflict Detection and Resolution (Multiple Replication Masters)</i>	No	Yes
<i>Hash Indexes</i>	No	Yes
<i>Online Addition of Nodes</i>	Read-only replicas using MySQL Replication	Yes (all node types)
<i>Online Upgrades</i>	No	Yes
<i>Online Schema Modifications</i>	No	Yes, as part of the MySQL 5.6 Release Candidate. See https://dev.mysql.com/tech-resources/articles/mysql-5.6-rc.html , for more information

17.1.5.2. **NDB** and **InnoDB** Workloads

MySQL Cluster has a range of unique attributes that make it ideal to serve applications requiring high availability, fast failover, high throughput, and low latency. Due to its distributed architecture and multi-node implementation, MySQL Cluster also has specific constraints that may keep some workloads from performing well. A number of major differences in behavior between the **NDB** and **InnoDB** storage engines with regard to some common types of database-driven application workloads are shown in the following table::

Workload	InnoDB	MySQL Cluster (NDB)
<i>High-Volume OLTP Applications</i>	Yes	Yes

Workload	InnoDB	MySQL Cluster (NDB)
<i>DSS Applications (data marts, analytics)</i>	Yes	Limited (Join operations across OLTP datasets not exceeding 3TB in size)
<i>Custom Applications</i>	Yes	Yes
<i>Packaged Applications</i>	Yes	Limited (should be mostly primary key access) <div style="display: flex; align-items: flex-start;"> <div style="margin-right: 10px;">  </div> <div> <p>Note</p> <p>A MySQL Cluster NDB 7.3 Development Milestone Release that includes a preview of foreign key support for the InnoDB storage engine is now available. See https://dev.mysql.com/tech-resources/articles/cluster-7.3-dmr.html, for more information.</p> </div> </div>
<i>In-Network Telecoms Applications (HLR, HSS, SDP)</i>	No	Yes
<i>Session Management and Caching</i>	Yes	Yes
<i>E-Commerce Applications</i>	Yes	Yes
<i>User Profile Management, AAA Protocol</i>	Yes	Yes

17.1.5.3. NDB and [InnoDB](#) Feature Usage Summary

When comparing application feature requirements to the capabilities of [InnoDB](#) with [NDB](#), some are clearly more compatible with one storage engine than the other. For example, since [NDB](#) does not support foreign keys, an application that requires them and cannot be re-engineered to remove this requirement is likely not to be a good match for MySQL Cluster.



Note

A MySQL Cluster NDB 7.3 Development Milestone Release that includes a preview of foreign key support for the [InnoDB](https://dev.mysql.com/tech-resources/articles/cluster-7.3-dmr.html) storage engine is now available. See <https://dev.mysql.com/tech-resources/articles/cluster-7.3-dmr.html>, for more information.

The following table shows required supported features for applications according to which of these two storage engines each of them is usually better suited:

Preferred application requirements for InnoDB	Preferred application requirements for NDB
<ul style="list-style-type: none"> • Foreign keys <div style="margin-left: 20px;">  <p>Note</p> <p>A MySQL Cluster NDB 7.3 Development Milestone Release that includes a preview of foreign key support for the InnoDB storage engine is now available. See https://dev.mysql.com/tech-resources/articles/cluster-7.3-dmr.html, for more information.</p> </div> <ul style="list-style-type: none"> • Full table scans • Very large databases, rows, or transactions • Transactions other than READ COMMITTED [1402] 	<ul style="list-style-type: none"> • Write scaling • 99.999% uptime • Online addition of nodes and online schema operations • Multiple SQL and NoSQL APIs (see MySQL Cluster APIs: Overview and Concepts) • Real-time performance • Limited use of BLOB columns

17.1.6. Known Limitations of MySQL Cluster

In the sections that follow, we discuss known limitations in current releases of MySQL Cluster as compared with the features available when using the [MyISAM](#) and [InnoDB](#) storage engines. If you check the “Cluster” category in the MySQL bugs database at <http://bugs.mysql.com>, you can find known bugs in the following categories under “MySQL Server:” in the MySQL bugs database at <http://bugs.mysql.com>, which we intend to correct in upcoming releases of MySQL Cluster:

- MySQL Cluster
- Cluster Direct API (NDBAPI)
- Cluster Disk Data
- Cluster Replication
- ClusterJ

This information is intended to be complete with respect to the conditions just set forth. You can report any discrepancies that you encounter to the MySQL bugs database using the instructions given in [Section 1.7, “How to Report Bugs or Problems”](#). If we do not plan to fix the problem in MySQL Cluster NDB 6.X or 7.X, we will add it to the list.

See [Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”](#) for a list of issues in MySQL Cluster in MySQL 5.1 that have been resolved in the current version.

**Note**

Limitations and other issues specific to MySQL Cluster Replication are described in [Section 17.6.3, “Known Issues in MySQL Cluster Replication”](#).

17.1.6.1. Noncompliance with SQL Syntax in MySQL Cluster

Some SQL statements relating to certain MySQL features produce errors when used with **NDB** tables, as described in the following list:

- **Temporary tables.** Temporary tables are not supported. Trying either to create a temporary table that uses the **NDB** storage engine or to alter an existing temporary table to use **NDB** fails with the error `Table storage engine 'ndbcluster' does not support the create option 'TEMPORARY'`.
- **Indexes and keys in NDB tables.** Keys and indexes on MySQL Cluster tables are subject to the following limitations:
 - **Column width.** Attempting to create an index on an **NDB** table column whose width is greater than 3072 bytes succeeds, but only the first 3072 bytes are actually used for the index. In such cases, a warning `Specified key was too long; max key length is 3072 bytes` is issued, and a `SHOW CREATE TABLE` statement shows the length of the index as 3072.
 - **TEXT and BLOB columns.** You cannot create indexes on **NDB** table columns that use any of the **TEXT** or **BLOB** data types.
 - **FULLTEXT indexes.** The **NDB** storage engine does not support **FULLTEXT** indexes, which are possible for **MyISAM** tables only.

However, you can create indexes on **VARCHAR** columns of **NDB** tables.

- **USING HASH keys and NULL.** Using nullable columns in unique keys and primary keys means that queries using these columns are handled as full table scans. To work around this issue, make the column **NOT NULL**, or re-create the index without the **USING HASH** option.
- **Prefixes.** There are no prefix indexes; only entire columns can be indexed. (The size of an **NDB** column index is always the same as the width of the column in bytes, up to and including 3072 bytes, as described earlier in this section. Also see [Section 17.1.6.6, “Unsupported or Missing Features in MySQL Cluster”](#), for additional information.)
- **BIT columns.** A **BIT** column cannot be a primary key, unique key, or index, nor can it be part of a composite primary key, unique key, or index.
- **AUTO_INCREMENT columns.** Like other MySQL storage engines, the **NDB** storage engine can handle a maximum of one **AUTO_INCREMENT** column per table. However, in the case of a Cluster table with no explicit primary key, an **AUTO_INCREMENT** column is automatically defined and used as a “hidden” primary key. For this reason, you cannot define a table that has an explicit **AUTO_INCREMENT** column unless that column is also declared using the **PRIMARY KEY** option. Attempting to create a table with an **AUTO_INCREMENT** column that is not the table's primary key, and using the **NDB** storage engine, fails with an error.
- **MySQL Cluster and geometry data types.** Geometry data types (**WKT** and **WKB**) are supported for **NDB** tables. However, spatial indexes are not supported.

- **Character sets and binary log files.** Currently, the `ndb_apply_status` and `ndb_binlog_index` tables are created using the `latin1` (ASCII) character set. Because names of binary logs are recorded in this table, binary log files named using non-Latin characters are not referenced correctly in these tables. This is a known issue, which we are working to fix. (Bug #50226)

To work around this problem, use only Latin-1 characters when naming binary log files or setting any the `--basedir` [435], `--log-bin` [1892], or `--log-bin-index` [1892] options.

- **Creating `NDBCLUSTER` tables with user-defined partitioning.** Support for user-defined partitioning for MySQL Cluster in MySQL 5.1 (including MySQL Cluster NDB 6.X and 7.X through 7.1) is restricted to `[LINEAR] KEY` partitioning. Beginning with MySQL 5.1.12, using any other partitioning type with `ENGINE=NDB` or `ENGINE=NDBCLUSTER` in a `CREATE TABLE` statement results in an error.

Default partitioning scheme. As of MySQL 5.1.6, all MySQL Cluster tables are by default partitioned by `KEY` using the table's primary key as the partitioning key. If no primary key is explicitly set for the table, the “hidden” primary key automatically created by the `NDBCLUSTER` storage engine is used instead. For additional discussion of these and related issues, see [Section 18.2.5, “KEY Partitioning”](#).

`CREATE TABLE` and `ALTER TABLE` statements that would cause a user-partitioned `NDBCLUSTER` table not to meet either or both of the following two requirements are not permitted, and fail with an error:

1. The table must have an explicit primary key.
2. All columns listed in the table's partitioning expression must be part of the primary key.

Exception. If a user-partitioned `NDBCLUSTER` table is created using an empty column-list (that is, using `PARTITION BY [LINEAR] KEY()`), then no explicit primary key is required.

Maximum number of partitions for `NDBCLUSTER` tables. The maximum number of partitions that can be defined for a `NDBCLUSTER` table when employing user-defined partitioning is 8 per node group. (See [Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#), for more information about MySQL Cluster node groups.

`DROP PARTITION` not supported. It is not possible to drop partitions from `NDB` tables using `ALTER TABLE ... DROP PARTITION`. The other partitioning extensions to `ALTER TABLE—ADD PARTITION`, `REORGANIZE PARTITION`, and `COALESCE PARTITION`—are supported for Cluster tables, but use copying and so are not optimized. See [Section 18.3.1, “Management of RANGE and LIST Partitions”](#) and [Section 13.1.7, “ALTER TABLE Syntax”](#).

- **Row-based replication.** When using row-based replication with MySQL Cluster, binary logging cannot be disabled. That is, the `NDB` storage engine ignores the value of `sql_log_bin` [555]. (Bug #16680)

17.1.6.2. Limits and Differences of MySQL Cluster from Standard MySQL Limits

In this section, we list limits found in MySQL Cluster that either differ from limits found in, or that are not found in, standard MySQL.

Memory usage and recovery. Memory consumed when data is inserted into an `NDB` table is not automatically recovered when deleted, as it is with other storage engines. Instead, the following rules hold true:

- A `DELETE` statement on an `NDB` table makes the memory formerly used by the deleted rows available for re-use by inserts on the same table only. However, this memory can be made available for general re-use by performing `OPTIMIZE TABLE`.

A rolling restart of the cluster also frees any memory used by deleted rows. See [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#).

- A `DROP TABLE` or `TRUNCATE TABLE` operation on an `NDB` table frees the memory that was used by this table for re-use by any `NDB` table, either by the same table or by another `NDB` table.



Note

Recall that `TRUNCATE TABLE` drops and re-creates the table. See [Section 13.1.33, “TRUNCATE TABLE Syntax”](#).

- **Limits imposed by the cluster's configuration.**

A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in [Section 17.3.2, “MySQL Cluster Configuration Files”](#). Most configuration parameters can be upgraded online. These hard limits include:

- Database memory size and index memory size (`DataMemory` [2043] and `IndexMemory` [2044], respectively).

`DataMemory` [2043] is allocated as 32KB pages. As each `DataMemory` [2043] page is used, it is assigned to a specific table; once allocated, this memory cannot be freed except by dropping the table.

See [Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”](#), for more information.

- The maximum number of operations that can be performed per transaction is set using the configuration parameters `MaxNoOfConcurrentOperations` [2047] and `MaxNoOfLocalOperations` [2048].



Note

Bulk loading, `TRUNCATE TABLE`, and `ALTER TABLE` are handled as special cases by running multiple transactions, and so are not subject to this limitation.

- Different limits related to tables and indexes. For example, the maximum number of ordered indexes in the cluster is determined by `MaxNoOfOrderedIndexes` [2055], and the maximum number of ordered indexes per table is 16.

- **Node and data object maximums.** The following limits apply to numbers of cluster nodes and metadata objects:

- The maximum number of data nodes is 48.

A data node must have a node ID in the range of 1 to 49, inclusive. (Management and API nodes may use node IDs in the range 1 to 255, inclusive.)

- The total maximum number of nodes in a MySQL Cluster is 255. This number includes all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers.
- The maximum number of metadata objects in current versions of MySQL Cluster is 20320. This limit is hard-coded.

See [Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”](#), for more information.

17.1.6.3. Limits Relating to Transaction Handling in MySQL Cluster

A number of limitations exist in MySQL Cluster with regard to the handling of transactions. These include the following:

- **Transaction isolation level.** The `NDBCLUSTER` storage engine supports only the `READ COMMITTED` [1402] transaction isolation level. (`InnoDB`, for example, supports `READ COMMITTED` [1402], `READ UNCOMMITTED` [1402], `REPEATABLE READ` [1401], and `SERIALIZABLE` [1402].) See Section 17.5.3.4, “MySQL Cluster Backup Troubleshooting”, for information on how this can affect backing up and restoring Cluster databases.)
- **Transactions and `BLOB` or `TEXT` columns.** `NDBCLUSTER` stores only part of a column value that uses any of MySQL's `BLOB` or `TEXT` data types in the table visible to MySQL; the remainder of the `BLOB` or `TEXT` is stored in a separate internal table that is not accessible to MySQL. This gives rise to two related issues of which you should be aware whenever executing `SELECT` statements on tables that contain columns of these types:
 1. For any `SELECT` from a MySQL Cluster table: If the `SELECT` includes a `BLOB` or `TEXT` column, the `READ COMMITTED` [1402] transaction isolation level is converted to a read with read lock. This is done to guarantee consistency.
 2. For any `SELECT` which uses a unique key lookup to retrieve any columns that use any of the `BLOB` or `TEXT` data types and that is executed within a transaction, a shared read lock is held on the table for the duration of the transaction—that is, until the transaction is either committed or aborted.

This issue does not occur for queries that use index or table scans, even against `NDB` tables having `BLOB` or `TEXT` columns.

For example, consider the table `t` defined by the following `CREATE TABLE` statement:

```
CREATE TABLE t (
  a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b INT NOT NULL,
  c INT NOT NULL,
  d TEXT,
  INDEX i(b),
  UNIQUE KEY u(c)
) ENGINE = NDB,
```

Either of the following queries on `t` causes a shared read lock, because the first query uses a primary key lookup and the second uses a unique key lookup:

```
SELECT * FROM t WHERE a = 1;
SELECT * FROM t WHERE c = 1;
```

However, none of the four queries shown here causes a shared read lock:

```
SELECT * FROM t WHERE b = 1;
SELECT * FROM t WHERE d = '1';
SELECT * FROM t;
SELECT b,c WHERE a = 1;
```

This is because, of these four queries, the first uses an index scan, the second and third use table scans, and the fourth, while using a primary key lookup, does not retrieve the value of any `BLOB` or `TEXT` columns.

You can help minimize issues with shared read locks by avoiding queries that use unique key lookups that retrieve `BLOB` or `TEXT` columns, or, in cases where such queries are not avoidable, by committing transactions as soon as possible afterward.

- **Rollbacks.** There are no partial transactions, and no partial rollbacks of transactions. A duplicate key or similar error causes the entire transaction to be rolled back.

This behavior differs from that of other transactional storage engines such as `InnoDB` that may roll back individual statements.

- **Transactions and memory usage.**

As noted elsewhere in this chapter, MySQL Cluster does not handle large transactions well; it is better to perform a number of small transactions with a few operations each than to attempt a single large transaction containing a great many operations. Among other considerations, large transactions require very large amounts of memory. Because of this, the transactional behavior of a number of MySQL statements is effected as described in the following list:

- `TRUNCATE TABLE` is not transactional when used on `NDB` tables. If a `TRUNCATE TABLE` fails to empty the table, then it must be re-run until it is successful.
- `DELETE FROM` (even with no `WHERE` clause) is transactional. For tables containing a great many rows, you may find that performance is improved by using several `DELETE FROM ... LIMIT ...` statements to “chunk” the delete operation. If your objective is to empty the table, then you may wish to use `TRUNCATE TABLE` instead.
- **LOAD DATA statements.** `LOAD DATA INFILE` is not transactional when used on `NDB` tables.



Important

When executing a `LOAD DATA INFILE` statement, the `NDB` engine performs commits at irregular intervals that enable better utilization of the communication network. It is not possible to know ahead of time when such commits take place.

`LOAD DATA FROM MASTER` is not supported in MySQL Cluster.

- **ALTER TABLE and transactions.** When copying an `NDB` table as part of an `ALTER TABLE`, the creation of the copy is nontransactional. (In any case, this operation is rolled back when the copy is deleted.)
- **Transactions and the `COUNT()` function.** When using MySQL Cluster Replication, it is not possible to guarantee the transactional consistency of the `COUNT()` function on the slave. In other words, when performing on the master a series of statements (`INSERT`, `DELETE`, or both) that changes the number of rows in a table within a single transaction, executing `SELECT COUNT(*) FROM table` queries on the slave may yield intermediate results. This is due to the fact that `SELECT COUNT(...)` may perform dirty reads, and is not a bug in the `NDB` storage engine. (See Bug #31321 for more information.)

17.1.6.4. MySQL Cluster Error Handling

Starting, stopping, or restarting a node may give rise to temporary errors causing some transactions to fail. These include the following cases:

- **Temporary errors.** When first starting a node, it is possible that you may see Error 1204 `Temporary failure, distribution changed` and similar temporary errors.

- **Errors due to node failure.** The stopping or failure of any data node can result in a number of different node failure errors. (However, there should be no aborted transactions when performing a planned shutdown of the cluster.)

In either of these cases, any errors that are generated must be handled within the application. This should be done by retrying the transaction.

See also [Section 17.1.6.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”](#).

17.1.6.5. Limits Associated with Database Objects in MySQL Cluster

Some database objects such as tables and indexes have different limitations when using the `NDBCLUSTER` storage engine:

- **Number of database objects.** The maximum number of *all* `NDB` database objects in a single MySQL Cluster—including databases, tables, and indexes—is limited to 20320.
- **Attributes per table.** The maximum number of attributes (that is, columns and indexes) that can belong to a given table is 512.
- **Attributes per key.** The maximum number of attributes per key is 32.
- **Row size.** The maximum permitted size of any one row is 14000 bytes (as of MySQL Cluster NDB 7.0). Each `BLOB` or `TEXT` column contributes $256 + 8 = 264$ bytes to this total.
- **Number of rows per partition.** The maximum number of rows that can be stored in a single MySQL Cluster partition varies with the number of replicas times the number of fragments. A single fragment can hold a maximum of approximately 45 million rows. Since the number of partitions is the same as the number of data nodes in the cluster (see [Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#)), you can increase the number of fragments—and thus the available space—by using more data nodes.

In MySQL Cluster NDB 7.0 and later MySQL Cluster release series, you can increase the number of data nodes in the cluster while the cluster remains in operation. See [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#), for more information.

It is also possible to increase the number of partitions for `NDB` tables by using explicit `KEY` or `LINEAR KEY` partitioning (see [Section 18.2.5, “KEY Partitioning”](#)).

17.1.6.6. Unsupported or Missing Features in MySQL Cluster

A number of features supported by other storage engines are not supported for `NDB` tables. Trying to use any of these features in MySQL Cluster does not cause errors in or of itself; however, errors may occur in applications that expects the features to be supported or enforced:

- **Foreign key constraints.** The foreign key construct is ignored, just as it is in `MyISAM` tables.
- **Index prefixes.** Prefixes on indexes are not supported for `NDBCLUSTER` tables. If a prefix is used as part of an index specification in a statement such as `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX`, the prefix is ignored.
- **LOAD TABLE ... FROM MASTER.** `LOAD TABLE FROM MASTER` is not supported.
- **Savepoints and rollbacks.** Savepoints and rollbacks to savepoints are ignored as in `MyISAM`.
- **Durability of commits.** There are no durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.

- **Replication.** Statement-based replication is not supported. Use `--binlog-format=ROW` [436] (or `--binlog-format=MIXED` [436]) when setting up cluster replication. See Section 17.6, “MySQL Cluster Replication”, for more information.



Note

See Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”, for more information relating to limitations on transaction handling in NDB.

17.1.6.7. Limitations Relating to Performance in MySQL Cluster

The following performance issues are specific to or especially pronounced in MySQL Cluster:

- **Range scans.** There are query performance issues due to sequential access to the NDB storage engine; it is also relatively more expensive to do many range scans than it is with either MyISAM or InnoDB.
- **Reliability of Records in range.** The `Records in range` statistic is available but is not completely tested or officially supported. This may result in nonoptimal query plans in some cases. If necessary, you can employ `USE INDEX` or `FORCE INDEX` to alter the execution plan. See Section 13.2.9.3, “Index Hint Syntax”, for more information on how to do this.
- **Unique hash indexes.** Unique hash indexes created with `USING HASH` cannot be used for accessing a table if `NULL` is given as part of the key.

17.1.6.8. Issues Exclusive to MySQL Cluster

The following are limitations specific to the NDBCLUSTER storage engine:

- **Machine architecture.** All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running `mysql` or other clients that may be accessing the cluster's SQL nodes.
- **Binary logging.** MySQL Cluster has the following limitations or restrictions with regard to binary logging:
 - `sql_log_bin` [555] has no effect on data operations; however, it is supported for schema operations.
 - MySQL Cluster cannot produce a binary log for tables having `BLOB` columns but no primary key.
 - Only the following schema operations are logged in a cluster binary log which is *not* on the `mysqld` executing the statement:
 - `CREATE TABLE`
 - `ALTER TABLE`
 - `DROP TABLE`
 - `CREATE DATABASE / CREATE SCHEMA`
 - `DROP DATABASE / DROP SCHEMA`
 - `CREATE TABLESPACE`
 - `ALTER TABLESPACE`

- `DROP TABLESPACE`
- `CREATE LOGFILE GROUP`
- `ALTER LOGFILE GROUP`
- `DROP LOGFILE GROUP`

See also [Section 17.1.6.10, “Limitations Relating to Multiple MySQL Cluster Nodes”](#).

17.1.6.9. Limitations Relating to MySQL Cluster Disk Data Storage

Disk Data object maximums and minimums. Disk data objects are subject to the following maximums and minimums:

- Maximum number of tablespaces: 2^{32} (4294967296)
- Maximum number of data files per tablespace: 2^{16} (65536)
- Maximum data file size: The theoretical limit is 64G; however, the practical upper limit is 32G. This is equivalent to 32768 extents of 1M each.

Since a MySQL Cluster Disk Data table can use at most 1 tablespace, this means that the theoretical upper limit to the amount of data (in bytes) that can be stored on disk by a single `NDB` table is $32\text{G} * 65536 = 2251799813685248$, or approximately 2 petabytes.

- The theoretical maximum number of extents per tablespace data file is 2^{16} (65536); however, for practical purposes, the recommended maximum number of extents per data file is 2^{15} (32768).

The minimum and maximum possible sizes of extents for tablespace data files are 32K and 2G, respectively. See [Section 13.1.18, “CREATE TABLESPACE Syntax”](#), for more information.

Disk Data tables and diskless mode. Use of Disk Data tables is not supported when running the cluster in diskless mode. Beginning with MySQL 5.1.12, it is prohibited altogether. (Bug #20008)

17.1.6.10. Limitations Relating to Multiple MySQL Cluster Nodes

Multiple SQL nodes.

The following are issues relating to the use of multiple MySQL servers as MySQL Cluster SQL nodes, and are specific to the `NDBCLUSTER` storage engine:

- **No distributed table locks.** A `LOCK TABLES` works only for the SQL node on which the lock is issued; no other SQL node in the cluster “sees” this lock. This is also true for a lock issued by any statement that locks tables as part of its operations. (See next item for an example.)
- **ALTER TABLE operations.** `ALTER TABLE` is not fully locking when running multiple MySQL servers (SQL nodes). (As discussed in the previous item, MySQL Cluster does not support distributed table locks.)

Multiple management nodes.

When using multiple management servers:

- You must give nodes explicit IDs in connect strings because automatic allocation of node IDs does not work across multiple management servers.
- When a management server starts, it first checks for any other management server in the same MySQL Cluster, and upon successful connection to the other management server uses its configuration data. This means that the management server `--reload [2191]` and `--initial [2190]` startup options are

ignored unless the management server is the only one running. It also means that, when performing a rolling restart of a MySQL Cluster with multiple management nodes, the management server reads its own configuration file if (and only if) it is the only management server running in this MySQL Cluster. See [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#), for more information.

Multiple network addresses. Multiple network addresses per data node are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits for confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable.



Note

It is possible to use multiple network hardware *interfaces* (such as Ethernet cards) for a single data node, but these must be bound to the same address. This also means that it not possible to use more than one `[tcp]` section per connection in the `config.ini` file. See [Section 17.3.2.8, “MySQL Cluster TCP/IP Connections”](#), for more information.

17.1.6.11. Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x

A number of limitations and related issues existing in earlier versions of MySQL Cluster have been resolved:

- **Variable-length column support.** The `NDBCLUSTER` storage engine now supports variable-length column types for in-memory tables.

Previously, for example, any Cluster table having one or more `VARCHAR` fields which contained only relatively small values, much more memory and disk space were required when using the `NDBCLUSTER` storage engine than would have been the case for the same table and data using the `MyISAM` engine. In other words, in the case of a `VARCHAR` column, such a column required the same amount of storage as a `CHAR` column of the same size. In MySQL 5.1, this is no longer the case for in-memory tables, where storage requirements for variable-length column types such as `VARCHAR` and `BINARY` are comparable to those for these column types when used in `MyISAM` tables (see [Section 11.6, “Data Type Storage Requirements”](#)).



Important

For MySQL Cluster Disk Data tables, the fixed-width limitation continues to apply. See [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#).

- **Replication with MySQL Cluster.** It is now possible to use MySQL replication with Cluster databases. For details, see [Section 17.6, “MySQL Cluster Replication”](#).

Circular Replication. Circular replication is also supported with MySQL Cluster, beginning with MySQL 5.1.18. See [Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”](#).

- **`auto_increment_increment` and `auto_increment_offset`.** The `auto_increment_increment` [1865] and `auto_increment_offset` [1868] server system variables are supported for MySQL Cluster Replication.
- **Backup and restore between architectures.** It is possible to perform a Cluster backup and restore between different architectures. Previously—for example—you could not back up a cluster running on a big-endian platform and then restore from that backup to a cluster running on a little-endian system. (Bug #19255)

- **Multiple data nodes, multi-threaded data nodes.** MySQL Cluster NDB 7.2 supports multiple data node processes on a single host as well as multi-threaded data node processes. See [Section 17.4.3, “ndbmt_d — The MySQL Cluster Data Node Daemon \(Multi-Threaded\)”](#), for more information.
- **Identifiers.** Formerly (in MySQL 5.0 and earlier), database names, table names and attribute names could not be as long for NDB tables as tables using other storage engines, because attribute names were truncated internally. In MySQL 5.1 and later, names of MySQL Cluster databases, tables, and table columns follow the same rules regarding length as they do for any other storage engine.
- **Length of CREATE TABLE statements.** CREATE TABLE statements may be no more than 4096 characters in length. *This limitation affects MySQL 5.1.6, 5.1.7, and 5.1.8 only.* (See Bug #17813)
- **IGNORE and REPLACE functionality.** In MySQL 5.1.7 and earlier, INSERT IGNORE, UPDATE IGNORE, and REPLACE were supported only for primary keys, but not for unique keys. It was possible to work around this issue by removing the constraint, then dropping the unique index, performing any inserts, and then adding the unique index again.

This limitation was removed for INSERT IGNORE and REPLACE in MySQL 5.1.8. (See Bug #17431.)

- **AUTO_INCREMENT columns.** In MySQL 5.1.10 and earlier versions, the maximum number of tables having AUTO_INCREMENT columns—including those belonging to hidden primary keys—was 2048.

This limitation was lifted in MySQL 5.1.11.

- **Maximum number of cluster nodes.** The total maximum number of nodes in a MySQL Cluster is 255, including all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers. The total number of data nodes and management nodes is 63, of which up to 48 can be data nodes.



Note

A data node cannot have a node ID greater than 49.

- **Recovery of memory from deleted rows.** Memory can be reclaimed from an NDB table for reuse with any NDB table by employing OPTIMIZE TABLE, subject to the following limitations:
 - Only in-memory tables are supported; the OPTIMIZE TABLE statement has no effect on MySQL Cluster Disk Data tables.
 - Only variable-length columns (such as those declared as VARCHAR, TEXT, or BLOB) are supported.

However, you can force columns defined using fixed-length data types (such as CHAR) to be dynamic using the ROW_FORMAT or COLUMN_FORMAT option with a CREATE TABLE or ALTER TABLE statement.

See [Section 13.1.17, “CREATE TABLE Syntax”](#), and [Section 13.1.7, “ALTER TABLE Syntax”](#), for information on these options.

You can regulate the effects of OPTIMIZE on performance by adjusting the value of the global system variable `ndb_optimization_delay`, which sets the number of milliseconds to wait between batches of rows being processed by OPTIMIZE. The default value is 10 milliseconds. It is possible to set a lower value (to a minimum of 0), but not recommended. The maximum is 100000 milliseconds (that is, 100 seconds).

- **Number of tables.** The maximum number of NDBCLUSTER tables in a single MySQL Cluster is included in the total maximum number of NDBCLUSTER database objects (20320). (See [Section 17.1.6.5, “Limits Associated with Database Objects in MySQL Cluster”](#).)

- **Adding and dropping of data nodes.** In MySQL Cluster NDB 7.2 (MySQL Cluster NDB 7.0 and later), it is possible to add new data nodes to a running MySQL Cluster by performing a rolling restart, so that the cluster and the data stored in it remain available to applications.

When planning to increase the number of data nodes in the cluster online, you should be aware of and take into account the following issues:

- New data nodes can be added online to a MySQL Cluster only as part of a new node group.
- New data nodes can be added online, but cannot be dropped online. Reducing the number of data nodes requires a system restart of the cluster.
- As in previous MySQL Cluster releases, it is not possible to change online either the number of replicas (`NoOfReplicas` [2041] configuration parameter) or the number of data nodes per node group. These changes require a system restart.
- Redistribution of existing cluster data using the new data nodes is not automatic; however, this can be accomplished using simple SQL statements in the `mysql` client or other MySQL client application once the nodes have been added. During this procedure, it is not possible to perform DDL operations, although DML operations can continue as normal.

The distribution of new cluster data (that is, data stored in the cluster *after* the new nodes have been added) uses the new nodes without manual intervention.

For more information, see [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#).

- **Distribution of MySQL users and privileges.** Previously, MySQL users and privileges created on one SQL node were unique to that SQL node, due to the fact that the MySQL grant tables were restricted to using the `MyISAM` storage engine. Beginning with MySQL Cluster NDB 7.2.0, it is possible, following installation of the MySQL Cluster software and setup of the desired users and privileges on one SQL node, to convert the grant tables to use `NDB` and thus to distribute the users and privileges across all SQL nodes connected to the cluster. You can do this by loading and making use of a set of stored procedures defined in an SQL script supplied with the MySQL Cluster distribution. For more information, see [Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”](#).
- **Number of rows per partition.** Previously, a single MySQL Cluster partition could hold a maximum of 46137488 rows. This limitation was removed in MySQL Cluster NDB 7.2.9.

If you are still using a previous MySQL Cluster release, you can work around this limitation by taking advantage of the fact that the number of partitions is the same as the number of data nodes in the cluster (see [Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#)). This means that, by increasing the number of data nodes, you can increase the available space for storing data.

MySQL Cluster NDB 7.2 also supports increasing the number of data nodes in the cluster while the cluster remains in operation. See [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#), for more information.

It is also possible to increase the number of partitions for `NDB` tables by using explicit `KEY` or `LINEAR KEY` partitioning (see [Section 18.2.5, “KEY Partitioning”](#)).

17.2. MySQL Cluster Installation

This section describes the basics for planning, installing, configuring, and running a MySQL Cluster. Whereas the examples in [Section 17.3, “MySQL Cluster Configuration”](#) provide more in-depth information on a variety of clustering options and configuration, the result of following the guidelines and procedures

outlined here should be a usable MySQL Cluster which meets the *minimum* requirements for availability and safeguarding of data.

For information about upgrading or downgrading a MySQL Cluster between release versions, see [Section 17.2.7, "Upgrading and Downgrading MySQL Cluster NDB 7.2"](#).

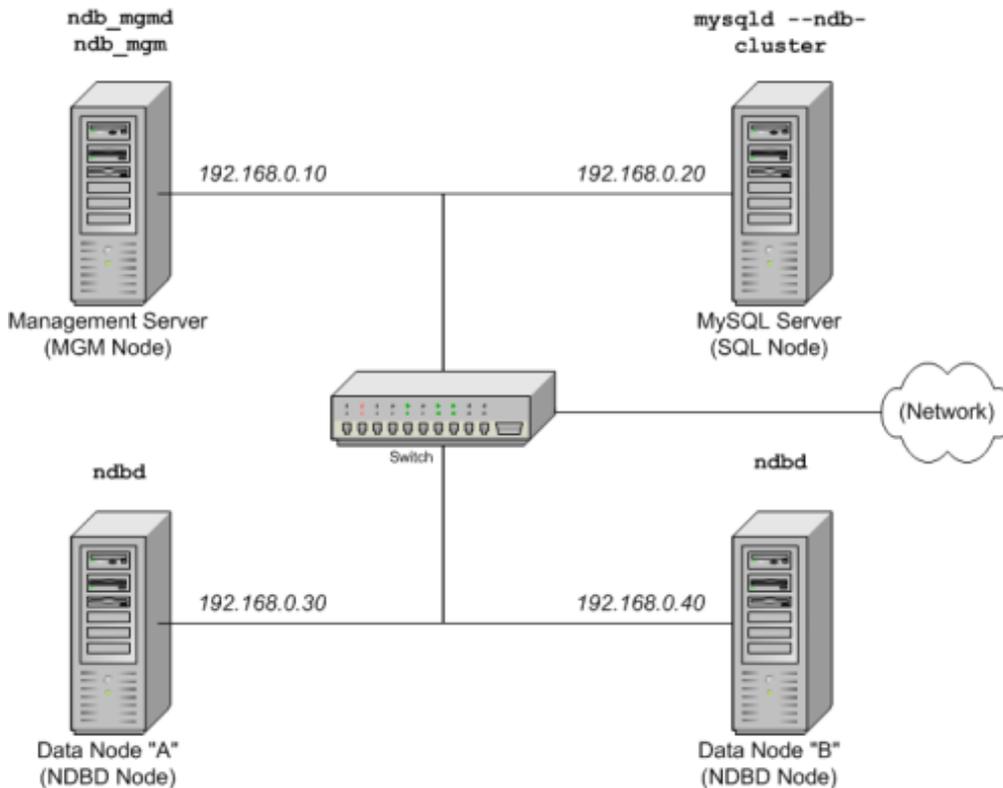
This section covers hardware and software requirements; networking issues; installation of MySQL Cluster; configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

Assumptions. The following sections make a number of assumptions regarding the cluster's physical and network configuration. These assumptions are discussed in the next few paragraphs.

Cluster nodes and host computers. The cluster consists of four nodes, each on a separate host computer, and each with a fixed network address on a typical Ethernet network as shown here:

Node	IP Address
Management node (<code>mgmd</code>)	192.168.0.10
SQL node (<code>mysqld</code>)	192.168.0.20
Data node "A" (<code>ndbd</code>)	192.168.0.30
Data node "B" (<code>ndbd</code>)	192.168.0.40

This may be made clearer by the following diagram:



Network addressing. In the interest of simplicity (and reliability), this *How-To* uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use host names in lieu of IP addresses in configuring Cluster. Alternatively, you can use the `hosts` file (typically `/etc/`

`hosts` for Linux and other Unix-like operating systems, `C:\WINDOWS\system32\drivers\etc\hosts` on Windows, or your operating system's equivalent) for providing a means to do host lookup if such is available.

Potential hosts file issues. A common problem when trying to use host names for Cluster nodes arises because of the way in which some operating systems (including some Linux distributions) set up the system's own host name in the `/etc/hosts` during installation. Consider two machines with the host names `ndb1` and `ndb2`, both in the `cluster` network domain. Red Hat Linux (including some derivatives such as CentOS and Fedora) places the following entries in these machines' `/etc/hosts` files:

```
# ndb1 /etc/hosts:
127.0.0.1    ndb1.cluster ndb1 localhost.localdomain localhost
```

```
# ndb2 /etc/hosts:
127.0.0.1    ndb2.cluster ndb2 localhost.localdomain localhost
```

SUSE Linux (including OpenSUSE) places these entries in the machines' `/etc/hosts` files:

```
# ndb1 /etc/hosts:
127.0.0.1    localhost
127.0.0.2    ndb1.cluster ndb1
```

```
# ndb2 /etc/hosts:
127.0.0.1    localhost
127.0.0.2    ndb2.cluster ndb2
```

In both instances, `ndb1` routes `ndb1.cluster` to a loopback IP address, but gets a public IP address from DNS for `ndb2.cluster`, while `ndb2` routes `ndb2.cluster` to a loopback address and obtains a public address for `ndb1.cluster`. The result is that each data node connects to the management server, but cannot tell when any other data nodes have connected, and so the data nodes appear to hang while starting.



Caution

You cannot mix `localhost` and other host names or IP addresses in `config.ini`. For these reasons, the solution in such cases (other than to use IP addresses for *all* `config.ini` `HostName` entries) is to remove the fully qualified host names from `/etc/hosts` and use these in `config.ini` for all cluster hosts.

Host computer type. Each host computer in our installation scenario is an Intel-based desktop PC running a supported operating system installed to disk in a standard configuration, and running no unnecessary services. The core operating system with standard TCP/IP networking capabilities should be sufficient. Also for the sake of simplicity, we also assume that the file systems on all hosts are set up identically. In the event that they are not, you should adapt these instructions accordingly.

Network hardware. Standard 100 Mbps or 1 gigabit Ethernet cards are installed on each machine, along with the proper drivers for the cards, and that all four hosts are connected through a standard-issue Ethernet networking appliance such as a switch. (All machines should use network cards with the same throughput. That is, all four machines in the cluster should have 100 Mbps cards *or* all four machines should have 1 Gbps cards.) MySQL Cluster works in a 100 Mbps network; however, gigabit Ethernet provides better performance.



Important

MySQL Cluster is *not* intended for use in a network for which throughput is less than 100 Mbps or which experiences a high degree of latency. For this reason (among others), attempting to run a MySQL Cluster over a wide area network such as the Internet is not likely to be successful, and is not supported in production.

Sample data. We use the `world` database which is available for download from the MySQL Web site (see <http://dev.mysql.com/doc/index-other.html>). We assume that each machine has sufficient memory for running the operating system, required MySQL Cluster processes, and (on the data nodes) storing the database.

For general information about installing MySQL, see [Chapter 2, *Installing and Upgrading MySQL*](#). For information about installation of MySQL Cluster on Linux and other Unix-like operating systems, see [Section 17.2.1, “Installing MySQL Cluster on Linux”](#). For information about installation of MySQL Cluster on Windows operating systems, see [Section 17.2.2, “Installing MySQL Cluster on Windows”](#).

For general information about MySQL Cluster hardware, software, and networking requirements, see [Section 17.1.3, “MySQL Cluster Hardware, Software, and Networking Requirements”](#).

17.2.1. Installing MySQL Cluster on Linux

This section covers installation of MySQL Cluster on Linux and other Unix-like operating systems. While the next few sections refer to a Linux operating system, the instructions and procedures given there should be easily adaptable to other supported Unix-like platforms.

MySQL Cluster NDB 7.2 is also available for Windows operating systems; for installation and setup instructions specific to Windows, see [Section 17.2.2, “Installing MySQL Cluster on Windows”](#).

Each MySQL Cluster host computer must have the correct executable programs installed. A host running an SQL node must have installed on it a MySQL Server binary (`mysqld`). Management nodes require the management server daemon (`ndb_mgmd`); data nodes require the data node daemon (`ndbd` or `ndbmt.d`). It is not necessary to install the MySQL Server binary on management node hosts and data node hosts. It is recommended that you also install the management client (`ndb_mgm`) on the management server host.

Installation of MySQL Cluster on Linux can be done using precompiled binaries from Oracle (downloaded as a `.tar.gz` archive), with RPM packages (also available from Oracle), or from source code. All three of these installation methods are described in the section that follow.

Regardless of the method used, it is still necessary following installation of the MySQL Cluster binaries to create configuration files for all cluster nodes, before you can start the cluster. See [Section 17.2.3, “Initial Configuration of MySQL Cluster”](#).

17.2.1.1. Installing a MySQL Cluster Binary Release on Linux

This section covers the steps necessary to install the correct executables for each type of Cluster node from precompiled binaries supplied by Oracle.

For setting up a cluster using precompiled binaries, the first step in the installation process for each cluster host is to download the latest MySQL Cluster NDB 7.2 binary archive (`mysql-cluster-gpl-7.2.10-linux-i686-glibc23.tar.gz`) from the [MySQL Cluster downloads area](#). We assume that you have placed this file in each machine's `/var/tmp` directory. (If you do require a custom binary, see [Section 2.10.3, “Installing MySQL from a Development Source Tree”](#).)



Note

After completing the installation, do not yet start any of the binaries. We show you how to do so following the configuration of the nodes (see [Section 17.2.3, “Initial Configuration of MySQL Cluster”](#)).

SQL nodes. On each of the machines designated to host SQL nodes, perform the following steps as the system `root` user:

1. Check your `/etc/passwd` and `/etc/group` files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a `mysql` group and `mysql` user on the system. Some OS distributions create these as part of the operating system installation process. If they are not already present, create a new `mysql` user group, and then add a `mysql` user to this group:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

2. Change location to the directory containing the downloaded file, unpack the archive, and create a symbolic link named `mysql` to the `mysql` directory. Note that the actual file and directory names vary according to the MySQL Cluster version number.

```
shell> cd /var/tmp
shell> tar -C /usr/local -xzf mysql-cluster-gpl-7.2.10-linux2.6.tar.gz
shell> ln -s /usr/local/mysql-cluster-gpl-7.2.10-linux2.6-i686 /usr/local/mysql
```

3. Change location to the `mysql` directory and run the supplied script for creating the system databases:

```
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
```

4. Set the necessary permissions for the MySQL server and data directories:

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

```
shell> cp support-files/mysql.server /etc/rc.d/init.d/
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server
```

(The startup scripts directory may vary depending on your operating system and version—for example, in some Linux distributions, it is `/etc/init.d`.)

Here we use Red Hat's `chkconfig` for creating links to the startup scripts; use whatever means is appropriate for this purpose on your platform, such as `update-rc.d` on Debian.

Remember that the preceding steps must be repeated on each machine where an SQL node is to reside.

Data nodes. Installation of the data nodes does not require the `mysqld` binary. Only the MySQL Cluster data node executable `ndbd` (singlethreaded) or `ndbmtbd` (multithreaded) is required. These binaries can also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root` (that is, after using `sudo`, `su root`, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install the data node binaries on the data node hosts:

1. Change location to the `/var/tmp` directory, and extract the `ndbd` and `ndbmtbd` binaries from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-5.5.29-ndb-7.2.10-linux-i686-glibc23.tar.gz
shell> cd mysql-5.5.29-ndb-7.2.10-linux-i686-glibc23
```

```
shell> cp bin/ndbd /usr/local/bin/ndbd
shell> cp bin/ndbmttd /usr/local/bin/ndbmttd
```

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndb*
```

The preceding steps should be repeated on each data node host.

Although only one of the data node executables is required to run a MySQL Cluster data node, we have shown you how to install both `ndbd` and `ndbmttd` in the preceding instructions. We recommend that you do this when installing or upgrading MySQL Cluster, even if you plan to use only one of them, since this will save time and trouble in the event that you later decide to change from one to the other.



Note

The data directory on each machine hosting a data node is `/usr/local/mysql/data`. This piece of information is essential when configuring the management node. (See [Section 17.2.3, “Initial Configuration of MySQL Cluster”](#).)

Management nodes. Installation of the management node does not require the `mysqld` binary. Only the MySQL Cluster management server (`ndb_mgmd`) is required; you most likely want to install the management client (`ndb_mgm`) as well. Both of these binaries also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root`, perform the following steps to install `ndb_mgmd` and `ndb_mgm` on the management node host:

1. Change location to the `/var/tmp` directory, and extract the `ndb_mgm` and `ndb_mgmd` from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-5.5.29-ndb-7.2.10-linux2.6-i686.tar.gz
shell> cd mysql-5.5.29-ndb-7.2.10-linux2.6-i686
shell> cp bin/ndb_mgm* /usr/local/bin
```

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

In [Section 17.2.3, “Initial Configuration of MySQL Cluster”](#), we create configuration files for all of the nodes in our example MySQL Cluster.

17.2.1.2. Installing MySQL Cluster from RPM

This section covers the steps necessary to install the correct executables for each type of MySQL Cluster node using RPM packages supplied by Oracle.

RPMs are available for both 32-bit and 64-bit Linux platforms. The filenames for these RPMs use the following pattern:

```
MySQL-Cluster-component-producttype-ndbversion.distribution.architecture.rpm

component:= {server | client [| other]}

producttype:= {gpl | advanced}

ndbversion:= major.minor.release

distribution:= {sles10 | rhel5 | el6}

architecture:= {i386 | x86_64}
```

The *component* can be *server* or *client*. (Other values are possible, but since only the *server* and *client* components are required for a working MySQL Cluster installation, we do not discuss them here.) The *producttype* for Community RPMs downloaded from <http://dev.mysql.com/downloads/cluster/> is always *gpl*; *advanced* is used to indicate commercial releases. *ndbversion* represents the three-part NDB storage engine version number in 7.2.x format. The *distribution* can be one of *sles11* (SUSE Enterprise Linux 11), *rhel5* (Oracle Linux 5, Red Hat Enterprise Linux 4 and 5), or *el6* (Oracle Linux 6, Red Hat Enterprise Linux 6) The *architecture* is *i386* for 32-bit RPMs and *x86_64* for 64-bit versions.

For a MySQL Cluster, one and possibly two RPMs are required:

- The *server* RPM (for example, `MySQL-Cluster-server-gpl-7.2.10-1.sles11.i386.rpm`), which supplies the core files needed to run a MySQL Server with NDBCLUSTER storage engine support (that is, as a MySQL Cluster SQL node) as well as all MySQL Cluster executables, including the management node, data node, and `ndb_mgm` client binaries. This RPM is always required for installing MySQL Cluster NDB 7.2.
- If you do not have your own client application capable of administering a MySQL server, you should also obtain and install the *client* RPM (for example, `MySQL-Cluster-client-gpl-7.2.10-1.sles11.i386.rpm`), which supplies the `mysql` client

The MySQL Cluster version number in the RPM file names (shown here as 7.2.10) can vary according to the version which you are actually using. *It is very important that all of the Cluster RPMs to be installed have the same version number.* The *architecture* designation should be appropriate to the machine on which the RPM is to be installed; in particular, you should keep in mind that 64-bit RPMs cannot be used with 32-bit operating system.

Data nodes. On a computer that is to host a cluster data node it is necessary to install only the *server* RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL web site:

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.2.10-1.sles11.i386.rpm
```

Although this installs all MySQL Cluster binaries, only the program `ndbd` or `ndbmt` (both in `/usr/sbin`) is actually needed to run a MySQL Cluster data node.

SQL nodes. On each machine to be used for hosting a cluster SQL node, install the *server* RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL web site:

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.2.10-1.sles11.i386.rpm
```

This installs the MySQL server binary (`mysqld`) with NDB storage engine support in the `/usr/sbin` directory, as well as all needed MySQL Server support files. It also installs the `mysql.server` and

`mysqld_safe` startup scripts (in `/usr/share/mysql` and `/usr/bin`, respectively). The RPM installer should take care of general configuration issues (such as creating the `mysql` user and group, if needed) automatically.

To administer the SQL node (MySQL server), you should also install the `client` RPM, as shown here:

```
shell> rpm -Uhv MySQL-Cluster-client-gpl-7.2.10-1.sles11.i386.rpm
```

This installs the `mysql` client program.

Management nodes. To install the MySQL Cluster management server, it is necessary only to use the `server` RPM. Copy this RPM to the computer intended to host the management node, and then install it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the `server` RPM downloaded from the MySQL web site):

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.2.10-1.sles11.i386.rpm
```

Although this RPM installs many other files, only the management server binary `ndb_mgmd` (in the `/usr/sbin` directory) is actually required for running a management node. The `server` RPM also installs `ndb_mgm`, the NDB management client.

See [Section 2.5.1, “Installing MySQL from RPM Packages on Linux”](#), for general information about installing MySQL using RPMs supplied by Oracle.

After installing from RPM, you still need to configure the cluster as discussed in [Section 17.2.3, “Initial Configuration of MySQL Cluster”](#).



Note

A number of RPMs used by MySQL Cluster NDB 7.1 were made obsolete and discontinued in MySQL Cluster NDB 7.2. These include the former `MySQL-Cluster-clusterj`, `MySQL-Cluster-extra`, `MySQL-Cluster-management`, `MySQL-Cluster-storage`, and `MySQL Cluster-tools` RPMs; all of these have been merged into the `MySQL-Cluster-server` RPM. When upgrading from a MySQL Cluster NDB 7.1 RPM installation to MySQL Cluster NDB 7.2.3 or an earlier MySQL Cluster NDB 7.2 release, it was necessary to remove these packages manually before installing the MySQL Cluster NDB 7.2 `MySQL-Cluster-server` RPM. This issue is fixed in MySQL Cluster NDB 7.2.4 and later, where the `MySQL-Cluster-server` package specifically obsoletes the discontinued packages (BUG #13545589).

17.2.1.3. Building MySQL Cluster from Source on Linux

This section provides information about compiling MySQL Cluster on Linux and other Unix-like platforms. Building MySQL Cluster from source is similar to building the standard MySQL Server, although it differs in a few key respects discussed here. For general information about building MySQL from source, see [Section 2.10, “Installing MySQL from Source”](#). For information about compiling MySQL Cluster on Windows platforms, see [Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”](#).

Building MySQL Cluster requires using the MySQL Cluster sources. These are available from the MySQL Cluster downloads page at <http://dev.mysql.com/downloads/cluster/>. The archived source file should have a name similar to `mysql-cluster-gpl-7.2.10.tar.gz`. You can also obtain MySQL development sources from launchpad.net. Attempting to build MySQL Cluster from standard MySQL Server 5.5 sources is not supported.

The `WITH_NDBCLUSTER_STORAGE_ENGINE` [157] option for `CMake` causes the binaries for the management nodes, data nodes, and other MySQL Cluster programs to be built; it also causes `mysqld` to be compiled with `NDB` storage engine support. This option is enabled by default in the MySQL Cluster NDB 7.2 sources.



Important

Beginning with MySQL Cluster NDB 7.2.9, the `WITH_NDB_JAVA` [157] option is enabled by default. This means that, by default, if `CMake` cannot find the location of Java on your system, the configuration process fails; if you do not wish to enable Java and ClusterJ support, you must indicate this explicitly by configuring the build using `-DWITH_NDB_JAVA=OFF`. (Bug #12379735) Use `WITH_CLASSPATH` [157] to provide the Java classpath if needed.

For more information about `CMake` options specific to building MySQL Cluster, see [Options for Compiling MySQL Cluster](#).

After you have run `make` && `make install` (or your system's equivalent), the result is similar to what is obtained by unpacking a precompiled binary to the same location.

Management nodes. When building from source and running the default `make install`, the management server and management client binaries (`ndb_mgmd` and `ndb_mgm`) can be found in `/usr/local/mysql/bin`. Only `ndb_mgmd` is required to be present on a management node host; however, it is also a good idea to have `ndb_mgm` present on the same host machine. Neither of these executables requires a specific location on the host machine's file system.

Data nodes. The only executable required on a data node host is the data node binary `ndbd` or `ndbmtbd`. (`mysqld`, for example, does not have to be present on the host machine.) By default, when building from source, this file is placed in the directory `/usr/local/mysql/bin`. For installing on multiple data node hosts, only `ndbd` or `ndbmtbd` need be copied to the other host machine or machines. (This assumes that all data node hosts use the same architecture and operating system; otherwise you may need to compile separately for each different platform.) The data node binary need not be in any particular location on the host's file system, as long as the location is known.

When compiling MySQL Cluster from source, no special options are required for building multi-threaded data node binaries. Configuring the build with `NDB` storage engine support causes `ndbmtbd` to be built automatically; `make install` places the `ndbmtbd` binary in the installation `bin` directory along with `mysqld`, `ndbd`, and `ndb_mgm`.

SQL nodes. If you compile MySQL with clustering support, and perform the default installation (using `make install` as the system `root` user), `mysqld` is placed in `/usr/local/mysql/bin`. Follow the steps given in [Section 2.10, "Installing MySQL from Source"](#) to make `mysqld` ready for use. If you want to run multiple SQL nodes, you can use a copy of the same `mysqld` executable and its associated support files on several machines. The easiest way to do this is to copy the entire `/usr/local/mysql` directory and all directories and files contained within it to the other SQL node host or hosts, then repeat the steps from [Section 2.10, "Installing MySQL from Source"](#) on each machine. If you configure the build with a nondefault `PREFIX` option, you must adjust the directory accordingly.

In [Section 17.2.3, "Initial Configuration of MySQL Cluster"](#), we create configuration files for all of the nodes in our example MySQL Cluster.

17.2.2. Installing MySQL Cluster on Windows

MySQL Cluster NDB 7.2 binaries for Windows can be obtained from <http://dev.mysql.com/downloads/cluster/>. For information about installing MySQL Cluster on Windows from a binary release provided by Oracle, see [Section 17.2.2.1, "Installing MySQL Cluster on Windows from a Binary Release"](#).

It is also possible to compile and install MySQL Cluster from source on Windows using Microsoft Visual Studio. For more information, see [Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”](#).

17.2.2.1. Installing MySQL Cluster on Windows from a Binary Release

This section describes a basic installation of MySQL Cluster on Windows using a binary `no-install` MySQL Cluster release provided by Oracle, using the same 4-node setup outlined in the beginning of this section (see [Section 17.2, “MySQL Cluster Installation”](#)), as shown in the following table:

Node	IP Address
Management (MGMD) node	192.168.0.10
MySQL server (SQL) node	192.168.0.20
Data (NDBD) node "A"	192.168.0.30
Data (NDBD) node "B"	192.168.0.40

As on other platforms, the MySQL Cluster host computer running an SQL node must have installed on it a MySQL Server binary (`mysqld.exe`). You should also have the MySQL client (`mysql.exe`) on this host. For management nodes and data nodes, it is not necessary to install the MySQL Server binary; however, each management node requires the management server daemon (`ndb_mgmd.exe`); each data node requires the data node daemon (`ndbd.exe` or `ndbmt.exe`). For this example, we refer to `ndbd.exe` as the data node executable, but you can install `ndbmt.exe`, the multi-threaded version of this program, instead, in exactly the same way. You should also install the management client (`ndb_mgm.exe`) on the management server host. This section covers the steps necessary to install the correct Windows binaries for each type of MySQL Cluster node.



Note

As with other Windows programs, MySQL Cluster executables are named with the `.exe` file extension. However, it is not necessary to include the `.exe` extension when invoking these programs from the command line. Therefore, we often simply refer to these programs in this documentation as `mysqld`, `mysql`, `ndb_mgmd`, and so on. You should understand that, whether we refer (for example) to `mysqld` or `mysqld.exe`, either name means the same thing (the MySQL Server program).

For setting up a MySQL Cluster using Oracle's `no-install` binaries, the first step in the installation process is to download the latest MySQL Cluster Windows binary archive from <http://dev.mysql.com/downloads/cluster/>. This archive has a filename of the form `mysql-cluster-gpl-noinstall-ver-winarch.zip`, where *ver* is the NDB storage engine version (such as `7.2.1`), and *arch* is the architecture (`32` for 32-bit binaries, and `64` for 64-bit binaries). For example, the MySQL Cluster NDB 7.2.1 `no-install` archive for 32-bit Windows systems is named `mysql-cluster-gpl-noinstall-7.2.1-win32.zip`.

You can run 32-bit MySQL Cluster binaries on both 32-bit and 64-bit versions of Windows; however, 64-bit MySQL Cluster binaries can be used only on 64-bit versions of Windows. If you are using a 32-bit version of Windows on a computer that has a 64-bit CPU, then you must use the 32-bit MySQL Cluster binaries.

To minimize the number of files that need to be downloaded from the Internet or copied between machines, we start with the computer where you intend to run the SQL node.

SQL node. We assume that you have placed a copy of the `no-install` archive in the directory `C:\Documents and Settings\username\My Documents\Downloads` on the computer having the IP address `192.168.0.20`, where *username* is the name of the current user. (You can obtain this name using `ECHO %USERNAME%` on the command line.) To install and run MySQL Cluster executables as Windows services, this user should be a member of the `Administrators` group.

Extract all the files from the archive. The Extraction Wizard integrated with Windows Explorer is adequate for this task. (If you use a different archive program, be sure that it extracts all files and directories from the archive, and that it preserves the archive's directory structure.) When you are asked for a destination directory, enter `C:\`, which causes the Extraction Wizard to extract the archive to the directory `C:\mysql-cluster-gpl-noinstall-ver-winarch`. Rename this directory to `C:\mysql`.

It is possible to install the MySQL Cluster binaries to directories other than `C:\mysql\bin`; however, if you do so, you must modify the paths shown in this procedure accordingly. In particular, if the MySQL Server (SQL node) binary is installed to a location other than `C:\mysql` or `C:\Program Files\MySQL\MySQL Server 5.5`, or if the SQL node's data directory is in a location other than `C:\mysql\data` or `C:\Program Files\MySQL\MySQL Server 5.5\data`, extra configuration options must be used on the command line or added to the `my.ini` or `my.cnf` file when starting the SQL node. For more information about configuring a MySQL Server to run in a nonstandard location, see [Section 2.3.7, "Installing MySQL on Microsoft Windows Using a noinstall Zip Archive"](#).

For a MySQL Server with MySQL Cluster support to run as part of a MySQL Cluster, it must be started with the options `--ndbcluster` [2142] and `--ndb-connectstring` [2141]. While you can specify these options on the command line, it is usually more convenient to place them in an option file. To do this, create a new text file in Notepad or another text editor. Enter the following configuration information into this file:

```
[mysqld]
# Options for mysqld process:
ndbcluster                # run NDB storage engine
ndb-connectstring=192.168.0.10 # location of management server
```

You can add other options used by this MySQL Server if desired (see [Section 2.3.7.2, "Creating an Option File"](#)), but the file must contain the options shown, at a minimum. Save this file as `C:\mysql\my.ini`. This completes the installation and setup for the SQL node.

Data nodes. A MySQL Cluster data node on a Windows host requires only a single executable, one of either `ndbd.exe` or `ndbmttd.exe`. For this example, we assume that you are using `ndbd.exe`, but the same instructions apply when using `ndbmttd.exe`. On each computer where you wish to run a data node (the computers having the IP addresses 192.168.0.30 and 192.168.0.40), create the directories `C:\mysql`, `C:\mysql\bin`, and `C:\mysql\cluster-data`; then, on the computer where you downloaded and extracted the `no-install` archive, locate `ndbd.exe` in the `C:\mysql\bin` directory. Copy this file to the `C:\mysql\bin` directory on each of the two data node hosts.

To function as part of a MySQL Cluster, each data node must be given the address or hostname of the management server. You can supply this information on the command line using the `--ndb-connectstring` [2197] or `-c` option when starting each data node process. However, it is usually preferable to put this information in an option file. To do this, create a new text file in Notepad or another text editor and enter the following text:

```
[mysql_cluster]
# Options for data node process:
ndb-connectstring=192.168.0.10 # location of management server
```

Save this file as `C:\mysql\my.ini` on the data node host. Create another text file containing the same information and save it on as `C:\mysql\my.ini` on the other data node host, or copy the `my.ini` file from the first data node host to the second one, making sure to place the copy in the second data node's `C:\mysql` directory. Both data node hosts are now ready to be used in the MySQL Cluster, which leaves only the management node to be installed and configured.

Management node. The only executable program required on a computer used for hosting a MySQL Cluster management node is the management server program `ndb_mgmd.exe`. However, in order

to administer the MySQL Cluster once it has been started, you should also install the MySQL Cluster management client program `ndb_mgm.exe` on the same machine as the management server. Locate these two programs on the machine where you downloaded and extracted the `no-install` archive; this should be the directory `C:\mysql\bin` on the SQL node host. Create the directory `C:\mysql\bin` on the computer having the IP address 192.168.0.10, then copy both programs to this directory.

You should now create two configuration files for use by `ndb_mgmd.exe`:

1. A local configuration file to supply configuration data specific to the management node itself. Typically, this file needs only to supply the location of the MySQL Cluster global configuration file (see item 2).

To create this file, start a new text file in Notepad or another text editor, and enter the following information:

```
[mysql_cluster]
# Options for management node process
config-file=C:/mysql/bin/config.ini
```

Save this file as the plaintext file `C:\mysql\bin\my.ini`.

2. A global configuration file from which the management node can obtain configuration information governing the MySQL Cluster as a whole. At a minimum, this file must contain a section for each node in the MySQL Cluster, and the IP addresses or hostnames for the management node and all data nodes (`HostName` configuration parameter). It is also advisable to include the following additional information:

- The IP address or hostname of any SQL nodes
- The data memory and index memory allocated to each data node (`DataMemory` [2043] and `IndexMemory` [2044] configuration parameters)
- The number of replicas, using the `NoOfReplicas` [2041] configuration parameter (see [Section 17.1.2, "MySQL Cluster Nodes, Node Groups, Replicas, and Partitions"](#))
- The directory where each data node stores its data and log file, and the directory where the management node keeps its log files (in both cases, the `DataDir` [2042] configuration parameter)

Create a new text file using a text editor such as Notepad, and input the following information:

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2          # Number of replicas
DataDir=C:/mysql/bin/cluster-data # Directory for each data node's data files
                                # Forward slashes used in directory path,
                                # rather than backslashes. This is correct;
                                # see Important note in text
DataMemory=80M         # Memory allocated to data storage
IndexMemory=18M        # Memory allocated to index storage
                                # For DataMemory and IndexMemory, we have used the
                                # default values. Since the "world" database takes up
                                # only about 500KB, this should be more than enough for
                                # this example Cluster setup.

[ndb_mgmd]
# Management process options:
HostName=192.168.0.10   # Hostname or IP address of management node
DataDir=C:/mysql/bin/cluster-logs # Directory for management node log files

[ndbd]
# Options for data node "A":
                                # (one [ndbd] section per data node)
```

```

HostName=192.168.0.30      # Hostname or IP address

[ndbd]
# Options for data node "B":
HostName=192.168.0.40      # Hostname or IP address

[mysqld]
# SQL node options:
HostName=192.168.0.20      # Hostname or IP address

```

Save this file as the plaintext file `C:\mysql\bin\config.ini`.



Important

A single backslash character (\) cannot be used when specifying directory paths in program options or configuration files used by MySQL Cluster on Windows. Instead, you must either escape each backslash character with a second backslash (\\), or replace the backslash with a forward slash character (/). For example, the following line from the `[ndb_mgmd]` section of a MySQL Cluster `config.ini` file does not work:

```
DataDir=C:\mysql\bin\cluster-logs
```

Instead, you may use either of the following:

```
DataDir=C:\\mysql\\bin\\cluster-logs # Escaped backslashes
```

```
DataDir=C:/mysql/bin/cluster-logs # Forward slashes
```

For reasons of brevity and legibility, we recommend that you use forward slashes in directory paths used in MySQL Cluster program options and configuration files on Windows.

17.2.2.2. Compiling and Installing MySQL Cluster from Source on Windows

Oracle provides precompiled MySQL Cluster binaries for Windows which should be adequate for most users. However, if you wish, it is also possible to compile MySQL Cluster for Windows from source code. The procedure for doing this is almost identical to the procedure used to compile the standard MySQL Server binaries for Windows, and uses the same tools. However, there are two major differences:

- To build MySQL Cluster, you must use the MySQL Cluster sources, which you can obtain from <http://dev.mysql.com/downloads/cluster/>.

Attempting to build MySQL Cluster from the source code for the standard MySQL Server is likely not to be successful, and is not supported by Oracle.

- You must configure the build using the `WITH_NDBCLUSTER_STORAGE_ENGINE [157]` or `WITH_NDBCLUSTER [157]` option in addition to any other build options you wish to use with `CMake`. (`WITH_NDBCLUSTER` is supported as an alias for `WITH_NDBCLUSTER_STORAGE_ENGINE`, and works in exactly the same way.)



Important

Beginning with MySQL Cluster NDB 7.2.9, the `WITH_NDB_JAVA [157]` option is enabled by default. This means that, by default, if `CMake` cannot find the location of Java on your system, the configuration process fails; if you do not wish to enable Java and ClusterJ support, you must indicate this explicitly by configuring the build using `-DWITH_NDB_JAVA=OFF`. (Bug #12379735) Use `WITH_CLASSPATH [157]` to provide the Java classpath if needed.

For more information about [CMake](#) options specific to building MySQL Cluster, see [Options for Compiling MySQL Cluster](#).

Once the build process is complete, you can create a Zip archive containing the compiled binaries; [Section 2.10.2, “Installing MySQL from a Standard Source Distribution”](#) provides the commands needed to perform this task on Windows systems. The MySQL Cluster binaries can be found in the `bin` directory of the resulting archive, which is equivalent to the `no-install` archive, and which can be installed and configured in the same manner. For more information, see [Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”](#).

17.2.2.3. Initial Startup of MySQL Cluster on Windows

Once the MySQL Cluster executables and needed configuration files are in place, performing an initial start of the cluster is simply a matter of starting the MySQL Cluster executables for all nodes in the cluster. Each cluster node process must be started separately, and on the host computer where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes.

1. On the management node host, issue the following command from the command line to start the management node process:

```
C:\mysql\bin> ndb_mgmd
2010-06-23 07:53:34 [MgmtSrvr] INFO -- NDB Cluster Management Server. mysql-5.5.29-ndb-7.2.10
2010-06-23 07:53:34 [MgmtSrvr] INFO -- Reading cluster configuration from 'config.ini'
```

The management node process continues to print logging output to the console. This is normal, because the management node is not running as a Windows service. (If you have used MySQL Cluster on a Unix-like platform such as Linux, you may notice that the management node's default behavior in this regard on Windows is effectively the opposite of its behavior on Unix systems, where it runs by default as a Unix daemon process. This behavior is also true of MySQL Cluster data node processes running on Windows.) For this reason, do not close the window in which `ndb_mgmd.exe` is running; doing so kills the management node process. (See [Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”](#), where we show how to install and run MySQL Cluster processes as Windows services.)

The required `-f` option tells the management node where to find the global configuration file (`config.ini`). The long form of this option is `--config-file` [\[2188\]](#).



Important

A MySQL Cluster management node caches the configuration data that it reads from `config.ini`; once it has created a configuration cache, it ignores the `config.ini` file on subsequent starts unless forced to do otherwise. This means that, if the management node fails to start due to an error in this file, you must make the management node re-read `config.ini` after you have corrected any errors in it. You can do this by starting `ndb_mgmd.exe` with the `--reload` [\[2191\]](#) or `--initial` [\[2190\]](#) option on the command line. Either of these options works to refresh the configuration cache.

It is not necessary or advisable to use either of these options in the management node's `my.ini` file.

For additional information about options which can be used with `ndb_mgmd`, see [Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”](#), as well as [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

2. On each of the data node hosts, run the command shown here to start the data node processes:

```
C:\mysql\bin> ndbd
2010-06-23 07:53:46 [ndbd] INFO -- Configuration fetched from 'localhost:1186', generation: 1
```

In each case, the first line of output from the data node process should resemble what is shown in the preceding example, and is followed by additional lines of logging output. As with the management node process, this is normal, because the data node is not running as a Windows service. For this reason, do not close the console window in which the data node process is running; doing so kills `ndbd.exe`. (For more information, see [Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”](#).)

- Do not start the SQL node yet; it cannot connect to the cluster until the data nodes have finished starting, which may take some time. Instead, in a new console window on the management node host, start the MySQL Cluster management client `ndb_mgm.exe`, which should be in `C:\mysql\bin` on the management node host. (Do not try to re-use the console window where `ndb_mgmd.exe` is running by typing **CTRL+C**, as this kills the management node.) The resulting output should look like this:

```
C:\mysql\bin> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm>
```

When the prompt `ndb_mgm>` appears, this indicates that the management client is ready to receive MySQL Cluster management commands. You can observe the status of the data nodes as they start by entering `ALL STATUS` at the management client prompt. This command causes a running report of the data nodes's startup sequence, which should look something like this:

```
ndb_mgm> ALL STATUS
Connected to Management Server at: localhost:1186
Node 2: starting (Last completed phase 3) (mysql-5.5.29-ndb-7.2.10)
Node 3: starting (Last completed phase 3) (mysql-5.5.29-ndb-7.2.10)

Node 2: starting (Last completed phase 4) (mysql-5.5.29-ndb-7.2.10)
Node 3: starting (Last completed phase 4) (mysql-5.5.29-ndb-7.2.10)

Node 2: Started (version 7.2.10)
Node 3: Started (version 7.2.10)

ndb_mgm>
```



Note

Commands issued in the management client are not case-sensitive; we use uppercase as the canonical form of these commands, but you are not required to observe this convention when inputting them into the `ndb_mgm` client. For more information, see [Section 17.5.2, “Commands in the MySQL Cluster Management Client”](#).

The output produced by `ALL STATUS` is likely to vary from what is shown here, according to the speed at which the data nodes are able to start, the release version number of the MySQL Cluster software you are using, and other factors. What is significant is that, when you see that both data nodes have started, you are ready to start the SQL node.

You can leave `ndb_mgm.exe` running; it has no negative impact on the performance of the MySQL Cluster, and we use it in the next step to verify that the SQL node is connected to the cluster after you have started it.

- On the computer designated as the SQL node host, open a console window and navigate to the directory where you unpacked the MySQL Cluster binaries (if you are following our example, this is `C:\mysql\bin`).

Start the SQL node by invoking `mysqld.exe` from the command line, as shown here:

```
C:\mysql\bin> mysqld --console
```

The `--console` [438] option causes logging information to be written to the console, which can be helpful in the event of problems. (Once you are satisfied that the SQL node is running in a satisfactory manner, you can stop it and restart it out without the `--console` [438] option, so that logging is performed normally.)

In the console window where the management client (`ndb_mgm.exe`) is running on the management node host, enter the `SHOW` command, which should produce output similar to what is shown here:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 @192.168.0.30 (Version: 5.5.29-ndb-7.2.10, Nodegroup: 0, Master)
id=3 @192.168.0.40 (Version: 5.5.29-ndb-7.2.10, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.10 (Version: 5.5.29-ndb-7.2.10)

[mysqld(API)] 1 node(s)
id=4 @192.168.0.20 (Version: 5.5.29-ndb-7.2.10)
```

You can also verify that the SQL node is connected to the MySQL Cluster in the `mysql` client (`mysql.exe`) using the `SHOW ENGINE NDB STATUS` [1481] statement.

You should now be ready to work with database objects and data using MySQL Cluster's `NDBCLUSTER` storage engine. See [Section 17.2.5, "MySQL Cluster Example with Tables and Data"](#), for more information and examples.

You can also install `ndb_mgmd.exe`, `ndbd.exe`, and `ndbmt.d.exe` as Windows services. For information on how to do this, see [Section 17.2.2.4, "Installing MySQL Cluster Processes as Windows Services"](#).

17.2.2.4. Installing MySQL Cluster Processes as Windows Services

Once you are satisfied that MySQL Cluster is running as desired, you can install the management nodes and data nodes as Windows services, so that these processes are started and stopped automatically whenever Windows is started or stopped. This also makes it possible to control these processes from the command line with the appropriate `NET START` or `NET STOP` command, or using the Windows graphical `Services` utility.

Installing programs as Windows services usually must be done using an account that has Administrator rights on the system.

To install the management node as a service on Windows, invoke `ndb_mgmd.exe` from the command line on the machine hosting the management node, using the `--install` [2193] option, as shown here:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --install
Installing service 'MySQL Cluster Management Server'
as "C:\mysql\bin\ndbd.exe" "--service=ndb_mgmd"
Service successfully installed.
```



Important

When installing a MySQL Cluster program as a Windows service, you should always specify the complete path; otherwise the service installation may fail with the error `The system cannot find the file specified`.

The `--install [2193]` option must be used first, ahead of any other options that might be specified for `ndb_mgmd.exe`. However, it is preferable to specify such options in an options file instead. If your options file is not in one of the default locations as shown in the output of `ndb_mgmd.exe --help [2241]`, you can specify the location using the `--config-file [2188]` option.

Now you should be able to start and stop the management server like this:

```
C:\> NET START ndb_mgmd
The MySQL Cluster Management Server service is starting.
The MySQL Cluster Management Server service was started successfully.

C:\> NET STOP ndb_mgmd
The MySQL Cluster Management Server service is stopping..
The MySQL Cluster Management Server service was stopped successfully.
```

You can also start or stop the management server as a Windows service using the descriptive name, as shown here:

```
C:\> NET START 'MySQL Cluster Management Server'
The MySQL Cluster Management Server service is starting.
The MySQL Cluster Management Server service was started successfully.

C:\> NET STOP 'MySQL Cluster Management Server'
The MySQL Cluster Management Server service is stopping..
The MySQL Cluster Management Server service was stopped successfully.
```

However, it is usually simpler to specify a short service name or to permit the default service name to be used when installing the service, and then reference that name when starting or stopping the service. To specify a service name other than `ndb_mgmd`, append it to the `--install [2193]` option, as shown in this example:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --install=mgmdl
Installing service 'MySQL Cluster Management Server'
  as 'C:\mysql\bin\ndb_mgmd.exe' "--service=mgmdl"
Service successfully installed.
```

Now you should be able to start or stop the service using the name you have specified, like this:

```
C:\> NET START mgmdl
The MySQL Cluster Management Server service is starting.
The MySQL Cluster Management Server service was started successfully.

C:\> NET STOP mgmdl
The MySQL Cluster Management Server service is stopping..
The MySQL Cluster Management Server service was stopped successfully.
```

To remove the management node service, invoke `ndb_mgmd.exe` with the `--remove [2194]` option, as shown here:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --remove
Removing service 'MySQL Cluster Management Server'
Service successfully removed.
```

If you installed the service using a service name other than the default, you can remove the service by passing this name as the value of the `--remove [2194]` option, like this:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --remove=mgmdl
Removing service 'mgmdl'
Service successfully removed.
```

Installation of a MySQL Cluster data node process as a Windows service can be done in a similar fashion, using the `--install [2180]` option for `ndbd.exe` (or `ndbmt.d.exe`), as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --install
Installing service 'MySQL Cluster Data Node Daemon' as '"C:\mysql\bin\ndbd.exe" "--service=ndbd"'
Service successfully installed.
```

Now you can start or stop the data node using either the default service name or the descriptive name with `net start` or `net stop`, as shown in the following example:

```
C:\> NET START ndbd
The MySQL Cluster Data Node Daemon service is starting.
The MySQL Cluster Data Node Daemon service was started successfully.

C:\> NET STOP ndbd
The MySQL Cluster Data Node Daemon service is stopping..
The MySQL Cluster Data Node Daemon service was stopped successfully.

C:\> NET START 'MySQL Cluster Data Node Daemon'
The MySQL Cluster Data Node Daemon service is starting.
The MySQL Cluster Data Node Daemon service was started successfully.

C:\> NET STOP 'MySQL Cluster Data Node Daemon'
The MySQL Cluster Data Node Daemon service is stopping..
The MySQL Cluster Data Node Daemon service was stopped successfully.
```

To remove the data node service, invoke `ndbd.exe` with the `--remove` [2181] option, as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --remove
Removing service 'MySQL Cluster Data Node Daemon'
Service successfully removed.
```

As with `ndb_mgmd.exe` (and `mysqld.exe`), when installing `ndbd.exe` as a Windows service, you can also specify a name for the service as the value of `--install` [2180], and then use it when starting or stopping the service, like this:

```
C:\> C:\mysql\bin\ndbd.exe --install=dnodel
Installing service 'dnodel' as '"C:\mysql\bin\ndbd.exe" "--service=dnodel"'
Service successfully installed.

C:\> NET START dnodel
The MySQL Cluster Data Node Daemon service is starting.
The MySQL Cluster Data Node Daemon service was started successfully.

C:\> NET STOP dnodel
The MySQL Cluster Data Node Daemon service is stopping..
The MySQL Cluster Data Node Daemon service was stopped successfully.
```

If you specified a service name when installing the data node service, you can use this name when removing it as well, by passing it as the value of the `--remove` [2181] option, as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --remove=dnodel
Removing service 'dnodel'
Service successfully removed.
```

Installation of the SQL node as a Windows service, starting the service, stopping the service, and removing the service are done in a similar fashion, using `mysqld --install`, `NET START`, `NET STOP`, and `mysqld --remove` [458]. For additional information, see [Section 2.3.7.7, “Starting MySQL as a Windows Service”](#).

17.2.3. Initial Configuration of MySQL Cluster

For our four-node, four-host MySQL Cluster, it is necessary to write four configuration files, one per node host.

- Each data node or SQL node requires a `my.cnf` file that provides two pieces of information: a `connectstring` that tells the node where to find the management node, and a line telling the MySQL server on this host (the machine hosting the data node) to enable the `NDBCLUSTER` storage engine.

For more information on connectstrings, see [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#).

- The management node needs a `config.ini` file telling it how many replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the data nodes and SQL nodes. The `my.cnf` file needed for the data nodes is fairly simple. The configuration file should be located in the `/etc` directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

```
shell> vi /etc/my.cnf
```



Note

We show `vi` being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, `my.cnf` should look like this:

```
[mysqld]
# Options for mysqld process:
ndbcluster                # run NDB storage engine

[mysql_cluster]
# Options for MySQL Cluster processes:
ndb-connectstring=192.168.0.10 # location of management server
```

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.



Important

Once you have started a `mysqld` process with the `ndbcluster` and `ndb-connectstring` parameters in the `[mysqld]` and `[mysql_cluster]` sections of the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. This is by design.

Configuring the management node. The first step in configuring the management node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as `root`):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our representative setup, the `config.ini` file should read as follows:

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2      # Number of replicas
DataMemory=80M     # How much memory to allocate for data storage
IndexMemory=18M    # How much memory to allocate for index storage
                   # For DataMemory and IndexMemory, we have used the
                   # default values. Since the "world" database takes up
                   # only about 500KB, this should be more than enough for
                   # this example Cluster setup.
```

```
[tcp default]
# TCP/IP options:
portnumber=2202 # This the default; however, you can use any
                # port that is free for all the hosts in the cluster
                # Note: It is recommended that you do not specify the port
                # number at all and simply allow the default value to be used
                # instead

[ndb_mgmd]
# Management process options:
hostname=192.168.0.10 # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster # Directory for MGM node log files

[ndbd]
# Options for data node "A":
                # (one [ndbd] section per data node)
hostname=192.168.0.30 # Hostname or IP address
datadir=/usr/local/mysql/data # Directory for this data node's data files

[ndbd]
# Options for data node "B":
hostname=192.168.0.40 # Hostname or IP address
datadir=/usr/local/mysql/data # Directory for this data node's data files

[mysqld]
# SQL node options:
hostname=192.168.0.20 # Hostname or IP address
                # (additional mysqld connections can be
                # specified for this node for various
                # purposes such as running ndb_restore)
```



Note

The `world` database can be downloaded from <http://dev.mysql.com/doc/>, where it can be found listed under “Examples”.

After all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss how this is done in [Section 17.2.4, “Initial Startup of MySQL Cluster”](#).

For more detailed information about the available MySQL Cluster configuration parameters and their uses, see [Section 17.3.2, “MySQL Cluster Configuration Files”](#), and [Section 17.3, “MySQL Cluster Configuration”](#). For configuration of MySQL Cluster as relates to making backups, see [Section 17.5.3.3, “Configuration for MySQL Cluster Backups”](#).



Note

The default port for Cluster management nodes is 1186; the default port for data nodes is 2202. However, the cluster can automatically allocate ports for data nodes from those that are already free.

17.2.4. Initial Startup of MySQL Cluster

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the management node process:

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

The first time that it is started, `ndb_mgmd` must be told where to find its configuration file, using the `-f` or `--config-file` [2188] option. (See [Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”](#), for details.)

For additional options which can be used with `ndb_mgmd`, see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

2. On each of the data node hosts, run this command to start the `ndbd` process:

```
shell> ndbd
```

3. If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the supplied startup script to start the MySQL server process on the SQL node.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the `ndb_mgm` management node client. The output should look like that shown here, although you might see some slight differences in the output depending upon the exact version of MySQL that you are using:

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)]      2 node(s)
id=2      @192.168.0.30 (Version: 5.5.29-ndb-7.2.10, Nodegroup: 0, Master)
id=3      @192.168.0.40 (Version: 5.5.29-ndb-7.2.10, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1      @192.168.0.10 (Version: 5.5.29-ndb-7.2.10)

[mysqld(API)]   1 node(s)
id=4      @192.168.0.20 (Version: 5.5.29-ndb-7.2.10)
```

The SQL node is referenced here as `[mysqld(API)]`, which reflects the fact that the `mysqld` process is acting as a MySQL Cluster API node.



Note

The IP address shown for a given MySQL Cluster SQL or other API node in the output of `SHOW` is the address used by the SQL or API node to connect to the cluster data nodes, and not to any management node.

You should now be ready to work with databases, tables, and data in MySQL Cluster. See [Section 17.2.5, “MySQL Cluster Example with Tables and Data”](#), for a brief discussion.

17.2.5. MySQL Cluster Example with Tables and Data



Note

The information in this section applies to MySQL Cluster running on both Unix and Windows platforms.

Working with database tables and data in MySQL Cluster is not much different from doing so in standard MySQL. There are two key points to keep in mind:

- For a table to be replicated in the cluster, it must use the `NDBCLUSTER` storage engine. To specify this, use the `ENGINE=NDBCLUSTER` or `ENGINE=NDB` option when creating the table:

```
CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;
```

Alternatively, for an existing table that uses a different storage engine, use `ALTER TABLE` to change the table to use `NDBCLUSTER`:

```
ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
```

- Every `NDBCLUSTER` table has a primary key. If no primary key is defined by the user when a table is created, the `NDBCLUSTER` storage engine automatically generates a hidden one. Such a key takes up space just as does any other table index. (It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of `mysqldump`, you can open the SQL script in a text editor and add the `ENGINE` option to any table creation statements, or replace any existing `ENGINE` options. Suppose that you have the `world` sample database on another MySQL server that does not support MySQL Cluster, and you want to export the `City` table:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

The resulting `city_table.sql` file will contain this table creation statement (and the `INSERT` statements necessary to import the table data):

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);(remaining INSERT statements omitted)
```

You need to make sure that MySQL uses the `NDBCLUSTER` storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition *before* importing it into the Cluster database. Using the `City` table as an example, modify the `ENGINE` option of the definition as follows:

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of `TYPE=engine_name` or `ENGINE=engine_name` with `ENGINE=NDBCLUSTER`. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use `ALTER TABLE` to change their storage engine. The particulars are given later in this section.

Assuming that you have already created a database named `world` on the SQL node of the cluster, you can then use the `mysql` command-line client to read `city_table.sql`, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address `192.168.0.20`).

To create a copy of the entire `world` database on the SQL node, use `mysqldump` on the noncluster server to export the database to a file named `world.sql`; for example, in the `/tmp` directory. Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

```
shell> mysql world < /tmp/world.sql
```

If you save the file to a different location, adjust the preceding instructions accordingly.

Running `SELECT` queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the `root` password at the `Enter password:` prompt):

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.5.29-ndb-7.2.10

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

We simply use the MySQL server's `root` account and assume that you have followed the standard security precautions for installing a MySQL server, including setting a strong `root` password. For more information, see [Section 2.11.2, “Securing the Initial MySQL Accounts”](#).

It is worth taking into account that Cluster nodes do *not* make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the `root` account) effects only applications that access the SQL node, not interaction between nodes. See [Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”](#), for more information.

If you did not modify the `ENGINE` clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Selecting a database and running a `SELECT` query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----+-----+
| Name      | Population |
+-----+-----+
| Bombay    | 10500000  |
| Seoul     | 9981619   |
| São Paulo | 9968485   |
| Shanghai  | 9696300   |
| Jakarta   | 9604900   |
+-----+-----+
5 rows in set (0.34 sec)
```

```
mysql> \q
Bye

shell>
```

Applications that use MySQL can employ standard APIs to access [NDB](#) tables. It is important to remember that your application must access the SQL node, and not the management or data nodes. This brief example shows how we might execute the [SELECT](#) statement just shown by using the PHP 5.X [mysqli](#) extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('192.168.0.20', 'root', 'root_password', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";

  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing="1">
  <tbody>
    <tr>
      <th width="10%">City</th>
      <th>Population</th>
    </tr>
  </tbody>
<?
  # then display the results...
  while($row = $result->fetch_object())
    printf("<tr>\n  <td align=\"center\">%s</td><td>%d</td>\n</tr>\n",
          $row->Name, $row->Population);
?>
  </tbody>
</table>
<?
  # ...and verify the number of rows that were retrieved
  printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
  }
  else
    # otherwise, tell us what went wrong
    echo mysqli_error();

  # free the result set and the mysqli connection object
  $result->close();
  $link->close();
?>
</body>
</html>
```

We assume that the process running on the Web server can reach the IP address of the SQL node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

17.2.6. Safe Shutdown and Restart of MySQL Cluster

To shut down the cluster, enter the following command in a shell on the machine hosting the management node:

```
shell> ndb_mgm -e shutdown
```

The `-e` option here is used to pass a command to the `ndb_mgm` client from the shell. (See [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#), for more information about this option.) The command causes the `ndb_mgm`, `ndb_mgmd`, and any `ndbd` or `ndbmt` processes to terminate gracefully. Any SQL nodes can be terminated using `mysqladmin shutdown` and other means. On Windows platforms, assuming that you have installed the SQL node as a Windows service, you can use `NET STOP MYSQL`.

To restart the cluster on Unix platforms, run these commands:

- On the management host (192.168.0.10 in our example setup):

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

- On each of the data node hosts (192.168.0.30 and 192.168.0.40):

```
shell> ndbd
```

- Use the `ndb_mgm` client to verify that both data nodes have started successfully.
- On the SQL host (192.168.0.20):

```
shell> mysql_safe &
```

On Windows platforms, assuming that you have installed all MySQL Cluster processes as Windows services using the default service names (see [Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”](#)), you can restart the cluster as follows:

- On the management host (192.168.0.10 in our example setup), execute the following command:

```
C:\> NET START ndb_mgmd
```

- On each of the data node hosts (192.168.0.30 and 192.168.0.40), execute the following command:

```
C:\> NET START ndbd
```

- On the management node host, use the `ndb_mgm` client to verify that the management node and both data nodes have started successfully (see [Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”](#)).

- On the SQL node host (192.168.0.20), execute the following command:

```
C:\> NET START mysql
```

In a production setting, it is usually not desirable to shut down the cluster completely. In many cases, even when making configuration changes, or performing upgrades to the cluster hardware or software (or both), which require shutting down individual host machines, it is possible to do so without shutting down the cluster as a whole by performing a *rolling restart* of the cluster. For more information about doing this, see [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#).

17.2.7. Upgrading and Downgrading MySQL Cluster NDB 7.2

This section provides information about MySQL Cluster software and table file compatibility between different MySQL Cluster NDB 7.2 releases with regard to performing upgrades and downgrades as well as compatibility matrices and notes. You are expected already to be familiar with installing and configuring a MySQL Cluster prior to attempting an upgrade or downgrade. See [Section 17.3, “MySQL Cluster Configuration”](#).



Important

Only compatibility between MySQL versions with regard to `NDBCLUSTER` is taken into account in this section, and there are likely other issues to be considered. *As with any other MySQL software upgrade or downgrade, you are strongly encouraged to review the relevant portions of the MySQL Manual for the MySQL versions from which and to which you intend to migrate, before attempting an upgrade or downgrade of the MySQL Cluster software.* This is especially true when planning a migration from MySQL Cluster NDB 7.1 (or earlier) to MySQL Cluster NDB 7.2, since the version of the underlying MySQL Server also changes from MySQL 5.1 to MySQL 5.5. See [Section 2.12.1, “Upgrading MySQL”](#).

Versions supported. The following versions of MySQL Cluster are supported for upgrades to MySQL Cluster NDB 7.2 (7.2.4 and later):

- MySQL Cluster NDB 7.1 GA releases (7.1.3 and later)
- MySQL Cluster NDB 7.0 GA releases (7.0.5 and later)
- MySQL Cluster NDB 6.3 GA releases (6.3.8 and later) that can be upgraded to MySQL Cluster NDB 7.1

For information about upgrades and downgrades in previous MySQL Cluster release series, see [Upgrade and Downgrade Compatibility: MySQL Cluster NDB 6.x](#), and [Upgrade and downgrade compatibility: MySQL Cluster NDB 7.x](#).

NDB API, ClusterJ, and other applications used with recent releases of MySQL Cluster NDB 6.3 and later should continue to work with MySQL Cluster NDB 7.2.4 and later without rewriting or recompiling.

In MySQL Cluster NDB 7.2, the default values for a number of node configuration parameters have changed. See [Improved default values for data node configuration parameters](#), for a listing of these.

In MySQL Cluster NDB 7.2.7 and later, the size of the hash map is 3840 LDM threads, an increase from 240 in previous versions. When upgrading a MySQL Cluster from MySQL Cluster NDB 7.2.6 and earlier to MySQL Cluster NDB 7.2.9 or later, you can modify existing tables online to take advantage of the new size: following the upgrade, increase the number of fragments by (for example) adding new data nodes to the cluster, and then execute `ALTER ONLINE TABLE ... REORGANIZE PARTITION` on any tables that were created in the older version. Following this, these tables can use the larger hash map size.

17.3. MySQL Cluster Configuration

A MySQL server that is part of a MySQL Cluster differs in one chief respect from a normal (nonclustered) MySQL server, in that it employs the `NDBCLUSTER` storage engine. This engine is also referred to simply as `NDB`, and the two forms of the name are synonymous.

To avoid unnecessary allocation of resources, the server is configured by default with the `NDB` storage engine disabled. To enable `NDB`, you must modify the server's `my.cnf` configuration file, or start the server with the `--ndbcluster [2142]` option.

For more information about `--ndbcluster` [2142] and other MySQL server options specific to MySQL Cluster, see [Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”](#).

The MySQL server is a part of the cluster, so it also must know how to access an MGM node to obtain the cluster configuration data. The default behavior is to look for the MGM node on `localhost`. However, should you need to specify that its location is elsewhere, this can be done in `my.cnf` or on the MySQL server command line. Before the `NDB` storage engine can be used, at least one MGM node must be operational, as well as any desired data nodes.

`NDB`, the MySQL Cluster storage engine, is available in binary distributions for Linux, Mac OS X, Solaris, and Windows. We are working to support MySQL Cluster on all operating systems supported by the MySQL Server. For information about installing MySQL Cluster, see [Section 17.2, “MySQL Cluster Installation”](#).

17.3.1. Quick Test Setup of MySQL Cluster

To familiarize you with the basics, we will describe the simplest possible configuration for a functional MySQL Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as `/var/lib/mysql-cluster`, by executing the following command as the system `root` user:

```
shell> mkdir /var/lib/mysql-cluster
```

In this directory, create a file named `config.ini` that contains the following information. Substitute appropriate values for `HostName` and `DataDir` as necessary for your system.

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: [db], [api], and [mgm] are aliases for [ndbd], [mysqld], and [ndb_mgmd],
# respectively. [db] is deprecated and should not be used in new installations.

[ndbd default]
NoOfReplicas= 1

[mysqld default]
[ndb_mgmd default]
[tcp default]

[ndb_mgmd]
HostName= myhost.example.com

[ndbd]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[mysqld]
[mysqld]
[mysqld]
```

You can now start the `ndb_mgmd` management server. By default, it attempts to read the `config.ini` file in its current working directory, so change location into the directory where the file is located and then invoke `ndb_mgmd`:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Then start a single data node by running `ndbd`:

```
shell> ndbd
```

For command-line options which can be used when starting `ndbd`, see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

By default, `ndbd` looks for the management server at `localhost` on port 1186.



Note

If you have installed MySQL from a binary tarball, you will need to specify the path of the `ndb_mgmd` and `ndbd` servers explicitly. (Normally, these will be found in `/usr/local/mysql/bin`.)

Finally, change location to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```
shell> mysqld_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

If all has gone well so far, you now can start using the cluster. Connect to the server and verify that the `NDBCLUSTER` storage engine is enabled:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.5.31

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
***** 12. row *****
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
***** 13. row *****
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...

```

The row numbers shown in the preceding example output may be different from those shown on your system, depending upon how your server is configured.

Try to create an `NDBCLUSTER` table:

```
shell> mysql
mysql> USE test;
Database changed
```

```
mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
***** 1. row *****
      Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

To check that your nodes were set up properly, start the management client:

```
shell> ndb_mgm
```

Use the `SHOW` command from within the management client to obtain a report on the cluster's status:

```
ndb_mgm> SHOW
Cluster Configuration
-----
[ndbd(NDB)]      1 node(s)
id=2      @127.0.0.1 (Version: 5.5.29-ndb-7.2.10, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1      @127.0.0.1 (Version: 5.5.29-ndb-7.2.10)

[mysqld(API)]   3 node(s)
id=3      @127.0.0.1 (Version: 5.5.29-ndb-7.2.10)
id=4      (not connected, accepting connect from any host)
id=5      (not connected, accepting connect from any host)
```

At this point, you have successfully set up a working MySQL Cluster. You can now store data in the cluster by using any table created with `ENGINE=NDBCLUSTER` or its alias `ENGINE=NDB`.

17.3.2. MySQL Cluster Configuration Files

Configuring MySQL Cluster requires working with two files:

- `my.cnf`: Specifies options for all MySQL Cluster executables. This file, with which you should be familiar with from previous work with MySQL, must be accessible by each executable running in the cluster.
- `config.ini`: This file, sometimes known as the *global configuration file*, is read only by the MySQL Cluster management server, which then distributes the information contained therein to all processes participating in the cluster. `config.ini` contains a description of each node involved in the cluster. This includes configuration parameters for data nodes and configuration parameters for connections between all nodes in the cluster. For a quick reference to the sections that can appear in this file, and what sorts of configuration parameters may be placed in each section, see [Sections of the config.ini File](#).

Caching of configuration data. In MySQL Cluster NDB 7.2, MySQL Cluster uses *stateful configuration*. Rather than reading the global configuration file every time the management server is restarted, the management server caches the configuration the first time it is started, and thereafter, the global configuration file is read only when one of the following conditions is true:

- **The management server is started using the `--initial` option.** In this case, the global configuration file is re-read, any existing cache files are deleted, and the management server creates a new configuration cache.
- **The management server is started using the `--reload` option.** In this case, the management server compares its cache with the global configuration file. If they differ, the management server

creates a new configuration cache; any existing configuration cache is preserved, but not used. If the management server's cache and the global configuration file contain the same configuration data, then the existing cache is used, and no new cache is created.

- **The management server is started using a `--config-cache` option.** This option can be used to force the management server to bypass configuration caching altogether. In this case, the management server ignores any configuration files that may be present, always reading its configuration data from the `config.ini` file instead.
- **No configuration cache is found.** In this case, the management server reads the global configuration file and creates a cache containing the same configuration data as found in the file.

Configuration cache files. The management server by default creates configuration cache files in a directory named `mysql-cluster` in the MySQL installation directory. (If you build MySQL Cluster from source on a Unix system, the default location is `/usr/local/mysql-cluster`.) This can be overridden at run time by starting the management server with the `--configdir` option. Configuration cache files are binary files named according to the pattern `ndb_node_id_config.bin.seq_id`, where `node_id` is the management server's node ID in the cluster, and `seq_id` is a cache identifier. Cache files are numbered sequentially using `seq_id`, in the order in which they are created. The management server uses the latest cache file as determined by the `seq_id`.



Note

It is possible to roll back to a previous configuration by deleting later configuration cache files, or by renaming an earlier cache file so that it has a higher `seq_id`. However, since configuration cache files are written in a binary format, you should not attempt to edit their contents by hand.

For more information about the `--configdir`, `--initial`, and `--reload` options for the MySQL Cluster management server, see [Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”](#).

We are continuously making improvements in Cluster configuration and attempting to simplify this process. Although we strive to maintain backward compatibility, there may be times when introduce an incompatible change. In such cases we will try to let Cluster users know in advance if a change is not backward compatible. If you find such a change and we have not documented it, please report it in the MySQL bugs database using the instructions given in [Section 1.7, “How to Report Bugs or Problems”](#).

17.3.2.1. MySQL Cluster Configuration: Basic Example

To support MySQL Cluster, you will need to update `my.cnf` as shown in the following example. You may also specify these parameters on the command line when invoking the executables.



Note

The options shown here should not be confused with those that are used in `config.ini` global configuration files. Global configuration options are discussed later in this section.

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid in MySQL 5.5)

# enable ndbcluster storage engine, and provide connectstring for
# management server host (default port is 1186)
[mysqld]
ndbcluster
```

```

ndb-connectstring=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini

```

(For more information on connectstrings, see [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#).)

```

# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connectstring for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186

```



Important

Once you have started a `mysqld` process with the `NDBCLUSTER` and `ndb-connectstring` parameters in the `[mysqld]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design.*

You may also use a separate `[mysql_cluster]` section in the cluster `my.cnf` file for settings to be read and used by all executables:

```

# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186

```

For additional `NDB` variables that can be set in the `my.cnf` file, see [Section 17.3.4.3, “MySQL Cluster System Variables”](#).

The MySQL Cluster global configuration file is named `config.ini` by default. It is read by `ndb_mgmd` at startup and can be placed anywhere. Its location and name are specified by using `--config-file=path_name` on the `ndb_mgmd` command line. If the configuration file is not specified, `ndb_mgmd` by default tries to read a file named `config.ini` located in the current working directory.

The global configuration file for MySQL Cluster uses INI format, which consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard INI format is that the parameter name and value can be separated by a colon (“:”) as well as the equal sign (“=”); however, the equal sign is preferred. Another deviation is that sections are not uniquely identified by section name. Instead, unique sections (such as two different nodes of the same type) are identified by a unique ID specified as a parameter within the section.

Default values are defined for most parameters, and can also be specified in `config.ini`. To create a default value section, simply add the word `default` to the section name. For example, an `[ndbd]` section contains parameters that apply to a particular data node, whereas an `[ndbd default]` section contains parameters that apply to all data nodes. Suppose that all data nodes should use the same data memory

size. To configure them all, create an `[ndbd default]` section that contains a `DataMemory` [2043] line to specify the data memory size.



Note

In some older releases of MySQL Cluster, there was no default value for `NoOfReplicas` [2041], which always had to be specified explicitly in the `[ndbd default]` section. Although this parameter now has a default value of 2, which is the recommended setting in most common usage scenarios, it is still recommended practice to set this parameter explicitly.

The global configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two data nodes and two MySQL servers is shown here:

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com

[ndbd default]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[ndb_mgmd]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[ndbd]
HostName= ndbd_2.mysql.com

[ndbd]
HostName= ndbd_3.mysql.com

[mysqld]
[mysqld]
HostName= mysqld_5.mysql.com
```



Note

The preceding example is intended as a minimal starting configuration for purposes of familiarization with MySQL Cluster, and is almost certain not to be sufficient for production settings. See [Section 17.3.2.2, “Recommended Starting Configuration for MySQL Cluster”](#), which provides a more complete example starting configuration.

Each node has its own section in the `config.ini` file. For example, this cluster has two data nodes, so the preceding configuration file contains two `[ndbd]` sections defining these nodes.



Note

Do not place comments on the same line as a section heading in the `config.ini` file; this causes the management server not to start because it cannot parse the configuration file in such cases.

Sections of the `config.ini` File

There are six different sections that you can use in the `config.ini` configuration file, as described in the following list:

- `[computer]`: Defines cluster hosts. This is not required to configure a viable MySQL Cluster, but be may used as a convenience when setting up a large cluster. See [Section 17.3.2.4, “Defining Computers in a MySQL Cluster”](#), for more information.
- `[ndbd]`: Defines a cluster data node (`ndbd` process). See [Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”](#), for details.
- `[mysqld]`: Defines the cluster's MySQL server nodes (also called SQL or API nodes). For a discussion of SQL node configuration, see [Section 17.3.2.7, “Defining SQL and Other API Nodes in a MySQL Cluster”](#).
- `[mgm]` or `[ndb_mgmd]`: Defines a cluster management server (MGM) node. For information concerning the configuration of MGM nodes, see [Section 17.3.2.5, “Defining a MySQL Cluster Management Server”](#).
- `[tcp]`: Defines a TCP/IP connection between cluster nodes, with TCP/IP being the default connection protocol. Normally, `[tcp]` or `[tcp default]` sections are not required to set up a MySQL Cluster, as the cluster handles this automatically; however, it may be necessary in some situations to override the defaults provided by the cluster. See [Section 17.3.2.8, “MySQL Cluster TCP/IP Connections”](#), for information about available TCP/IP configuration parameters and how to use them. (You may also find [Section 17.3.2.9, “MySQL Cluster TCP/IP Connections Using Direct Connections”](#) to be of interest in some cases.)
- `[shm]`: Defines shared-memory connections between nodes. In MySQL 5.5, it is enabled by default, but should still be considered experimental. For a discussion of SHM interconnects, see [Section 17.3.2.10, “MySQL Cluster Shared-Memory Connections”](#).
- `[sci]`: Defines *Scalable Coherent Interface* connections between cluster data nodes. Such connections require software which, while freely available, is not part of the MySQL Cluster distribution, as well as specialized hardware. See [Section 17.3.2.11, “SCI Transport Connections in MySQL Cluster”](#) for detailed information about SCI interconnects.

You can define `default` values for each section. All Cluster parameter names are case-insensitive, which differs from parameters specified in `my.cnf` or `my.ini` files.

17.3.2.2. Recommended Starting Configuration for MySQL Cluster

Achieving the best performance from a MySQL Cluster depends on a number of factors including the following:

- MySQL Cluster software version
- Numbers of data nodes and SQL nodes
- Hardware
- Operating system
- Amount of data to be stored
- Size and type of load under which the cluster is to operate

Therefore, obtaining an optimum configuration is likely to be an iterative process, the outcome of which can vary widely with the specifics of each MySQL Cluster deployment. Changes in configuration are also likely to be indicated when changes are made in the platform on which the cluster is run, or in applications that use the MySQL Cluster's data. For these reasons, it is not possible to offer a single configuration that is ideal for all usage scenarios. However, in this section, we provide a recommended base configuration.

Starting `config.ini` file. The following `config.ini` file is a recommended starting point for configuring a cluster running MySQL Cluster NDB 7.2:

```
# TCP PARAMETERS

[tcp default]SendBufferMemory [2096]=2M
ReceiveBufferMemory [2097]=2M

# Increasing the sizes of these 2 buffers beyond the default values
# helps prevent bottlenecks due to slow disk I/O.

# MANAGEMENT NODE PARAMETERS

[ndb_mgmd default]
DataDir [2037]=path/to/management/server/data/directory

# It is possible to use a different data directory for each management
# server, but for ease of administration it is preferable to be
# consistent.

[ndb_mgmd]
HostName [2035]=management-server-A-hostname
# NodeId [2035]=management-server-A-nodeid

[ndb_mgmd]
HostName [2035]=management-server-B-hostname
# NodeId [2035]=management-server-B-nodeid

# Using 2 management servers helps guarantee that there is always an
# arbitrator in the event of network partitioning, and so is
# recommended for high availability. Each management server must be
# identified by a HostName. You may for the sake of convenience specify
# a NodeId for any management server, although one will be allocated
# for it automatically; if you do so, it must be in the range 1-255
# inclusive and must be unique among all IDs specified for cluster
# nodes.

# DATA NODE PARAMETERS

[ndbd default]
NoOfReplicas [2041]=2

# Using 2 replicas is recommended to guarantee availability of data;
# using only 1 replica does not provide any redundancy, which means
# that the failure of a single data node causes the entire cluster to
# shut down. We do not recommend using more than 2 replicas, since 2 is
# sufficient to provide high availability, and we do not currently test
# with greater values for this parameter.

LockPagesInMainMemory [2057]=1

# On Linux and Solaris systems, setting this parameter locks data node
# processes into memory. Doing so prevents them from swapping to disk,
# which can severely degrade cluster performance.

DataMemory [2043]=3072M
IndexMemory [2044]=384M

# The values provided for DataMemory and IndexMemory assume 4 GB RAM
# per data node. However, for best results, you should first calculate
# the memory that would be used based on the data you actually plan to
# store (you may find the ndb_size.pl utility helpful in estimating
# this), then allow an extra 20% over the calculated values. Naturally,
# you should ensure that each data node host has at least as much
# physical memory as the sum of these two values.

# ODirect [2059]=1
```

MySQL Cluster Configuration Files

```
# Enabling this parameter causes NDBCLUSTER to try using O_DIRECT
# writes for local checkpoints and redo logs; this can reduce load on
# CPUs. We recommend doing so when using MySQL Cluster on systems running
# Linux kernel 2.6 or later.

NoOfFragmentLogFiles [2051]=300
DataDir [2042]=path/to/data/node/data/directory
MaxNoOfConcurrentOperations [2047]=100000

SchedulerSpinTimer [2077]=400
SchedulerExecutionTimer [2077]=100
RealTimeScheduler [2077]=1
# Setting these parameters allows you to take advantage of real-time scheduling
# of NDBCLUSTER threads to get higher throughput.

TimeBetweenGlobalCheckpoints [2065]=1000
TimeBetweenEpochs [2065]=200
DiskCheckpointSpeed [2067]=10M
DiskCheckpointSpeedInRestart [2068]=100M
RedoBuffer [2070]=32M

# CompressedLCP [2060]=1
# CompressedBackup [2059]=1
# Enabling CompressedLCP and CompressedBackup causes, respectively, local
# checkpoint files and backup files to be compressed, which can result in a space
# savings of up to 50% over noncompressed LCPs and backups.

# MaxNoOfLocalScans [2050]=64
MaxNoOfTables [2054]=1024
MaxNoOfOrderedIndexes [2055]=256

[ndbd]
HostName [2040]=data-node-A-hostname
# NodeId [2039]=data-node-A-nodeid

LockExecuteThreadToCPU [2076]=1
LockMaintThreadsToCPU [2077]=0
# On systems with multiple CPUs, these parameters can be used to lock NDBCLUSTER
# threads to specific CPUs

[ndbd]
HostName [2040]=data-node-B-hostname
# NodeId [2039]=data-node-B-nodeid

LockExecuteThreadToCPU [2076]=1
LockMaintThreadsToCPU [2077]=0

# You must have an [ndbd] section for every data node in the cluster;
# each of these sections must include a HostName. Each section may
# optionally include a NodeId for convenience, but in most cases, it is
# sufficient to allow the cluster to allocate node IDs dynamically. If
# you do specify the node ID for a data node, it must be in the range 1
# to 48 inclusive and must be unique among all IDs specified for
# cluster nodes.

# SQL NODE / API NODE PARAMETERS

[mysqld]
# HostName [2091]=sql-node-A-hostname
# NodeId [2090]=sql-node-A-nodeid

[mysqld]

[mysqld]

# Each API or SQL node that connects to the cluster requires a [mysqld]
```

```
# or [api] section of its own. Each such section defines a connection
# "slot"; you should have at least as many of these sections in the
# config.ini file as the total number of API nodes and SQL nodes that
# you wish to have connected to the cluster at any given time. There is
# no performance or other penalty for having extra slots available in
# case you find later that you want or need more API or SQL nodes to
# connect to the cluster at the same time.
# If no HostName is specified for a given [mysqld] or [api] section,
# then any API or SQL node may use that slot to connect to the
# cluster. You may wish to use an explicit HostName for one connection slot
# to guarantee that an API or SQL node from that host can always
# connect to the cluster. If you wish to prevent API or SQL nodes from
# connecting from other than a desired host or hosts, then use a
# HostName for every [mysqld] or [api] section in the config.ini file.
# You can if you wish define a node ID (NodeId parameter) for any API or
# SQL node, but this is not necessary; if you do so, it must be in the
# range 1 to 255 inclusive and must be unique among all IDs specified
# for cluster nodes.
```

Recommended `my.cnf` options for SQL nodes. MySQL Servers acting as MySQL Cluster SQL nodes must always be started with the `--ndbcluster` [2142] and `--ndb-connectstring` options, either on the command line or in `my.cnf`. In addition, set the following options for all `mysqld` processes in the cluster, unless your setup requires otherwise:

- `--ndb-use-exact-count=0`
- `--ndb-index-stat-enable=0`
- `--ndb-force-send=1`
- `--engine-condition-pushdown=1`

17.3.2.3. The MySQL Cluster Connectstring

With the exception of the MySQL Cluster management server (`ndb_mgmd`), each node that is part of a MySQL Cluster requires a *connectstring* that points to the management server's location. This connectstring is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connectstring is as follows:

```
[nodeid=node_id, ]host-definition[, host-definition[, ...]]
host-definition:
    host_name[:port_number]
```

`node_id` is an integer larger than 1 which identifies a node in `config.ini`. `host_name` is a string representing a valid Internet host name or IP address. `port_number` is an integer referring to a TCP/IP port number.

```
example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3:1200"
example 2 (short):  "myhost1"
```

`localhost:1186` is used as the default connectstring value if none is provided. If `port_num` is omitted from the connectstring, the default port is 1186. This port should always be available on the network because it has been assigned by IANA for this purpose (see <http://www.iana.org/assignments/port-numbers> for details).

By listing multiple host definitions, it is possible to designate several redundant management servers. A MySQL Cluster data or API node attempts to contact successive management servers on each host in the order specified, until a successful connection has been established.

It is also possible to specify in a connectstring one or more bind addresses to be used by nodes having multiple network interfaces for connecting to management servers. A bind address consists of a hostname or network address and an optional port number. This enhanced syntax for connectstrings is shown here:

```
[nodeid=node_id, ]
  [bind-address=host-definition, ]
  host-definition[; bind-address=host-definition]
  host-definition[; bind-address=host-definition]
  [, ...]]
```

```
host-definition:
  host_name[:port_number]
```

If a single bind address is used in the connectstring *prior* to specifying any management hosts, then this address is used as the default for connecting to any of them (unless overridden for a given management server; see later in this section for an example). For example, the following connectstring causes the node to use `192.168.178.242` regardless of the management server to which it connects:

```
bind-address=192.168.178.242, poseidon:1186, perch:1186
```

If a bind address is specified *following* a management host definition, then it is used only for connecting to that management node. Consider the following connectstring:

```
poseidon:1186;bind-address=localhost, perch:1186;bind-address=192.168.178.242
```

In this case, the node uses `localhost` to connect to the management server running on the host named `poseidon` and `192.168.178.242` to connect to the management server running on the host named `perch`.

You can specify a default bind address and then override this default for one or more specific management hosts. In the following example, `localhost` is used for connecting to the management server running on host `poseidon`; since `192.168.178.242` is specified first (before any management server definitions), it is the default bind address and so is used for connecting to the management servers on hosts `perch` and `orca`:

```
bind-address=192.168.178.242,poseidon:1186;bind-address=localhost,perch:1186,orca:2200
```

There are a number of different ways to specify the connectstring:

- Each executable has its own command-line option which enables specifying the management server at startup. (See the documentation for the respective executable.)
- It is also possible to set the connectstring for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server's `my.cnf` file.
- For backward compatibility, two other options are available, using the same syntax:
 1. Set the `NDB_CONNECTSTRING` environment variable to contain the connectstring.
 2. Write the connectstring for each executable into a text file named `Ndb.cfg` and place this file in the executable's startup directory.

However, these are now deprecated and should not be used for new installations.

The recommended method for specifying the connectstring is to set it on the command line or in the `my.cnf` file for each executable.

17.3.2.4. Defining Computers in a MySQL Cluster

The `[computer]` section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- [Id](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	string	[none]	...
Restart Type: IN			

This is a unique identifier, used to refer to the host computer elsewhere in the configuration file.



Important

The computer ID is *not* the same as the node ID used for a management, API, or data node. Unlike the case with node IDs, you cannot use `NodeId` in place of `Id` in the `[computer]` section of the `config.ini` file.

- [HostName](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...
Restart Type: S			

This is the computer's hostname or IP address.

17.3.2.5. Defining a MySQL Cluster Management Server

The `[ndb_mgmd]` section is used to configure the behavior of the management server. `[mgm]` can be used as an alias; the two section names are equivalent. All parameters in the following list are optional and assume their default values if omitted.



Note

If neither the `ExecuteOnComputer` nor the `HostName` parameter is present, the default value `localhost` will be assumed for both.

- [Id](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.1.1	unsigned	[none]	1 - 255
Restart Type: N			

Each node in the cluster has a unique identity. For a management node, this is represented by an integer value in the range 1 to 255, inclusive. This ID is used by all internal cluster messages for addressing the node, and so must be unique for each MySQL Cluster node, regardless of the type of node.



Note

Data node IDs must be less than 49. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for management nodes (and API nodes) to values greater than 48.

The use of the `id` parameter for identifying management nodes is deprecated in favor of `NodeId` [2035]. Although `id` continues to be supported for backward compatibility, it now generates a warning and is subject to removal in a future version of MySQL Cluster.

- [NodeId](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.1.1	unsigned	[none]	1 - 255
Restart Type: N			

Each node in the cluster has a unique identity. For a management node, this is represented by an integer value in the range 1 to 255 inclusive. This ID is used by all internal cluster messages for addressing the node, and so must be unique for each MySQL Cluster node, regardless of the type of node.



Note

Data node IDs must be less than 49. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for management nodes (and API nodes) to values greater than 48.

`NodeId` is the preferred parameter name to use when identifying management nodes. Although the older `id` [2034] continues to be supported for backward compatibility, it is now deprecated and generates a warning when used; it is also subject to removal in a future MySQL Cluster release.

- [ExecuteOnComputer](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name	[none]	...
Restart Type: S			

This refers to the `id` set for one of the computers defined in a `[computer]` section of the `config.ini` file.

- [PortNumber](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	1186	0 - 64K
Restart Type: N			

This is the port number on which the management server listens for configuration requests and management commands.

- [HostName](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...

Effective Version	Type/Units	Default	Range or Permitted Values
Restart Type: S			

Specifying this parameter defines the hostname of the computer on which the management node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- [LogDestination \[2036\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	{CONSOLE SYSLOG FILE}	[see text]	...
Restart Type: N			

This parameter specifies where to send cluster logging information. There are three options in this regard—`CONSOLE`, `SYSLOG`, and `FILE`—with `FILE` being the default:

- `CONSOLE` outputs the log to `stdout`:

```
CONSOLE
```

- `SYSLOG` sends the log to a `syslog` facility, possible values being one of `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, or `local7`.



Note

Not every facility is necessarily supported by every operating system.

```
SYSLOG:facility=syslog
```

- `FILE` pipes the cluster log output to a regular file on the same machine. The following values can be specified:

- `filename`: The name of the log file.

In MySQL Cluster NDB 7.2.6 and earlier, the log file's default name, used if `FILE` was specified without also setting `filename`, was `logger.log`. Beginning with MySQL Cluster NDB 7.2.7, the default log file name used in such cases is `ndb_nodeid_cluster.log`.

- `maxsize`: The maximum size (in bytes) to which the file can grow before logging rolls over to a new file. When this occurs, the old log file is renamed by appending `.N` to the file name, where `N` is the next number not yet used with this name.
- `maxfiles`: The maximum number of log files.

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

The default value for the `FILE` parameter is

```
FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=6, where node_id is the ID of the node.
```

It is possible to specify multiple log destinations separated by semicolons as shown here:

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

- [ArbitrationRank](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	0-2	1	0 - 2
Restart Type: N			

This parameter is used to define which nodes can act as arbitrators. Only management nodes and SQL nodes can be arbitrators. [ArbitrationRank](#) can take one of the following values:

- **0**: The node will never be used as an arbitrator.
- **1**: The node has high priority; that is, it will be preferred as an arbitrator over low-priority nodes.
- **2**: Indicates a low-priority node which be used as an arbitrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its [ArbitrationRank](#) to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

You can disable arbitration completely either by setting [ArbitrationRank](#) to 0 on all management and SQL nodes, or by setting the [Arbitration \[2068\]](#) parameter in the `[ndbd default]` section of the `config.ini` global configuration file. Setting [Arbitration \[2068\]](#) causes any settings for [ArbitrationRank](#) to be disregarded.

- [ArbitrationDelay](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	0	0 - 4G
Restart Type: N			

An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

- [DataDir](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	path
Restart Type: N			

This specifies the directory where output files from the management server will be placed. These files include cluster log files, process output files, and the daemon's process ID (PID) file. (For log files, this location can be overridden by setting the `FILE` parameter for [LogDestination \[2036\]](#) as discussed previously in this section.)

The default value for this parameter is the directory in which `ndb_mgmd` is located.

- [HeartbeatThreadPriority](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.2	string	[none]	...
Restart Type: S			

Set the scheduling policy and priority of heartbeat threads for management and API nodes.

The syntax for setting this parameter is shown here:

```
HeartbeatThreadPriority = policy[, priority]

policy:
{FIFO | RR}
```

When setting this parameter, you must specify a policy. This is one of [FIFO](#) (first in, first out) or [RR](#) (round robin). The policy value is followed optionally by the priority (an integer).

- [TotalSendBufferMemory](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.0	bytes	256K	0 - 4G
Restart Type: N			

This parameter is available beginning with MySQL Cluster NDB 6.4.0. It is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

If this parameter is set, its minimum permitted value is 256KB; the maximum is 4294967039. For more detailed information about the behavior and use of [TotalSendBufferMemory](#) and configuring send buffer memory parameters, see [Section 17.3.2.12, “Configuring MySQL Cluster Send Buffer Parameters”](#).



Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect.

To add new management servers to a running MySQL Cluster, it is also necessary to perform a rolling restart of all cluster nodes after modifying any existing [config.ini](#) files. For more information about issues arising when using multiple management nodes, see [Section 17.1.6.10, “Limitations Relating to Multiple MySQL Cluster Nodes”](#).

17.3.2.6. Defining MySQL Cluster Data Nodes

The [\[ndbd\]](#) and [\[ndbd default\]](#) sections are used to configure the behavior of the cluster's data nodes.

[\[ndbd\]](#) and [\[ndbd default\]](#) are always used as the section names whether you are using [ndbd](#) or [ndbmta](#) binaries for the data node processes.

There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameter is either one of [ExecuteOnComputer](#) or [HostName](#); this must be defined in the local [\[ndbd\]](#) section.

The parameter `NoOfReplicas` [2041] should be defined in the `[ndbd default]` section, as it is common to all Cluster data nodes. It is not strictly necessary to set `NoOfReplicas` [2041], but it is good practice to set it explicitly.

Most data node parameters are set in the `[ndbd default]` section. Only those parameters explicitly stated as being able to set local values are permitted to be changed in the `[ndbd]` section. Where present, `HostName`, `NodeId` and `ExecuteOnComputer` *must* be defined in the local `[ndbd]` section, and not in any other section of `config.ini`. In other words, settings for these parameters are specific to one data node.

For those parameters affecting memory usage or buffer sizes, it is possible to use `K`, `M`, or `G` as a suffix to indicate units of 1024, 1024×1024, or 1024×1024×1024. (For example, `100K` means 100 × 1024 = 102400.) Parameter names and values are currently case-sensitive.

Information about configuration parameters specific to MySQL Cluster Disk Data tables can be found later in this section (see [Disk Data Configuration Parameters](#)).

All of these parameters also apply to `ndbmt`d (the multi-threaded version of `ndbd`). Three additional data node configuration parameters—`MaxNoOfExecutionThreads` [2079], `ThreadConfig` [2081], and `NoOfFragmentLogParts` [2080]—apply to `ndbmt`d only; these have no effect when used with `ndbd`. For more information, see [Multi-Threading Configuration Parameters \(ndbmt\)](#). See also [Section 17.4.3, “ndbmt — The MySQL Cluster Data Node Daemon \(Multi-Threaded\)”](#).

Identifying data nodes. The `NodeId` or `Id` value (that is, the data node identifier) can be allocated on the command line when the node is started or in the configuration file.

- `Id`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	[none]	1 - 48
Restart Type: N			

A unique node ID is used as the node's address for all cluster internal messages. For data nodes, this is an integer in the range 1 to 48 inclusive. Each node in the cluster must have a unique identifier.

`NodeId` is the preferred parameter name to use when identifying data nodes. Although the older `Id` [2039] is still supported for backward compatibility, it is now deprecated, and generates a warning when used. `Id` is also subject to removal in a future MySQL Cluster release.

- `NodeId`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	[none]	1 - 48
Restart Type: N			

A unique node ID is used as the node's address for all cluster internal messages. For data nodes, this is an integer in the range 1 to 48 inclusive. Each node in the cluster must have a unique identifier.

`NodeId` is the preferred parameter name to use when identifying data nodes. Although `Id` [2039] continues to be supported for backward compatibility, it is now deprecated, generates a warning when used, and is subject to removal in a future version of MySQL Cluster.

- `ExecuteOnComputer`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name	[none]	...
Restart Type: S			

This refers to the `Id` set for one of the computers defined in a `[computer]` section.

- [HostName](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	localhost	...
Restart Type: S			

Specifying this parameter defines the hostname of the computer on which the data node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- [ServerPort \[2040\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	[none]	1 - 64K
Restart Type: N			

Each node in the cluster uses a port to connect to other nodes. By default, this port is allocated dynamically in such a way as to ensure that no two nodes on the same host computer receive the same port number, so it should normally not be necessary to specify a value for this parameter.

However, if you need to be able to open specific ports in a firewall to permit communication between data nodes and API nodes (including SQL nodes), you can set this parameter to the number of the desired port in an `[ndbd]` section or (if you need to do this for multiple data nodes) the `[ndbd default]` section of the `config.ini` file, and then open the port having that number for incoming connections from SQL nodes, API nodes, or both.



Note

Connections from data nodes to management nodes is done using the `ndb_mgmd` management port (the management server's `PortNumber`; see [Section 17.3.2.5, "Defining a MySQL Cluster Management Server"](#)) so outgoing connections to that port from any data nodes should always be permitted.

- [TcpBind_INADDR_ANY](#)

Setting this parameter to `TRUE` or `1` binds `IP_ADDR_ANY` so that connections can be made from anywhere (for autogenerated connections). The default is `FALSE` (`0`).

- [NodeGroup \[2040\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.0	2040	[none]	0 - 65536

Effective Version	Type/Units	Default	Range or Permitted Values
Restart Type: IS			

This parameter can be used to assign a data node to a specific node group. It is read only when the cluster is started for the first time, and cannot be used to reassign a data node to a different node group online. It is generally not desirable to use this parameter in the `[ndbd default]` section of the `config.ini` file, and care must be taken not to assign nodes to node groups in such a way that an invalid numbers of nodes are assigned to any node groups.

The `NodeGroup [2040]` parameter is chiefly intended for use in adding a new node group to a running MySQL Cluster without having to perform a rolling restart. For this purpose, you should set it to 65536 (the maximum value). You are not required to set a `NodeGroup [2040]` value for all cluster data nodes, only for those nodes which are to be started and added to the cluster as a new node group at a later time. For more information, see [Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”](#).

- `NoOfReplicas [2041]`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.6	integer	2	1 - 4
Restart Type: IS			

This global parameter can be set only in the `[ndbd default]` section, and defines the number of replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes all storing the same information.

Node groups are formed implicitly. The first node group is formed by the set of data nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 data nodes and that `NoOfReplicas [2041]` is set to 2. The four data nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the same node groups are not placed on the same computer because a single hardware failure would cause the entire cluster to fail.

If no node IDs are provided, the order of the data nodes will be the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client's `SHOW` command.

The default value for `NoOfReplicas [2041]` is 2, which is the recommended setting in most common usage scenarios.

The maximum possible value is 4; *currently, only the values 1 and 2 are actually supported.*



Important

Setting `NoOfReplicas [2041]` to 1 means that there is only a single copy of all Cluster data; in this case, the loss of a single data node causes the cluster to fail because there are no additional copies of the data stored by that node.

The value for this parameter must divide evenly into the number of data nodes in the cluster. For example, if there are two data nodes, then `NoOfReplicas [2041]` must be equal to either 1 or 2, since

2/3 and 2/4 both yield fractional values; if there are four data nodes, then `NoOfReplicas` [2041] must be equal to 1, 2, or 4.

- [DataDir](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	path
Restart Type: IN			

This parameter specifies the directory where trace files, log files, pid files and error logs are placed.

The default is the data node process working directory.

- [FileSystemPath](#) [2042]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	path	DataDir	...
Restart Type: IN			

This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs (for Disk Data tables), and data files are placed. The default is the directory specified by `DataDir`.



Note

This directory must exist before the `ndbd` process is initiated.

The recommended directory hierarchy for MySQL Cluster includes `/var/lib/mysql-cluster`, under which a directory for the node's file system is created. The name of this subdirectory contains the node ID. For example, if the node ID is 2, this subdirectory is named `ndb_2_fs`.

- [BackupDataDir](#) [2042]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	path	[see text]	...
Restart Type: IN			

This parameter specifies the directory in which backups are placed.



Important

The string `'/BACKUP'` is always appended to this value. For example, if you set the value of `BackupDataDir` [2042] to `/var/lib/cluster-data`, then all backups are stored under `/var/lib/cluster-data/BACKUP`. This also means that the *effective* default backup location is the directory named `BACKUP` under the location specified by the `FileSystemPath` [2042] parameter.

Data Memory, Index Memory, and String Memory

`DataMemory` [2043] and `IndexMemory` [2044] are `[ndbd]` parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to

understand how [DataMemory \[2043\]](#) and [IndexMemory \[2044\]](#) are used, as they usually need to be updated to reflect actual usage by the cluster:

- [DataMemory \[2043\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	80M	1M - 1024G
Restart Type: N			

This parameter defines the amount of space (in bytes) available for storing database records. The entire amount specified by this value is allocated in memory, so it is extremely important that the machine has sufficient physical memory to accommodate it.

The memory allocated by [DataMemory \[2043\]](#) is used to store both the actual records and indexes. There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page.

For variable-size table attributes, the data is stored on separate datapages, allocated from [DataMemory \[2043\]](#). Variable-length records use a fixed-size part with an extra overhead of 4 bytes to reference the variable-size part. The variable-size part has 2 bytes overhead plus 2 bytes per attribute.

The maximum record size is 14000 bytes.

The memory space defined by [DataMemory \[2043\]](#) is also used to store ordered indexes, which use about 10 bytes per record. Each table row is represented in the ordered index. A common error among users is to assume that all indexes are stored in the memory allocated by [IndexMemory \[2044\]](#), but this is not the case: Only primary key and unique hash indexes use this memory; ordered indexes use the memory allocated by [DataMemory \[2043\]](#). However, creating a primary key or unique hash index also creates an ordered index on the same keys, unless you specify `USING HASH` in the index creation statement. This can be verified by running `ndb_desc -d db_name table_name` in the management client.

Currently, MySQL Cluster can use a maximum of 512 MB for hash indexes per partition, which means in some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free [DataMemory \[2043\]](#). This can also pose a problem with data node restarts on nodes that are heavily loaded with data. You can force NDB to create extra partitions for MySQL Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. In general, setting `MAX_ROWS` to twice the number of rows that you expect to store in the table should be sufficient. In MySQL Cluster 7.2.3 and later, you can also use the [MinFreePct \[2045\]](#) configuration parameter to help avoid problems with node restarts. (Bug #13436216)

The memory space allocated by [DataMemory \[2043\]](#) consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are data nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in [NoOfReplicas \[2041\]](#).

Once a page has been allocated, it is currently not possible to return it to the pool of free pages, except by deleting the table. (This also means that [DataMemory \[2043\]](#) pages, once allocated to a given table, cannot be used by other tables.) Performing a data node recovery also compresses the partition because all records are inserted into empty partitions from other live nodes.

The [DataMemory \[2043\]](#) memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the [DataMemory \[2043\]](#). There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure
- Large transactions use more memory

The default value for [DataMemory \[2043\]](#) is 80MB; the minimum is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached. This limit is determined by the amount of physical RAM available on the machine and by the amount of memory that the operating system may commit to any one process. 32-bit operating systems are generally limited to 2–4GB per process; 64-bit operating systems can use more. For large databases, it may be preferable to use a 64-bit operating system for this reason.

- [IndexMemory \[2044\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	18M	1M - 1T
Restart Type: N			

This parameter controls the amount of storage used for hash indexes in MySQL Cluster. Hash indexes are always used for primary key indexes, unique indexes, and unique constraints. Note that when defining a primary key and a unique index, two indexes will be created, one of which is a hash index used for all tuple accesses as well as lock handling. It is also used to enforce unique constraints.

The size of the hash index is 25 bytes per record, plus the size of the primary key. For primary keys larger than 32 bytes another 8 bytes is added.

The default value for [IndexMemory \[2044\]](#) is 18MB. The minimum is 1MB.

- [StringMemory \[2044\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	% or bytes	5	0 - 4G
NDB 7.2.1	% or bytes	25	0 - 4G
Restart Type: S			

This parameter determines how much memory is allocated for strings such as table names, and is specified in an `[ndbd]` or `[ndbd default]` section of the `config.ini` file. A value between 0 and 100 inclusive is interpreted as a percent of the maximum default value, which is calculated based on a number of factors including the number of tables, maximum table name size, maximum size of `.FRM` files, [MaxNoOfTriggers \[2056\]](#), maximum column name size, and maximum default column value.

A value greater than 100 is interpreted as a number of bytes.

The default value is 25—that is, 25 percent of the default maximum.

Under most circumstances, the default value should be sufficient, but when you have a great many Cluster tables (1000 or more), it is possible to get Error 773 *Out of string memory, please modify StringMemory config parameter: Permanent error: Schema error*, in which case you should increase this value. 25 (25 percent) is not excessive, and should prevent this error from recurring in all but the most extreme conditions.

The following example illustrates how memory is used for a table. Consider this table definition:

```
CREATE TABLE example (  
  a INT NOT NULL,  
  b INT NOT NULL,  
  c INT NOT NULL,  
  PRIMARY KEY(a),  
  UNIQUE(b)  
) ENGINE=NDBCLUSTER;
```

For each record, there are 12 bytes of data plus 12 bytes overhead. Having no nullable columns saves 4 bytes of overhead. In addition, we have two ordered indexes on columns *a* and *b* consuming roughly 10 bytes each per record. There is a primary key hash index on the base table using roughly 29 bytes per record. The unique constraint is implemented by a separate table with *b* as primary key and *a* as a column. This other table consumes an additional 29 bytes of index memory per record in the *example* table as well 8 bytes of record data plus 12 bytes of overhead.

Thus, for one million records, we need 58MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in MySQL Cluster to handle uniqueness constraints.

Currently, the only partitioning algorithm is hashing and ordered indexes are local to each node. Thus, ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both [IndexMemory \[2044\]](#) and [DataMemory \[2043\]](#) is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with two replicas, there will be two node groups. Thus, the total data memory available is $2 \times$ [DataMemory \[2043\]](#) for each data node.

It is highly recommended that [DataMemory \[2043\]](#) and [IndexMemory \[2044\]](#) be set to the same values for all nodes. Data distribution is even over all nodes in the cluster, so the maximum amount of space available for any node can be no greater than that of the smallest node in the cluster.

[DataMemory \[2043\]](#) and [IndexMemory \[2044\]](#) can be changed, but decreasing either of these can be risky; doing so can easily lead to a node or even an entire MySQL Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be acceptable, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, and then restarting the management server followed by restarting each data node in turn.

Beginning with MySQL Cluster NDB 7.2.3, a proportion (5% by default) of data node resources including [DataMemory \[2043\]](#) and [IndexMemory \[2044\]](#) is kept in reserve to insure that the data node does not exhaust its memory when performing a restart. This can be adjusted using the [MinFreePct \[2045\]](#) data node configuration parameter (default 5) introduced in the same version of MySQL Cluster.

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.18	unsigned	5	0 - 100
NDB 7.2.3	unsigned	5	0 - 100
Restart Type: N			

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.

Transaction parameters. The next few `[ndbd]` parameters that we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. `MaxNoOfConcurrentTransactions` [2046] sets the number of parallel transactions possible in a node. `MaxNoOfConcurrentOperations` [2047] sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially `MaxNoOfConcurrentOperations` [2047]) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, to ensure that these do not use excessive memory.

`MaxDMLOperationsPerTransaction` [2048] sets the maximum number of DML operations that can be performed in a given transaction.

- `MaxNoOfConcurrentTransactions` [2046]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	4096	32 - 4G
Restart Type: S			

Each cluster data node requires a transaction record for each active transaction in the cluster. The task of coordinating transactions is distributed among all of the data nodes. The total number of transaction records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

Transaction records are allocated to individual MySQL servers. Each connection to a MySQL server requires at least one transaction record, plus an additional transaction object per table accessed by that connection. This means that a reasonable minimum for the total number of transactions in the cluster can be expressed as

```
TotalNoOfConcurrentTransactions =
    (maximum number of tables accessed in any single transaction + 1)
    * number of cluster SQL nodes
```

Suppose that there are 10 SQL nodes using the cluster. A single join involving 10 tables requires 11 transaction records; if there are 10 such joins in a transaction, then $10 * 11 = 110$ transaction records are required for this transaction, per MySQL server, or $110 * 10 = 1100$ transaction records total. Each data node can be expected to handle `TotalNoOfConcurrentTransactions` / number of data nodes. For a MySQL Cluster having 4 data nodes, this would mean setting `MaxNoOfConcurrentTransactions` on each data node to $1100 / 4 = 275$. In addition, you should provide for failure recovery by insuring that a single node group can accommodate all concurrent transactions; in other words, that each data node's `MaxNoOfConcurrentTransactions` is sufficient to cover a number of transaction equal to `TotalNoOfConcurrentTransactions` / number of node groups. If this cluster has a single node group, then `MaxNoOfConcurrentTransactions` should be set to 1100 (the same as the total number of concurrent transactions for the entire cluster).

In addition, each transaction involves at least one operation; for this reason, the value set for `MaxNoOfConcurrentTransactions` should always be no more than the value of `MaxNoOfConcurrentOperations` [2047].

This parameter must be set to the same value for all cluster data nodes. This is due to the fact that, when a data node fails, the oldest surviving node re-creates the transaction state of all transactions that were ongoing in the failed node.

Changing the value of `MaxNoOfConcurrentTransactions` [2046] requires a complete shutdown and restart of the cluster.

The default value is 4096.

- `MaxNoOfConcurrentOperations` [2047]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	32K	32 - 4G
Restart Type: N			

It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions of only a few operations each and not involving a great many records, there is no need to set this parameter very high. When performing large transactions involving many records need to set this parameter higher.

Records are kept for each transaction updating cluster data, both in the transaction coordinator and in the nodes where the actual updates are performed. These records contain state information needed to find UNDO records for rollback, lock queues, and other purposes.

This parameter should be set at a minimum to the number of records to be updated simultaneously in transactions, divided by the number of cluster data nodes. For example, in a cluster which has four data nodes and which is expected to handle one million concurrent updates using transactions, you should set this value to $1000000 / 4 = 250000$. To help provide resiliency against failures, it is suggested that you set this parameter to a value that is high enough to permit an individual data node to handle the load for its node group. In other words, you should set the value equal to `total number of concurrent operations / number of node groups`. (In the case where there is a single node group, this is the same as the total number of concurrent operations for the entire cluster.)

Because each transaction always involves at least one operation, the value of `MaxNoOfConcurrentOperations` should always be greater than or equal to the value of `MaxNoOfConcurrentTransactions` [2046].

Read queries which set locks also cause operation records to be created. Some extra space is allocated within individual nodes to accommodate cases where the distribution is not perfect over the nodes.

When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

The default value is 32768.

This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction coordinator. The second part specifies how many operation records are to be local to the database.

A very large transaction performed on an eight-node cluster requires as many operation records in the transaction coordinator as there are reads, updates, and deletes involved in the transaction. However, the operation records of the are spread over all eight nodes. Thus, if it is necessary to configure the system for one very large transaction, it is a good idea to configure the two parts separately. [MaxNoOfConcurrentOperations \[2047\]](#) will always be used to calculate the number of operation records in the transaction coordinator portion of the node.

It is also important to have an idea of the memory requirements for operation records. These consume about 1KB per record.

- [MaxNoOfLocalOperations \[2048\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	UNDEFINED	32 - 4G
Restart Type: N			

By default, this parameter is calculated as $1.1 \times \text{MaxNoOfConcurrentOperations [2047]}$. This fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, it is a good idea to override the default value by explicitly specifying this parameter.

- [MaxDMLOperationsPerTransaction \[2048\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.15	operations (DML)	4294967295	32 - 4294967295
Restart Type: N			

This parameter limits the size of a transaction. The transaction is aborted if it requires more than this many DML operations. The minimum number of operations per transaction is 32; however, you can set [MaxDMLOperationsPerTransaction](#) to 0 to disable any limitation on the number of DML operations per transaction. The maximum (and default) is 4294967295.

Transaction temporary storage. The next set of [\[ndbd\]](#) parameters is used to determine temporary storage when executing a statement that is part of a Cluster transaction. All records are released when the statement is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these values to enable better parallelism in the system, whereas users whose applications require relatively small transactions can decrease the values to save memory.

- [MaxNoOfConcurrentIndexOperations \[2048\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	8K	0 - 4G
Restart Type: N			

For queries using a unique hash index, another temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus, this record is

allocated only while executing a part of a query. As soon as this part has been executed, the record is released. The state needed to handle aborts and commits is handled by the normal operation records, where the pool size is set by the parameter `MaxNoOfConcurrentOperations` [2047].

The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree of parallelism is not required for the cluster.

- `MaxNoOfFiredTriggers` [2049]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	4000	0 - 4G
Restart Type: N			

The default value of `MaxNoOfFiredTriggers` [2049] is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short-lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

- `TransactionBufferMemory` [2049]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	1M	1K - 4G
Restart Type: N			

The memory affected by this parameter is used for tracking operations fired when updating index tables and reading unique indexes. This memory is used to store the key and column information for these operations. It is only very rarely that the value for this parameter needs to be altered from the default.

The default value for `TransactionBufferMemory` [2049] is 1MB.

Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter `ZATTRBUF_FILESIZE` (found in `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) set to 4000×128 bytes (500KB). A similar buffer for key information, `ZDATABUF_FILESIZE` (also in `Dbtc.hpp`) contains $4000 \times 16 = 62.5$ KB of buffer space. `Dbtc` is the module that handles transaction coordination.

Scans and buffering. There are additional [`ndbd`] parameters in the `Dblqh` module (in `ndb/src/kernel/blocks/Dblqh/Dblqh.hpp`) that affect reads and updates. These include `ZATTRINBUF_FILESIZE`, set by default to 10000×128 bytes (1250KB) and `ZDATABUF_FILE_SIZE`, set by default to 10000×16 bytes (roughly 156KB) of buffer space. To date, there have been neither any reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

- `MaxNoOfConcurrentScans` [2049]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	256	2 - 500
Restart Type: N			

This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction coordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain [MaxNoOfConcurrentScans \[2049\]](#) scans concurrently from all nodes in the cluster.

Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. The order is kept on the local partitions only, so it is necessary to perform the index scan on all partitions.

The default value of [MaxNoOfConcurrentScans \[2049\]](#) is 256. The maximum value is 500.

- [MaxNoOfLocalScans \[2050\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	UNDEFINED	32 - 4G
Restart Type: N			

Specifies the number of local scan records if many scans are not fully parallelized. In MySQL Cluster NDB 7.2.0 and later, when the number of local scan records is not provided, it is calculated as 4 times the product of [MaxNoOfConcurrentScans \[2049\]](#) and the number of data nodes in the system. (Previously, it was calculated as the product of [MaxNoOfConcurrentScans \[2049\]](#) and the number of data nodes.) The minimum value is 32.

- [BatchSizePerLocalScan \[2050\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	64	1 - 992
NDB 7.2.1	integer	256	1 - 992
Restart Type: N			

This parameter is used to calculate the number of lock records used to handle concurrent scan operations.

[BatchSizePerLocalScan](#) has a strong connection to the [BatchSize \[2092\]](#) defined in the SQL nodes.

- [LongMessageBuffer \[2050\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	bytes	4M	512K - 4G
Restart Type: N			

This is an internal buffer used for passing messages within individual nodes and between nodes. The default is 4MB.

This parameter seldom needs to be changed from the default.

- [MaxParallelScansPerFragment \[2051\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	256	1 - 1G
Restart Type: N			

It is possible to configure the maximum number of parallel scans ([TUP](#) scans and [TUX](#) scans) allowed before they begin queuing for serial handling. You can increase this to take advantage of any unused CPU when performing large number of scans in parallel and improve their performance.

Beginning with MySQL Cluster NDB 7.2.0, the default value for this parameter was increased from 32 to 256.

Memory Allocation

- [MaxAllocate \[2051\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	32M	1M - 1G
Restart Type: N			

This is the maximum size of the memory unit to use when allocating memory for tables. In cases where [NDB](#) gives *Out of memory* errors, but it is evident by examining the cluster logs or the output of [DUMP 1000](#) (see [DUMP 1000](#)) that all available memory has not yet been used, you can increase the value of this parameter (or [MaxNoOfTables \[2054\]](#), or both) to cause [NDB](#) to make sufficient memory available.

Logging and checkpointing. The following [\[ndbd\]](#) parameters control log and checkpoint behavior.

- [NoOfFragmentLogFiles \[2051\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	16	3 - 4G
Restart Type: IN			

This parameter sets the number of REDO log files for the node, and thus the amount of space allocated to REDO logging. Because the REDO log files are organized in a ring, it is extremely important that the first and last log files in the set (sometimes referred to as the “head” and “tail” log files, respectively)

do not meet. When these approach one another too closely, the node begins aborting all transactions encompassing updates due to a lack of room for new log records.

A REDO log record is not removed until the required number of local checkpoints has been completed since that log record was inserted. (In MySQL Cluster NDB 7.2, only 2 local checkpoints are necessary). Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

The default parameter value is 16, which by default means 16 sets of 4 16MB files for a total of 1024MB. The size of the individual log files is configurable using the `FragmentLogFileSize` [2052] parameter. In scenarios requiring a great many updates, the value for `NoOfFragmentLogFiles` [2051] may need to be set as high as 300 or even higher to provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeopardizing recovery, all updating transactions are aborted with internal error code 410 (`Out of log file space temporarily`). This condition prevails until a checkpoint has completed and the log tail can be moved forward.



Important

This parameter cannot be changed “on the fly”; you must restart the node using `--initial`. If you wish to change this value for all data nodes in a running cluster, you can do so using a rolling node restart (using `--initial` when starting each data node).

- `FragmentLogFileSize` [2052]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.1.11	bytes	16M	4M - 1G
Restart Type: IN			

Setting this parameter enables you to control directly the size of redo log files. This can be useful in situations when MySQL Cluster is operating under a high load and it is unable to close fragment log files quickly enough before attempting to open new ones (only 2 fragment log files can be open at one time); increasing the size of the fragment log files gives the cluster more time before having to open each new fragment log file. The default value for this parameter is 16M.

For more information about fragment log files, see the description for `NoOfFragmentLogFiles` [2051].

- `InitFragmentLogFiles` [2052]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.19	[see values]	SPARSE	SPARSE, FULL
Restart Type: IN			

By default, fragment log files are created sparsely when performing an initial start of a data node—that is, depending on the operating system and file system in use, not all bytes are necessarily written to disk. However, it is possible to override this behavior and force all bytes to be written, regardless of the platform and file system type being used, by means of this parameter. `InitFragmentLogFiles` [2052] takes either of two values:

- `SPARSE`. Fragment log files are created sparsely. This is the default value.

- [FULL](#). Force all bytes of the fragment log file to be written to disk.

Depending on your operating system and file system, setting `InitFragmentLogFiles=FULL` may help eliminate I/O errors on writes to the REDO log.

- [MaxNoOfOpenFiles \[2053\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	0	20 - 4G
Restart Type: N			

This parameter sets a ceiling on how many internal threads to allocate for open files. *Any situation requiring a change in this parameter should be reported as a bug.*

The default value is 0. However, the minimum value to which this parameter can be set is 20.

- [InitialNoOfOpenFiles \[2053\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	files	27	20 - 4G
Restart Type: N			

This parameter sets the initial number of internal threads to allocate for open files.

The default value is 27.

- [MaxNoOfSavedMessages \[2053\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	25	0 - 4G
Restart Type: N			

This parameter sets the maximum number of trace files that are kept before overwriting old ones. Trace files are generated when, for whatever reason, the node crashes.

The default is 25 trace files.

- [MaxLCPStartDelay \[2053\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.3	seconds	0	0 - 600
Restart Type: N			

In parallel data node recovery, only table data is actually copied and synchronized in parallel; synchronization of metadata such as dictionary and checkpoint information is done in a serial fashion. In addition, recovery of dictionary and checkpoint information cannot be executed in parallel with performing of local checkpoints. This means that, when starting or restarting many data nodes

concurrently, data nodes may be forced to wait while a local checkpoint is performed, which can result in longer node recovery times.

It is possible to force a delay in the local checkpoint to permit more (and possibly all) data nodes to complete metadata synchronization; once each data node's metadata synchronization is complete, all of the data nodes can recover table data in parallel, even while the local checkpoint is being executed. To force such a delay, set `MaxLCPStartDelay` [2053], which determines the number of seconds the cluster can wait to begin a local checkpoint while data nodes continue to synchronize metadata. This parameter should be set in the `[ndbd default]` section of the `config.ini` file, so that it is the same for all data nodes. The maximum value is 600; the default is 0.

Metadata objects. The next set of `[ndbd]` parameters defines pool sizes for metadata objects, used to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters. Note that these act merely as “suggestions” to the cluster, and any that are not specified revert to the default values shown.

- `MaxNoOfAttributes` [2054]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	1000	32 - 4G
Restart Type: N			

This parameter sets a suggested maximum number of attributes that can be defined in the cluster; like `MaxNoOfTables` [2054], it is not intended to function as a hard upper limit.

(In older MySQL Cluster releases, this parameter was sometimes treated as a hard limit for certain operations. This caused problems with MySQL Cluster Replication, when it was possible to create more tables than could be replicated, and sometimes led to confusion when it was possible [or not possible, depending on the circumstances] to create more than `MaxNoOfAttributes` attributes.)

The default value is 1000, with the minimum possible value being 32. The maximum is 4294967039. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

When setting `MaxNoOfAttributes` [2054], it is important to prepare in advance for any `ALTER TABLE` statements that you might want to perform in the future. This is due to the fact, during the execution of `ALTER TABLE` on a Cluster table, 3 times the number of attributes as in the original table are used, and a good practice is to permit double this amount. For example, if the MySQL Cluster table having the greatest number of attributes (`greatest_number_of_attributes`) has 100 attributes, a good starting point for the value of `MaxNoOfAttributes` [2054] would be $6 * \text{greatest_number_of_attributes} = 600$.

You should also estimate the average number of attributes per table and multiply this by `MaxNoOfTables` [2054]. If this value is larger than the value obtained in the previous paragraph, you should use the larger value instead.

Assuming that you can create all desired tables without any problems, you should also verify that this number is sufficient by trying an actual `ALTER TABLE` after configuring the parameter. If this is not successful, increase `MaxNoOfAttributes` [2054] by another multiple of `MaxNoOfTables` [2054] and test it again.

- `MaxNoOfTables` [2054]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	128	8 - 20320
Restart Type: N			

A table object is allocated for each table and for each unique hash index in the cluster. This parameter sets a suggested maximum number of table objects for the cluster as a whole; like [MaxNoOfAttributes \[2054\]](#), it is not intended to function as a hard upper limit.

(In older MySQL Cluster releases, this parameter was sometimes treated as a hard limit for certain operations. This caused problems with MySQL Cluster Replication, when it was possible to create more tables than could be replicated, and sometimes led to confusion when it was possible [or not possible, depending on the circumstances] to create more than [MaxNoOfTables](#) tables.)

For each attribute that has a [BLOB](#) data type an extra table is used to store most of the [BLOB](#) data. These tables also must be taken into account when defining the total number of tables.

The default value of this parameter is 128. The minimum is 8 and the maximum is 20320. Each table object consumes approximately 20KB per node.



Note

The sum of [MaxNoOfTables \[2054\]](#), [MaxNoOfOrderedIndexes \[2055\]](#), and [MaxNoOfUniqueHashIndexes \[2055\]](#) must not exceed $2^{32} - 2$ (4294967294).

- [MaxNoOfOrderedIndexes \[2055\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	128	0 - 4G
Restart Type: N			

For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default, each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index. [MaxNoOfOrderedIndexes \[2055\]](#) sets the total number of ordered indexes that can be in use in the system at any one time.

The default value of this parameter is 128. Each index object consumes approximately 10KB of data per node.



Note

The sum of [MaxNoOfTables \[2054\]](#), [MaxNoOfOrderedIndexes \[2055\]](#), and [MaxNoOfUniqueHashIndexes \[2055\]](#) must not exceed $2^{32} - 2$ (4294967294).

- [MaxNoOfUniqueHashIndexes \[2055\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	64	0 - 4G

Effective Version	Type/Units	Default	Range or Permitted Values
Restart Type: N			

For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default, an ordered index is also defined for each unique index. To prevent this, you must specify the `USING HASH` option when defining the unique index.

The default value is 64. Each index consumes approximately 15KB per node.



Note

The sum of `MaxNoOfTables` [2054], `MaxNoOfOrderedIndexes` [2055], and `MaxNoOfUniqueHashIndexes` [2055] must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfTriggers` [2056]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	768	0 - 4G
Restart Type: N			

Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that three triggers are created for each unique hash index.) However, an *ordered* index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

Replication between clusters also makes use of internal triggers.

This parameter sets the maximum number of trigger objects in the cluster.

The default value is 768.

- `MaxNoOfIndexes` [2056]

This parameter is deprecated and subject to removal in a future version of MySQL Cluster. You should use `MaxNoOfOrderedIndexes` [2055] and `MaxNoOfUniqueHashIndexes` [2055] instead.

This parameter is used only by unique hash indexes. There needs to be one record in this pool for each unique hash index defined in the cluster.

The default value of this parameter is 128.

- `MaxNoOfSubscriptions` [2056]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.7	unsigned	0	0 - 4G
Restart Type: N			

Each NDB table in a MySQL Cluster requires a subscription in the NDB kernel. For some NDB API applications, it may be necessary or desirable to change this parameter. However, for normal usage with MySQL servers acting as SQL nodes, there is not any need to do so.

The default value for `MaxNoOfSubscriptions` [2056] is 0, which is treated as equal to `MaxNoOfTables` [2054]. Each subscription consumes 108 bytes.

- `MaxNoOfSubscribers` [2057]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.7	unsigned	0	0 - 4G
Restart Type: N			

This parameter is of interest only when using MySQL Cluster Replication. The default value is 0, which is treated as $2 * \text{MaxNoOfTables}$; that is, there is one subscription per NDB table for each of two MySQL servers (one acting as the replication master and the other as the slave). Each subscriber uses 16 bytes of memory.

When using circular replication, multi-master replication, and other replication setups involving more than 2 MySQL servers, you should increase this parameter to the number of `mysqld` processes included in replication (this is often, but not always, the same as the number of clusters). For example, if you have a circular replication setup using three MySQL Clusters, with one `mysqld` attached to each cluster, and each of these `mysqld` processes acts as a master and as a slave, you should set `MaxNoOfSubscribers` [2057] equal to $3 * \text{MaxNoOfTables}$.

For more information, see [Section 17.6, “MySQL Cluster Replication”](#).

- `MaxNoOfConcurrentSubOperations` [2057]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.7	unsigned	256	0 - 4G
Restart Type: N			

This parameter sets a ceiling on the number of operations that can be performed by all API nodes in the cluster at one time. The default value (256) is sufficient for normal operations, and might need to be adjusted only in scenarios where there are a great many API nodes each performing a high volume of operations concurrently.

Boolean parameters. The behavior of data nodes is also affected by a set of `[ndbd]` parameters taking on boolean values. These parameters can each be specified as `TRUE` by setting them equal to `1` or `Y`, and as `FALSE` by setting them equal to `0` or `N`.

- `LockPagesInMainMemory` [2057]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	numeric	0	0 - 2
Restart Type: N			

For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and so avoid any swapping to disk. This can be used to help guarantee the cluster's real-time characteristics.

This parameter takes one of the integer values `0`, `1`, or `2`, which act as shown in the following list:

- `0`: Disables locking. This is the default value.
- `1`: Performs the lock after allocating memory for the process.
- `2`: Performs the lock before memory for the process is allocated.

If the operating system is not configured to permit unprivileged users to lock pages, then the data node process making use of this parameter may have to be run as system root. ([LockPagesInMainMemory \[2057\]](#) uses the `mlockall` function. From Linux kernel 2.6.9, unprivileged users can lock memory as limited by `max locked memory`. For more information, see `ulimit -l` and <http://linux.die.net/man/2/mlock>).



Note

In older MySQL Cluster releases, this parameter was a Boolean. `0` or `false` was the default setting, and disabled locking. `1` or `true` enabled locking of the process after its memory was allocated. In MySQL Cluster NDB 7.2, using `true` or `false` as the value of this parameter causes an error.

- [StopOnError \[2058\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	boolean	true	true, false
Restart Type: N			

This parameter specifies whether an `ndbd` process should exit or perform an automatic restart when an error condition is encountered.

This feature is enabled by default.

- [CrashOnCorruptedTuple \[2058\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.28, 5.1.56-ndb-7.1.17	boolean	false	true, false
NDB 7.2.1	boolean	true	true, false
Restart Type: S			

When this parameter is enabled, it forces a data node to shut down whenever it encounters a corrupted tuple. In MySQL Cluster NDB 7.2.1 and later, it is enabled by default. This is a change from MySQL Cluster NDB 7.0 and MySQL Cluster NDB 7.1, where it was disabled by default.

- [Diskless \[2058\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	true false (1 0)	false	true, false
Restart Type: IS			

It is possible to specify MySQL Cluster tables as *diskless*, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that neither the tables nor the records in those tables survive a crash. However, when operating in diskless mode, it is possible to run `ndbd` on a diskless computer.



Important

This feature causes the *entire* cluster to operate in diskless mode.

When this feature is enabled, Cluster online backup is disabled. In addition, a partial start of the cluster is not possible.

`Diskless` [2058] is disabled by default.

- `ODirect` [2059]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.0	boolean	false	true, false
Restart Type: N			

Enabling this parameter causes `NDB` to attempt using `O_DIRECT` writes for LCP, backups, and redo logs, often lowering `kswapd` and CPU usage. When using MySQL Cluster on Linux, enable `ODirect` [2059] if you are using a 2.6 or later kernel.

`ODirect` [2059] is disabled by default.

- `RestartOnErrorInsert` [2059]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	error code	2	0 - 4
Restart Type: N			

This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

This feature is disabled by default.

- `CompressedBackup` [2059]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.7	boolean	false	true, false
Restart Type: N			

Setting this parameter to `1` causes backup files to be compressed. The compression used is equivalent to `gzip --fast`, and can save 50% or more of the space required on the data node to store uncompressed backup files. Compressed backups can be enabled for individual data nodes, or for all data nodes (by setting this parameter in the `[ndbd default]` section of the `config.ini` file).

**Important**

You cannot restore a compressed backup to a cluster running a MySQL version that does not support this feature.

The default value is `0` (disabled).

- [CompressedLCP \[2060\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.7	boolean	false	true, false
Restart Type: N			

Setting this parameter to `1` causes local checkpoint files to be compressed. The compression used is equivalent to `gzip --fast`, and can save 50% or more of the space required on the data node to store uncompressed checkpoint files. Compressed LCPs can be enabled for individual data nodes, or for all data nodes (by setting this parameter in the `[ndbd default]` section of the `config.ini` file).

**Important**

You cannot restore a compressed local checkpoint to a cluster running a MySQL version that does not support this feature.

The default value is `0` (disabled).

Controlling Timeouts, Intervals, and Disk Paging

There are a number of `[ndbd]` parameters specifying timeouts and intervals between various actions in Cluster data nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this are mentioned where applicable.

- [TimeBetweenWatchDogCheck \[2060\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	6000	70 - 4G
Restart Type: N			

To prevent the main thread from getting stuck in an endless loop at some point, a “watchdog” thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the process remains in the same state after three checks, the watchdog thread terminates it.

This parameter can easily be changed for purposes of experimentation or to adapt to local conditions. It can be specified on a per-node basis although there seems to be little reason for doing so.

The default timeout is 6000 milliseconds (6 seconds).

- [TimeBetweenWatchDogCheckInitial \[2060\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	6000	70 - 4G

Effective Version	Type/Units	Default	Range or Permitted Values
Restart Type: N			

This is similar to the [TimeBetweenWatchDogCheck \[2060\]](#) parameter, except that [TimeBetweenWatchDogCheckInitial \[2060\]](#) controls the amount of time that passes between execution checks inside a database node in the early start phases during which memory is allocated.

The default timeout is 6000 milliseconds (6 seconds).

- [StartPartialTimeout \[2061\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	30000	0 - 4G
Restart Type: N			

This parameter specifies how long the Cluster waits for all data nodes to come up before the cluster initialization routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default value is 30000 milliseconds (30 seconds). 0 disables the timeout, in which case the cluster may start only if all nodes are available.

- [StartPartitionedTimeout \[2061\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	60000	0 - 4G
Restart Type: N			

If the cluster is ready to start after waiting for [StartPartialTimeout \[2061\]](#) milliseconds but is still possibly in a partitioned state, the cluster waits until this timeout has also passed. If [StartPartitionedTimeout \[2061\]](#) is set to 0, the cluster waits indefinitely.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default timeout is 60000 milliseconds (60 seconds).

- [StartFailureTimeout \[2061\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	0	0 - 4G
Restart Type: N			

If a data node has not completed its startup sequence within the time specified by this parameter, the node startup fails. Setting this parameter to 0 (the default value) means that no data node timeout is applied.

For nonzero values, this parameter is measured in milliseconds. For data nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a data node

containing several gigabytes of data, a period as long as 10–15 minutes (that is, 600000 to 1000000 milliseconds) might be required to perform a node restart.

- [StartNoNodeGroupTimeout \[2062\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.24	milliseconds	15000	0 - 4294967039
Restart Type: N			

When a data node is configured with `Nodegroup = 65536 [2040]`, is regarded as not being assigned to any node group. When that is done, the cluster waits `StartNoNodegroupTimeout` milliseconds, then treats such nodes as though they had been added to the list passed to the `--nowait-nodes [2180]` option, and starts. The default value is `15000` (that is, the management server waits 15 seconds). Setting this parameter equal to `0` means that the cluster waits indefinitely.

`StartNoNodegroupTimeout` must be the same for all data nodes in the cluster; for this reason, you should always set it in the `[ndbd default]` section of the `config.ini` file, rather than for individual data nodes.

See [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#), for more information.

- [HeartbeatIntervalDbDb \[2062\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	5000	10 - 4G
Restart Type: N			

One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. After missing three heartbeat intervals in a row, the node is declared dead. Thus, the maximum time for discovering a failure through the heartbeat mechanism is four times the heartbeat interval.

In MySQL Cluster NDB 7.2.0 and later, the default heartbeat interval is 5000 milliseconds (5 seconds). (Previously, the default was 1500 milliseconds [1.5 seconds]). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds, obviously the node will be declared dead very quickly. This parameter can be changed during an online software upgrade, but only in small increments.

See also [Network communication and latency](#).

- [HeartbeatIntervalDbApi \[2062\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	1500	100 - 4G
Restart Type: N			

Each data node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared “dead,” in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until all

activities initiated by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for [HeartbeatIntervalDbDb \[2062\]](#).

The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between individual data nodes because each data node watches the MySQL servers connected to it, independently of all other data nodes.

For more information, see [Network communication and latency](#).

- [HeartbeatOrder \[2063\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.5	numeric	0	0 - 65535
Restart Type: S			

Data nodes send heartbeats to one another in a circular fashion whereby each data node monitors the previous one. If a heartbeat is not detected by a given data node, this node declares the previous data node in the circle “dead” (that is, no longer accessible by the cluster). The determination that a data node is dead is done globally; in other words; once a data node is declared dead, it is regarded as such by all nodes in the cluster.

It is possible for heartbeats between data nodes residing on different hosts to be too slow compared to heartbeats between other pairs of nodes (for example, due to a very low heartbeat interval or temporary connection problem), such that a data node is declared dead, even though the node can still function as part of the cluster. .

In this type of situation, it may be that the order in which heartbeats are transmitted between data nodes makes a difference as to whether or not a particular data node is declared dead. If this declaration occurs unnecessarily, this can in turn lead to the unnecessary loss of a node group and as thus to a failure of the cluster.

Consider a setup where there are 4 data nodes A, B, C, and D running on 2 host computers [host1](#) and [host2](#), and that these data nodes make up 2 node groups, as shown in the following table:

	host1	host2
<i>Node Group 0:</i>	Node A	Node B
<i>Node Group 1:</i>	Node C	Node D

Suppose the heartbeats are transmitted in the order A->B->C->D->A. In this case, the loss of the heartbeat between the hosts causes node B to declare node A dead and node C to declare node B dead. This results in loss of Node Group 0, and so the cluster fails. On the other hand, if the order of transmission is A->B->D->C->A (and all other conditions remain as previously stated), the loss of the heartbeat causes nodes A and D to be declared dead; in this case, each node group has one surviving node, and the cluster survives.

The [HeartbeatOrder \[2063\]](#) configuration parameter makes the order of heartbeat transmission user-configurable. The default value for [HeartbeatOrder \[2063\]](#) is zero; allowing the default value to be used on all data nodes causes the order of heartbeat transmission to be determined by [NDB](#). If this parameter is used, it must be set to a nonzero value (maximum 65535) for every data node in the cluster, and this value must be unique for each data node; this causes the heartbeat transmission to proceed from data node to data node in the order of their [HeartbeatOrder \[2063\]](#) values from lowest to highest (and then directly from the data node having the highest [HeartbeatOrder \[2063\]](#) to the data

node having the lowest value, to complete the circle). The values need not be consecutive; for example, to force the heartbeat transmission order A->B->D->C->A in the scenario outlined previously, you could set the `HeartbeatOrder` [2063] values as shown here:

Node	HeartbeatOrder
A	10
B	20
C	30
D	25

To use this parameter to change the heartbeat transmission order in a running MySQL Cluster, you must first set `HeartbeatOrder` [2063] for each data node in the cluster in the global configuration (`config.ini`) file (or files). To cause the change to take effect, you must perform either of the following:

- A complete shutdown and restart of the entire cluster.
- 2 rolling restarts of the cluster in succession. *All nodes must be restarted in the same order in both rolling restarts.*

You can use `DUMP 908` to observe the effect of this parameter in the data node logs.

- `ConnectCheckIntervalDelay` [2064]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.13	string	1500	0 - 4G
NDB 7.2.1	string	0	0 - 4G
Restart Type: N			

This parameter enables connection checking between data nodes. A data node that fails to respond within an interval of `ConnectCheckIntervalDelay` seconds is considered suspect, and is considered dead after two such intervals.

The default value for this parameter is 0; this is a change from MySQL Cluster NDB 7.1.

- `TimeBetweenLocalCheckpoints` [2064]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	number of 4-byte words, as a base-2 logarithm	20	0 - 31
Restart Type: N			

This parameter is an exception in that it does not specify a time to wait before starting a new local checkpoint; rather, it is used to ensure that local checkpoints are not performed in a cluster where relatively few updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the base-2 logarithm of the number of 4-byte words, so that the default value 20 means 4MB (4×2^{20}) of write operations, 21 would mean 8MB, and so on up to a maximum value of 31, which equates to 8GB of write operations.

All the write operations in the cluster are added together. Setting [TimeBetweenLocalCheckpoints \[2064\]](#) to 6 or less means that local checkpoints will be executed continuously without pause, independent of the cluster's workload.

- [TimeBetweenGlobalCheckpoints \[2065\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.2.0	milliseconds	2000	20 - 32000
Restart Type: N			

When a transaction is committed, it is committed in main memory in all nodes on which the data is mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behind this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases—a complete crash of the cluster—is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group. Later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- [TimeBetweenEpochs \[2065\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.2.5	milliseconds	100	0 - 32000
Restart Type: N			

This parameter defines the interval between synchronization epochs for MySQL Cluster Replication. The default value is 100 milliseconds.

[TimeBetweenEpochs \[2065\]](#) is part of the implementation of “micro-GCPs”, which can be used to improve the performance of MySQL Cluster Replication.

- [TimeBetweenEpochsTimeout \[2065\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	4000	0 - 256000
Restart Type: N			

This parameter defines a timeout for synchronization epochs for MySQL Cluster Replication. If a node fails to participate in a global checkpoint within the time determined by this parameter, the node is shut down. In MySQL Cluster NDB 7.2.0 and later, the default value is 0; in other words, the timeout is disabled. This represents a change from previous versions of MySQL Cluster, in which the default value was 4000 milliseconds (4 seconds).

[TimeBetweenEpochsTimeout \[2065\]](#) is part of the implementation of “micro-GCPs”, which can be used to improve the performance of MySQL Cluster Replication.

The current value of this parameter and a warning are written to the cluster log whenever a GCP save takes longer than 1 minute or a GCP save takes longer than 10 seconds.

Setting this parameter to zero has the effect of disabling GCP stops caused by save timeouts, commit timeouts, or both. The maximum possible value for this parameter is 256000 milliseconds.

- [MaxBufferedEpochs \[2066\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.2.14	epochs	100	0 - 100000
Restart Type: N			

The number of unprocessed epochs by which a subscribing node can lag behind. Exceeding this number causes a lagging subscriber to be disconnected.

The default value of 100 is sufficient for most normal operations. If a subscribing node does lag enough to cause disconnections, it is usually due to network or scheduling issues with regard to processes or threads. (In rare circumstances, the problem may be due to a bug in the [NDB](#) client.) It may be desirable to set the value lower than the default when epochs are longer.

Disconnection prevents client issues from affecting the data node service, running out of memory to buffer data, and eventually shutting down. Instead, only the client is affected as a result of the disconnect (by, for example gap events in the binary log), forcing the client to reconnect or restart the process.

- [TimeBetweenInactiveTransactionAbortCheck \[2066\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	1000	1000 - 4G
Restart Type: N			

Timeout handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, every transaction will be checked for timing out once per second.

The default value is 1000 milliseconds (1 second).

- [TransactionInactiveTimeout \[2066\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	4G	0 - 4G
Restart Type: N			

This parameter states the maximum time that is permitted to lapse between operations in the same transaction before the transaction is aborted.

The default for this parameter is 4G (also the maximum). For a real-time database that needs to ensure that no transaction keeps locks for too long, this parameter should be set to a relatively small value. The unit is milliseconds.

- [TransactionDeadlockDetectionTimeout \[2067\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	1200	50 - 4G
Restart Type: N			

When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. A failure to respond can occur for any of the following reasons:

- The node is “dead”
- The operation has entered a lock queue
- The node requested to perform the action could be heavily overloaded.

This timeout parameter states how long the transaction coordinator waits for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection.

The default timeout value is 1200 milliseconds (1.2 seconds).

The minimum for this parameter is 50 milliseconds.

- [DiskSyncSize \[2067\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	4M	32K - 4G
Restart Type: N			

This is the maximum number of bytes to store before flushing data to a local checkpoint file. This is done to prevent write buffering, which can impede performance significantly. This parameter is *not* intended to take the place of [TimeBetweenLocalCheckpoints \[2064\]](#).



Note

When [ODirect \[2059\]](#) is enabled, it is not necessary to set [DiskSyncSize \[2067\]](#); in fact, in such cases its value is simply ignored.

The default value is 4M (4 megabytes).

- [DiskCheckpointSpeed \[2067\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	10M	1M - 4G
Restart Type: N			

The amount of data, in bytes per second, that is sent to disk during a local checkpoint. This allocation is shared by DML operations and backups (but not backup logging), which means that backups started during times of intensive DML may be impaired by flooding of the redo log buffer and may fail altogether if the contention is sufficiently severe.

The default value is 10M (10 megabytes per second).

- [DiskCheckpointSpeedInRestart \[2068\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	100M	1M - 4G
Restart Type: N			

The amount of data, in bytes per second, that is sent to disk during a local checkpoint as part of a restart operation.

The default value is 100M (100 megabytes per second).

- [NoOfDiskPagesToDiskAfterRestartTUP \[2068\]](#)

This parameter is deprecated and subject to removal in a future version of MySQL Cluster. Use [DiskCheckpointSpeedInRestart \[2068\]](#) and [DiskSyncSize \[2067\]](#) instead.

- [NoOfDiskPagesToDiskAfterRestartACC \[2068\]](#)

This parameter is deprecated and subject to removal in a future version of MySQL Cluster. Use [DiskCheckpointSpeedInRestart \[2068\]](#) and [DiskSyncSize \[2067\]](#) instead.

- [NoOfDiskPagesToDiskDuringRestartTUP \[2068\]](#) (DEPRECATED)

This parameter is deprecated and subject to removal in a future version of MySQL Cluster. Use [DiskCheckpointSpeedInRestart \[2068\]](#) and [DiskSyncSize \[2067\]](#) instead.

- [NoOfDiskPagesToDiskDuringRestartACC \[2068\]](#) (DEPRECATED)

This parameter is deprecated and subject to removal in a future version of MySQL Cluster. Use [DiskCheckpointSpeedInRestart \[2068\]](#) and [DiskSyncSize \[2067\]](#) instead.

- [ArbitrationTimeout \[2068\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	7500	10 - 4G
Restart Type: N			

This parameter specifies how long data nodes wait for a response from the arbitrator to an arbitration message. If this is exceeded, the network is assumed to have split.

In MySQL Cluster NDB 7.2.0 and later, the default value is 7500 milliseconds (7.5 seconds). Previously, this was 3000 milliseconds (3 seconds).

- [Arbitration \[2068\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.7	enumeration	Default	Default, Disabled, WaitExternal
Restart Type: N			

The [Arbitration \[2068\]](#) parameter enables a choice of arbitration schemes, corresponding to one of 3 possible values for this parameter:

- **Default.** This enables arbitration to proceed normally, as determined by the [ArbitrationRank](#) settings for the management and API nodes. This is the default value.
- **Disabled.** Setting `Arbitration = Disabled` in the `[ndbd default]` section of the `config.ini` file to accomplish the same task as setting `ArbitrationRank` to 0 on all management and API nodes. When `Arbitration` is set in this way, any `ArbitrationRank` settings are ignored.
- **WaitExternal.** The [Arbitration \[2068\]](#) parameter also makes it possible to configure arbitration in such a way that the cluster waits until after the time determined by [ArbitrationTimeout \[2068\]](#) has passed for an external cluster manager application to perform arbitration instead of handling arbitration internally. This can be done by setting `Arbitration = WaitExternal` in the `[ndbd default]` section of the `config.ini` file. For best results with the `WaitExternal` setting, it is recommended that [ArbitrationTimeout \[2068\]](#) be 2 times as long as the interval required by the external cluster manager to perform arbitration.



Important

This parameter should be used only in the `[ndbd default]` section of the cluster configuration file. The behavior of the cluster is unspecified when [Arbitration \[2068\]](#) is set to different values for individual data nodes.

Buffering and logging. Several `[ndbd]` configuration parameters enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front ends to the file system when writing log records to disk. If the node is running in diskless mode, these parameters can be set to their minimum values without penalty due to the fact that disk writes are “faked” by the NDB storage engine's file system abstraction layer.

- [UndoIndexBuffer \[2069\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	2M	1M - 4G
Restart Type: N			

The UNDO index buffer, whose size is set by this parameter, is used during local checkpoints. The NDB storage engine uses a recovery scheme based on checkpoint consistency in conjunction with an operational REDO log. To produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

The UNDO index buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the NDB storage engine writes UNDO log records that map all physical changes to an index page so that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to ensure that these operations need no UNDO logging.

This buffer is 2MB by default. The minimum value is 1MB, which is sufficient for most applications. For applications doing extremely large or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the NDB storage engine issues internal error code 677 ([Index UNDO buffers overloaded](#)).



Important

It is not safe to decrease the value of this parameter during a rolling restart.

- [UndoDataBuffer \[2070\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	16M	1M - 4G
Restart Type: N			

This parameter sets the size of the UNDO data buffer, which performs a function similar to that of the UNDO index buffer, except the UNDO data buffer is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.

Because UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

This amount of memory may be unnecessarily large for some applications. In such cases, it is possible to decrease this size to a minimum of 1MB.

It is rarely necessary to increase the size of this buffer. If there is such a need, it is a good idea to check whether the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

If this buffer is too small and gets congested, the NDB storage engine issues internal error code 891 ([Data UNDO buffers overloaded](#)).



Important

It is not safe to decrease the value of this parameter during a rolling restart.

- [RedoBuffer \[2070\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	bytes	32M	1M - 4G
Restart Type: N			

All update activities also need to be logged. The REDO log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a “fuzzy” checkpoint of the data together with the UNDO log, and then applies the REDO log to play back all changes up to the restoration point.

[RedoBuffer \[2070\]](#) sets the size of the buffer in which the REDO log is written. The default value is 32MB; the minimum value is 1MB.

If this buffer is too small, the NDB storage engine issues error code 1221 (`REDO log buffers overloaded`). For this reason, you should exercise care if you attempt to decrease the value of [RedoBuffer \[2070\]](#) as part of an online change in the cluster's configuration.

Controlling log messages. In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to `stdout`. For each event category, there are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to `stdout`; setting it to 0 means that there will be no event reports made in that category.

By default, only the startup message is sent to `stdout`, with the remaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server's cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

- [LogLevelStartup \[2071\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	1	0 - 15
Restart Type: N			

The reporting level for events generated during startup of the process.

The default level is 1.

- [LogLevelShutdown \[2071\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	0	0 - 15
Restart Type: N			

The reporting level for events generated as part of graceful shutdown of a node.

The default level is 0.

- [LogLevelStatistic \[2071\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	0	0 - 15
Restart Type: N			

The reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.

The default level is 0.

- [LogLevelCheckpoint \[2072\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	log level	0	0 - 15
Restart Type: N			

The reporting level for events generated by local and global checkpoints.

The default level is 0.

- [LogLevelNodeRestart \[2072\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	0	0 - 15
Restart Type: N			

The reporting level for events generated during node restart.

The default level is 0.

- [LogLevelConnection \[2072\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	0	0 - 15
Restart Type: N			

The reporting level for events generated by connections between cluster nodes.

The default level is 0.

- [LogLevelError \[2072\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	0	0 - 15
Restart Type: N			

The reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

The default level is 0.

- [LogLevelCongestion \[2072\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	levelr	0	0 - 15
Restart Type: N			

The reporting level for events generated by congestion. These errors do not cause node failure but are still considered worth reporting.

The default level is 0.

- [LogLevelInfo \[2073\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	integer	0	0 - 15
Restart Type: N			

The reporting level for events generated for information about the general state of the cluster.

The default level is 0.

- [MemReportFrequency \[2073\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	0	0 - 4G
Restart Type: N			

This parameter controls how often data node memory usage reports are recorded in the cluster log; it is an integer value representing the number of seconds between reports.

Each data node's data memory and index memory usage is logged as both a percentage and a number of 32 KB pages of the [DataMemory \[2043\]](#) and [IndexMemory \[2044\]](#), respectively, set in the `config.ini` file. For example, if [DataMemory \[2043\]](#) is equal to 100 MB, and a given data node is using 50 MB for data memory storage, the corresponding line in the cluster log might look like this:

```
2006-12-24 01:18:16 [MgmSrvr] INFO -- Node 2: Data usage is 50%(1280 32K pages of total 2560)
```

[MemReportFrequency \[2073\]](#) is not a required parameter. If used, it can be set for all cluster data nodes in the `[ndbd default]` section of `config.ini`, and can also be set or overridden for individual data nodes in the corresponding `[ndbd]` sections of the configuration file. The minimum value—which is also the default value—is 0, in which case memory reports are logged only when memory usage reaches certain percentages (80%, 90%, and 100%), as mentioned in the discussion of statistics events in [Section 17.5.6.2, “MySQL Cluster Log Events”](#).

- [StartupStatusReportFrequency \[2073\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.0	seconds	[none]	...
Restart Type: N			

When a data node is started with the `--initial` [2178], it initializes the redo log file during Start Phase 4 (see Section 17.5.1, “Summary of MySQL Cluster Start Phases”). When very large values are set for `NoOfFragmentLogFiles` [2051], `FragmentLogFileSize` [2052], or both, this initialization can take a long time. You can force reports on the progress of this process to be logged periodically, by means of the `StartupStatusReportFrequency` [2073] configuration parameter. In this case, progress is reported in the cluster log, in terms of both the number of files and the amount of space that have been initialized, as shown here:

```
2009-06-20 16:39:23 [MgmSrvr] INFO -- Node 1: Local redo log file initialization status:
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15557
2009-06-20 16:39:23 [MgmSrvr] INFO -- Node 2: Local redo log file initialization status:
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15570
```

These reports are logged each `StartupStatusReportFrequency` [2073] seconds during Start Phase 4. If `StartupStatusReportFrequency` [2073] is 0 (the default), then reports are written to the cluster log only when at the beginning and at the completion of the redo log file initialization process.

Debugging Parameters. In MySQL Cluster NDB 7.2, it is possible to cause logging of traces for events generated by creating and dropping tables using `DictTrace`. This parameter is useful only in debugging NDB kernel code. `DictTrace` takes an integer value; currently, 0 (default - no logging) and 1 (logging enabled) are the only supported values.

Backup parameters. The `[ndbd]` parameters discussed in this section define memory buffers set aside for execution of online backups.

- `BackupDataBufferSize` [2074]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	bytes	16M	0 - 4G
Restart Type: N			

In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as `BackupWriteSize` [2075], the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens, the backup process pauses the scan and waits until some disk writes have completed freed up memory so that scanning may continue.

The default value for this parameter is 16MB.

- `BackupLogBufferSize` [2074]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	bytes	16M	0 - 4G
Restart Type: N			

The backup log buffer fulfills a role similar to that played by the backup data buffer, except that it is used for generating a log of all table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the backup log buffer must

be large enough to handle the load caused by write activities while the backup is being made. See [Section 17.5.3.3, “Configuration for MySQL Cluster Backups”](#).

The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster is unlikely to be able to perform the desired operations.

It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

The default value for this parameter is 16MB.

- [BackupMemory \[2075\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	bytes	32M	0 - 4G
Restart Type: N			

This parameter is simply the sum of [BackupDataBufferSize \[2074\]](#) and [BackupLogBufferSize \[2074\]](#).

The default value of this parameter in MySQL Cluster NDB 7.2 is 16MB + 16MB = 32MB.



Important

If [BackupDataBufferSize \[2074\]](#) and [BackupLogBufferSize \[2074\]](#) taken together exceed the default value for [BackupMemory \[2075\]](#), then this parameter must be set explicitly in the `config.ini` file to their sum.

- [BackupReportFrequency \[2075\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.2.3	seconds	0	0 - 4G
Restart Type: N			

This parameter controls how often backup status reports are issued in the management client during a backup, as well as how often such reports are written to the cluster log (provided cluster event logging is configured to permit it—see [Logging and checkpointing](#)). [BackupReportFrequency \[2075\]](#) represents the time in seconds between backup status reports.

The default value is 0.

- [BackupWriteSize \[2075\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	bytes	256K	2K - 4G
Restart Type: N			

This parameter specifies the default size of messages written to disk by the backup log and backup data buffers.

The default value for this parameter is 256KB.

- [BackupMaxWriteSize \[2076\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	1M	2K - 4G
NDB 7.2.1	bytes	1M	2K - 4G
Restart Type: N			

This parameter specifies the maximum size of messages written to disk by the backup log and backup data buffers.

The default value for this parameter is 1MB.



Important

When specifying these parameters, the following relationships must hold true. Otherwise, the data node will be unable to start.

- `BackupDataBufferSize >= BackupWriteSize + 188KB`
- `BackupLogBufferSize >= BackupWriteSize + 16KB`
- `BackupMaxWriteSize >= BackupWriteSize`

MySQL Cluster Realtime Performance Parameters

The `[ndbd]` parameters discussed in this section are used in scheduling and locking of threads to specific CPUs on multiprocessor data node hosts.



Note

To make use of these parameters, the data node process must be run as system root.

- [LockExecuteThreadToCPU \[2076\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.4	CPU ID	64K	0 - 64K
Restart Type: N			

When used with `ndbd`, this parameter (now a string) specifies the ID of the CPU assigned to handle the `NDBCLUSTER` execution thread. When used with `ndbmt_d`, the value of this parameter is a comma-separated list of CPU IDs assigned to handle execution threads. Each CPU ID in the list should be an integer in the range 0 to 65535 (inclusive).

The number of IDs specified should match the number of execution threads determined by [MaxNoOfExecutionThreads \[2079\]](#). However, there is no guarantee that threads are assigned to

CPUs in any given order when using this parameter; beginning with in MySQL Cluster NDB 7.2.5, you can obtain more finely-grained control of this type using [ThreadConfig \[2081\]](#).

[LockExecuteThreadToCPU \[2076\]](#) has no default value.

- [LockMaintThreadsToCPU \[2077\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.0	CPU ID	[none]	0 - 64K
Restart Type: N			

This parameter specifies the ID of the CPU assigned to handle `NDBCLUSTER` maintenance threads.

The value of this parameter is an integer in the range 0 to 65535 (inclusive). In MySQL Cluster NDB 7.2, there is no default value.

- [RealtimeScheduler \[2077\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.4	boolean	false	true, false
Restart Type: N			

Setting this parameter to 1 enables real-time scheduling of `NDBCLUSTER` threads.

The default is 0 (scheduling disabled).

- [SchedulerExecutionTimer \[2077\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.4	µsec	50	0 - 11000
Restart Type: N			

This parameter specifies the time in microseconds for threads to be executed in the scheduler before being sent. Setting it to 0 minimizes the response time; to achieve higher throughput, you can increase the value at the expense of longer response times.

The default is 50 µsec, which our testing shows to increase throughput slightly in high-load cases without materially delaying requests.

- [SchedulerSpinTimer \[2077\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.4	µsec	0	0 - 500
Restart Type: N			

This parameter specifies the time in microseconds for threads to be executed in the scheduler before sleeping.

The default value is 0.

- [BuildIndexThreads \[2078\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.11	numeric	0	0 - 128
Restart Type: S			

This parameter determines the number of threads to create when rebuilding indexes during a system or node start. It is supported only when there is more than one fragment for the table per data node (for example, when the `MAX_ROWS` option has been used with `CREATE TABLE`).

Setting this parameter to 0 (which is also the default value) disables multi-threaded building of ordered indexes. The maximum allowed value is 128.

This parameter is supported when using `ndbd` or `ndbmt.d`.

You can enable multi-threaded builds during data node initial restarts by setting the `TwoPassInitialNodeRestartCopy [2078]` data node configuration parameter to `TRUE`.

- [TwoPassInitialNodeRestartCopy \[2078\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.11	boolean	false	true, false
Restart Type: N			

Multi-threaded building of ordered indexes can be enabled for initial restarts of data nodes by setting this configuration parameter to `TRUE`, which enables two-pass copying of data during initial node restarts.

You must also set `BuildIndexThreads [2078]` to a nonzero value.

- [Numa \[2078\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.9	boolean	false	...
Restart Type: N			

`NDB` is extremely sensitive to Non-Uniform Memory Access settings and multi-CPU systems due to timeouts that it can cause. Due to this fact, and because most MySQL Cluster users do not employ `numactl`, support for NUMA is ignored by default by `ndbd` when running on a Linux system. If your Linux system provides NUMA support and you wish for data node memory to be subject to NUMA control, you can set this parameter equal to 0.

The `Numa [2078]` configuration parameter is supported only on Linux systems where `libnuma.so` is installed.

Multi-Threading Configuration Parameters (`ndbmt.d`). `ndbmt.d` runs by default as a single-threaded process and must be configured to use multiple threads, using either of two methods, both of which require setting configuration parameters in the `confi2078i` file. The first method is simply to set an appropriate

value for the `MaxNoOfExecutionThreads` [2079] configuration parameter. In MySQL Cluster NDB 7.2.3 and later, a second method is also supported, whereby it is possible to set up more complex rules for `ndbmt` multi-threading using `ThreadConfig` [2081]. The next few paragraphs provide information about these parameters and their use with multi-threaded data nodes.

- `MaxNoOfExecutionThreads`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	integer	2	2 - 8
NDB 7.2.5	integer	2	2 - 36
Restart Type: N			

Starting with MySQL Cluster NDB 7.2.5, this parameter directly controls the number of execution threads used by `ndbmt`, up to a maximum of 36. Although this parameter is set in `[ndbd]` or `[ndbd default]` sections of the `config.ini` file, it is exclusive to `ndbmt` and does not apply to `ndbd`.

Setting `MaxNoOfExecutionThreads` sets the number of threads by type as determined in the following table:

<code>MaxNoOfExecutionThreads</code> Value	LQH Threads	TC Threads	Send Threads	Receive Threads
0 .. 3	1	1	0	1
4 .. 6	2	1	0	1
7 .. 8	4	1	0	1
9	4	2	0	1
10	4	2	1	1
11	4	3	1	1
12	4	3	1	2
13	4	3	2	2
14	4	4	2	2
15	4	5	2	2
16	8	3	1	2
17	8	4	1	2
18	8	4	2	2
19	8	5	2	2
20	8	5	2	3
21	8	5	3	3
22	8	6	3	3
23	8	7	3	3
24	12	5	2	3
25	12	6	2	3
26	12	6	3	3
27	12	7	3	3

<code>MaxNoOfExecutionThreads</code> Value	LQH Threads	TC Threads	Send Threads	Receive Threads
28	12	7	3	4
29	12	8	3	4
30	12	8	4	4
31	12	9	4	4
32	16	8	3	3
33	16	8	3	4
34	16	8	4	4
35	16	9	4	4
36	16	10	4	4

There is always one SUMA (replication) thread. There was no separate send thread in MySQL Cluster NDB 7.2.4 and earlier, as well as no means of changing the number of TC threads.

The thread types are described later in this section (see [ThreadConfig \[2081\]](#)).

Setting this parameter outside the permitted range of values causes the management server to abort on startup with the error `Error line number: Illegal value value for parameter MaxNoOfExecutionThreads`.

For `MaxNoOfExecutionThreads`, a value of 0 or 1 is rounded up internally by NDB to 2, so that 2 is considered this parameter's default and minimum value.

`MaxNoOfExecutionThreads` is generally intended to be set equal to the number of CPU threads available, and to allocate a number of threads of each type suitable to typical workloads. It does not assign particular threads to specified CPUs. For cases where it is desirable to vary from the settings provided, or to bind threads to CPUs, you should use [ThreadConfig \[2081\]](#) instead, which allows you to allocate each thread directly to a desired type, CPU, or both.

In MySQL Cluster NDB 7.2.5 and later, the multi-threaded data node process always spawns at least 5 threads, listed here:

- 1 local query handler (LQH) thread
- 1 transaction coordinator (TC) thread
- 1 send thread

(It is possible to keep any separate send threads from being employed, as explained elsewhere in this section.)

- 1 receive thread
- 1 subscription manager (SUMA or replication) thread

In MySQL Cluster NDB 7.2.4 and earlier, there were only 4 thread types, with LDM threads being responsible for their own sends. In addition, it was not possible to cause `ndbmt_d` to use more than 1 TC thread.

- `NoOfFragmentLogParts`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.5	numeric	4	4, 8, 12, 16
Restart Type: IN			

Set the number of log file groups for redo logs belonging to this `ndbmtid`. The value must be an even multiple of 4 between 4 and 16, inclusive.

- [ThreadConfig](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.17	string	"	...
NDB 7.2.3	string	"	...
Restart Type: N			

This parameter is used with `ndbmtid` to assign threads of different types to different CPUs. Its value is a string whose format has the following syntax:

```
ThreadConfig := entry[,entry[,...]]
entry := type={param[,param[,...]]}
type := ldm | main | recv | send | rep | io
param := count=number | cpubind=cpu_list
```

The curly braces (`{...}`) surrounding the list of parameters is required, even if there is only one parameter in the list.

A `param` (parameter) specifies the number of threads of the given type (`count`), the CPUs to which the threads of the given type are to be bound (`cpubind`), or both.

The `type` attribute represents an NDB thread type. The thread types supported in MySQL Cluster NDB 7.2 and the range of permitted `count` values for each are provided in the following list:

- `ldm`: Local query handler (`DBLQH` kernel block) that handles data. The more LDM threads that are used, the more highly partitioned the data becomes. Each LDM thread maintains its own sets of data and index partitions, as well as its own redo log. In MySQL Cluster NDB 7.2, the maximum is 16 such threads (in MySQL Cluster NDB 7.1, the maximum was 4).

Range: 1 - 16.

- `tc`: Transaction coordinator thread (`DBTC` kernel block) containing the state of an ongoing transaction. In MySQL Cluster NDB 7.2.5 and later, the number of TC threads is configurable, with a total of 16 possible.

Optimally, every new transaction can be assigned to a new TC thread. In most cases 1 TC thread per 2 LDM threads is sufficient to guarantee that this can happen. In cases where the number of writes is relatively small when compared to the number of reads, it is possible that only 1 TC thread per 4 LQH threads is required to maintain transaction states. Conversely, in applications that perform a great many updates, it may be necessary for the ratio of TC threads to LDM threads to approach 1 (for example, 3 TC threads to 4 LDM threads).

Range: 1 - 16 (always 1 prior to MySQL Cluster 7.2.5).

- **main**: Data dictionary and transaction coordinator (**DBDIH** and **DBTC** kernel blocks), providing schema management. This is always handled by a single dedicated thread.

Range: 1 only.

- **recv**: Receive thread (**CMVMI** kernel block). Each receive thread handles one or more sockets for communicating with other nodes in a MySQL Cluster, with one socket per node. Previously, this was limited to a single thread, but MySQL Cluster 7.2 implements multiple receive threads (up to 8).

Range: 1 - 8.

- **send**: Send thread (**CMVMI** kernel block). Added in MySQL Cluster NDB 7.2.5. To increase throughput, it is possible in MySQL Cluster NDB 7.2.5 and later to perform sends from one or more separate, dedicated threads (maximum 8).

Previously, all threads handled their own sending directly; this can still be made to happen by setting the number of send threads to 0 (this also happens when **MaxNoOfExecutionThreads** [2079] is set equal to 9). While doing so can have an adverse impact on throughput, it can also in some cases provide decreased latency.

Range: 0 - 8.

- **rep**: Replication thread (**SUMA** kernel block). Asynchronous replication operations are always handled by a single dedicated thread.

Range: 1 only.

- **io**: File system and other miscellaneous operations. These are not demanding tasks, and are always handled as a group by a single, dedicated I/O thread.

Range: 1 only.

Simple examples:

```
# Example 1.
ThreadConfig=ldm={count=2,cpubind=1,2},main={cpubind=12},rep={cpubind=11}

# Example 2.
Threadconfig=main={cpubind=0},ldm={count=4,cpubind=1,2,5,6},io={count=2,cpubind=3,4}
```

It is usually desirable when configuring thread usage for a data node host to reserve one or more number of CPUs for operating system and other tasks. Thus, for a host machine with 24 CPUs, you might want to use 20 CPU threads (leaving 4 for other uses), with 8 LDM threads, 4 TC threads (half the number of LDM threads), 3 send threads, 3 receive threads, and 1 thread each for schema management, asynchronous replication, and I/O operations. (This is almost the same distribution of threads used when **MaxNoOfExecutionThreads** is set equal to 20.) The following **ThreadConfig** setting performs these assignments, additionally binding all of these threads to specific CPUs:

```
ThreadConfig=ldm{count=8,cpubind=1,2,3,4,5,6,7,8},main={cpubind=9},io={cpubind=9}, \
rep={cpubind=10},tc{count=4,cpubind=11,12,13,14},recv={count=3,cpubind=15,16,17}, \
send{count=3,cpubind=18,19,20}
```

It should be possible in most cases to bind the main (schema management) thread and the I/O thread to the same CPU, as we have done in the example just shown.

In order to take advantage of the enhanced stability that the use of `ThreadConfig` offers, it is necessary to insure that CPUs are isolated, and that they not subject to interrupts, or to being scheduled for other tasks by the operating system. On many Linux systems, you can do this by setting `IRQBALANCE_BANNED_CPUS` in `/etc/sysconfig/irqbalance` to `0xFFFFF0`, and by using the `isolcpus` boot option in `grub.conf`. For specific information, see your operating system or platform documentation.

Disk Data Configuration Parameters. Configuration parameters affecting Disk Data behavior include the following:

- `DiskPageBufferMemory` [2083]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	64M	4M - 1T
Restart Type: N			

This determines the amount of space used for caching pages on disk, and is set in the `[ndbd]` or `[ndbd default]` section of the `config.ini` file. It is measured in bytes. Each page takes up 32 KB. This means that Cluster Disk Data storage always uses $N * 32$ KB memory where N is some nonnegative integer.

The default value for this parameter is `64M` (2000 pages of 32 KB each).

You can query the `ndbinfo.diskpagebuffer` table to help determine whether the value for this parameter should be increased to minimize unnecessary disk seeks. See [Section 17.5.10.8, “The ndbinfo diskpagebuffer Table”](#), for more information.

- `SharedGlobalMemory` [2083]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	20M	0 - 64T
Restart Type: N			

This parameter determines the amount of memory that is used for log buffers, disk operations (such as page requests and wait queues), and metadata for tablespaces, log file groups, `UNDO` files, and data files. The shared global memory pool also provides memory used for satisfying the memory requirements of the `INITIAL_SIZE` and `UNDO_BUFFER_SIZE` options used with `CREATE LOGFILE GROUP` and `ALTER LOGFILE GROUP` statements, including any default value implied for these options by the setting of the `InitialLogFileGroup` [2085] data node configuration parameter. `SharedGlobalMemory` can be set in the `[ndbd]` or `[ndbd default]` section of the `config.ini` configuration file, and is measured in bytes.

As of MySQL Cluster NDB 7.2.0, the default value is `128M`. (Previously, this was `20M`.)

- `DiskIOThreadPool` [2083]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.0	threads	2	0 - 4G
Restart Type: N			

This parameter determines the number of unbound threads used for Disk Data file access. Before [DiskIOThreadPool \[2083\]](#) was introduced, exactly one thread was spawned for each Disk Data file, which could lead to performance issues, particularly when using very large data files. With [DiskIOThreadPool \[2083\]](#), you can—for example—access a single large data file using several threads working in parallel.

Currently, this parameter applies to Disk Data I/O threads only, but we plan in the future to make the number of such threads configurable for in-memory data as well.

The optimum value for this parameter depends on your hardware and configuration, and includes these factors:

- **Physical distribution of Disk Data files.** You can obtain better performance by placing data files, undo log files, and the data node file system on separate physical disks. If you do this with some or all of these sets of files, then you can set [DiskIOThreadPool \[2083\]](#) higher to enable separate threads to handle the files on each disk.
- **Disk performance and types.** The number of threads that can be accommodated for Disk Data file handling is also dependent on the speed and throughput of the disks. Faster disks and higher throughput allow for more disk I/O threads. Our test results indicate that solid-state disk drives can handle many more disk I/O threads than conventional disks, and thus higher values for [DiskIOThreadPool \[2083\]](#).

In MySQL Cluster NDB 7.2, the default value for this parameter is 2.

- **Disk Data file system parameters.** The parameters in the following list make it possible to place MySQL Cluster Disk Data files in specific directories without the need for using symbolic links.
 - [FileSystemPathDD \[2084\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.3	filename	[see text]	...
Restart Type: IN			

If this parameter is specified, then MySQL Cluster Disk Data data files and undo log files are placed in the indicated directory. This can be overridden for data files, undo log files, or both, by specifying values for [FileSystemPathDataFiles \[2084\]](#), [FileSystemPathUndoFiles \[2085\]](#), or both, as explained for these parameters. It can also be overridden for data files by specifying a path in the `ADD DATAFILE` clause of a `CREATE TABLESPACE` or `ALTER TABLESPACE` statement, and for undo log files by specifying a path in the `ADD UNDOFILE` clause of a `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP` statement. If [FileSystemPathDD \[2084\]](#) is not specified, then [FileSystemPath \[2042\]](#) is used.

If a [FileSystemPathDD \[2084\]](#) directory is specified for a given data node (including the case where the parameter is specified in the `[ndbd default]` section of the `config.ini` file), then starting that data node with `--initial` causes all files in the directory to be deleted.

- [FileSystemPathDataFiles \[2084\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.3	filename	[see text]	...

Effective Version	Type/Units	Default	Range or Permitted Values
Restart Type: IN			

If this parameter is specified, then MySQL Cluster Disk Data data files are placed in the indicated directory. This overrides any value set for `FileSystemPathDD` [2084]. This parameter can be overridden for a given data file by specifying a path in the `ADD DATAFILE` clause of a `CREATE TABLESPACE` or `ALTER TABLESPACE` statement used to create that data file. If `FileSystemPathDataFiles` [2084] is not specified, then `FileSystemPathDD` [2084] is used (or `FileSystemPath` [2042], if `FileSystemPathDD` [2084] has also not been set).

If a `FileSystemPathDataFiles` [2084] directory is specified for a given data node (including the case where the parameter is specified in the `[ndbd default]` section of the `config.ini` file), then starting that data node with `--initial` causes all files in the directory to be deleted.

- `FileSystemPathUndoFiles` [2085]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.3	filename	[see text]	...
Restart Type: IN			

If this parameter is specified, then MySQL Cluster Disk Data undo log files are placed in the indicated directory. This overrides any value set for `FileSystemPathDD` [2084]. This parameter can be overridden for a given data file by specifying a path in the `ADD UNDO` clause of a `CREATE LOGFILE GROUP` or `CREATE LOGFILE GROUP` statement used to create that data file. If `FileSystemPathUndoFiles` [2085] is not specified, then `FileSystemPathDD` [2084] is used (or `FileSystemPath` [2042], if `FileSystemPathDD` [2084] has also not been set).

If a `FileSystemPathUndoFiles` [2085] directory is specified for a given data node (including the case where the parameter is specified in the `[ndbd default]` section of the `config.ini` file), then starting that data node with `--initial` causes all files in the directory to be deleted.

For more information, see [Section 17.5.12.1, “MySQL Cluster Disk Data Objects”](#).

- **Disk Data object creation parameters.** The next two parameters enable you—when starting the cluster for the first time—to cause a Disk Data log file group, tablespace, or both, to be created without the use of SQL statements.
- `InitialLogFileGroup` [2085]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.3	string	[see text]	...
Restart Type: S			

This parameter can be used to specify a log file group that is created when performing an initial start of the cluster. `InitialLogFileGroup` [2085] is specified as shown here:

```
InitialLogFileGroup = [name=name;] [undo_buffer_size=size;] file-specification-list
file-specification-list:
    file-specification[; file-specification[; ...]]
```

```
file-specification:
  filename:size
```

The `name` of the log file group is optional and defaults to `DEFAULT-LG`. The `undo_buffer_size` is also optional; if omitted, it defaults to `64M`. Each `file-specification` corresponds to an undo log file, and at least one must be specified in the `file-specification-list`. Undo log files are placed according to any values that have been set for `FileSystemPath` [2042], `FileSystemPathDD` [2084], and `FileSystemPathUndoFiles` [2085], just as if they had been created as the result of a `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP` statement.

Consider the following:

```
InitialLogFileGroup = name=LG1; undo_buffer_size=128M; undo1.log:250M; undo2.log:150M
```

This is equivalent to the following SQL statements:

```
CREATE LOGFILE GROUP LG1
  ADD UNDOFILE 'undo1.log'
  INITIAL_SIZE 250M
  UNDO_BUFFER_SIZE 128M
  ENGINE NDBCLUSTER;

ALTER LOGFILE GROUP LG1
  ADD UNDOFILE 'undo2.log'
  INITIAL_SIZE 150M
  ENGINE NDBCLUSTER;
```

This logfile group is created when the data nodes are started with `--initial`.

Resources for the initial log file group are taken from the global memory pool whose size is determined by the value of the `SharedGlobalMemory` [2083] data node configuration parameter; if this parameter is set too low and the values set in `InitialLogFileGroup` for the logfile group's initial size or undo buffer size are too high, the cluster may fail to create the default log file group when starting, or fail to start altogether.

This parameter, if used, should always be set in the `[nbd default]` section of the `config.ini` file. The behavior of a MySQL Cluster when different values are set on different data nodes is not defined.

- `InitialTablespace` [2086]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.3	string	[see text]	...
Restart Type: S			

This parameter can be used to specify a MySQL Cluster Disk Data tablespace that is created when performing an initial start of the cluster. `InitialTablespace` [2086] is specified as shown here:

```
InitialTablespace = [name=name;] [extent_size=size;] file-specification-list
```

The `name` of the tablespace is optional and defaults to `DEFAULT-TS`. The `extent_size` is also optional; it defaults to `1M`. The `file-specification-list` uses the same syntax as shown with the `InitialLogFileGroup` [2085] parameter, the only difference being that each `file-specification` used with `InitialTablespace` [2086] corresponds to a data file. At least one must be specified in the `file-specification-list`. Data files are placed according to any values that have been set for `FileSystemPath` [2042], `FileSystemPathDD` [2084], and

[FileSystemPathDataFiles \[2084\]](#), just as if they had been created as the result of a `CREATE TABLESPACE` or `ALTER TABLESPACE` statement.

For example, consider the following line specifying [InitialTablespace \[2086\]](#) in the `[nbd default]` section of the `config.ini` file (as with [InitialLogfileGroup \[2085\]](#), this parameter should always be set in the `[nbd default]` section, as the behavior of a MySQL Cluster when different values are set on different data nodes is not defined):

```
InitialTablespace = name=TS1; extent_size=8M; data1.dat:2G; data2.dat:4G
```

This is equivalent to the following SQL statements:

```
CREATE TABLESPACE TS1
  ADD DATAFILE 'data1.dat'
  EXTENT_SIZE 8M
  INITIAL_SIZE 2G
  ENGINE NDBCLUSTER;

ALTER TABLESPACE TS1
  ADD UNDOFILE 'data2.dat'
  INITIAL_SIZE 4G
  ENGINE NDBCLUSTER;
```

This tablespace is created when the data nodes are started with `--initial`, and can be used whenever creating MySQL Cluster Disk Data tables thereafter.

Disk Data and GCP Stop errors. Errors encountered when using Disk Data tables such as `Node nodeid killed this node because GCP stop was detected` (error 2303) are often referred to as “GCP stop errors”. Such errors occur when the redo log is not flushed to disk quickly enough; this is usually due to slow disks and insufficient disk throughput.

You can help prevent these errors from occurring by using faster disks, and by placing Disk Data files on a separate disk from the data node file system. Reducing the value of [TimeBetweenGlobalCheckpoints \[2065\]](#) tends to decrease the amount of data to be written for each global checkpoint, and so may provide some protection against redo log buffer overflows when trying to write a global checkpoint; however, reducing this value also permits less time in which to write the GCP, so this must be done with caution.

In addition to the considerations given for [DiskPageBufferMemory \[2083\]](#) as explained previously, it is also very important that the [DiskIOThreadPool \[2083\]](#) configuration parameter be set correctly; having [DiskIOThreadPool \[2083\]](#) set too high is very likely to cause GCP stop errors (Bug #37227).

GCP stops can be caused by save or commit timeouts; the [TimeBetweenEpochsTimeout \[2065\]](#) data node configuration parameter determines the timeout for commits. However, it is possible to disable both types of timeouts by setting this parameter to 0.

Parameters for configuring send buffer memory allocation. Send buffer memory is allocated dynamically from a memory pool shared between all transporters, which means that the size of the send buffer can be adjusted as necessary. (Previously, the NDB kernel used a fixed-size send buffer for every node in the cluster, which was allocated when the node started and could not be changed while the node was running.) The [TotalSendBufferMemory \[2088\]](#) and [OverLoadLimit \[2096\]](#) data node configuration parameters permit the setting of limits on this memory allocation. For more information about the use of these parameters (as well as [SendBufferMemory \[2096\]](#)), see [Section 17.3.2.12, “Configuring MySQL Cluster Send Buffer Parameters”](#).

- [ExtraSendBufferMemory \[2087\]](#)

This parameter specifies the amount of transporter send buffer memory to allocate in addition to any set using [TotalSendBufferMemory \[2088\]](#), [SendBufferMemory \[2096\]](#), or both.

This parameter was added in MySQL Cluster NDB 7.2.5. (Bug #11760629, Bug #53053)

- [TotalSendBufferMemory \[2088\]](#)

This parameter is available beginning with MySQL Cluster NDB 6.4.0. It is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

Prior to MySQL Cluster NDB 7.2.5, this parameter did not work correctly with `ndbmt.d`. (Bug #13633845)

If this parameter is set, its minimum permitted value is 256KB; the maximum is 4294967039.

- [ReservedSendBufferMemory \[2088\]](#)

This parameter is present in `NDBCLUSTER` source code beginning with MySQL Cluster NDB 6.4.0. However, it is not currently enabled.

As of MySQL Cluster NDB 7.2.5, this parameter is deprecated, and is subject to removal in a future release of MySQL Cluster (Bug #11760629, Bug #53053).

For more detailed information about the behavior and use of [TotalSendBufferMemory \[2088\]](#) and about configuring send buffer memory parameters in MySQL Cluster, see [Section 17.3.2.12, “Configuring MySQL Cluster Send Buffer Parameters”](#).

See also [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#).

Redo log over-commit handling. It is possible to control a data node's handling of operations when too much time is taken flushing redo logs to disk. This occurs when a given redo log flush takes longer than [RedoOverCommitLimit \[2088\]](#) seconds, more than [RedoOverCommitCounter \[2088\]](#) times, causing any pending transactions to be aborted. When this happens, the API node that sent the transaction can handle the operations that should have been committed either by queuing the operations and re-trying them, or by aborting them, as determined by [DefaultOperationRedoProblemAction \[2094\]](#). The data node configuration parameters for setting the timeout and number of times it may be exceeded before the API node takes this action are described in the following list:

- [RedoOverCommitCounter \[2088\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.10	numeric	3	0 - 4G
Restart Type: N			

When [RedoOverCommitLimit \[2088\]](#) is exceeded when trying to write a given redo log to disk this many times or more, any transactions that were not committed as a result are aborted, and an API node where any of these transactions originated handles the operations making up those transactions according to its value for [DefaultOperationRedoProblemAction \[2094\]](#) (by either queuing the operations to be re-tried, or aborting them).

[RedoOverCommitCounter](#) defaults to 3. Set it to 0 to disable the limit.

- [RedoOverCommitLimit \[2088\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.10	seconds	20	0 - 4G

Effective Version	Type/Units	Default	Range or Permitted Values
Restart Type: N			

This parameter sets an upper limit in seconds for trying to write a given redo log to disk before timing out. The number of times the data node tries to flush this redo log, but takes longer than `RedoOverCommitLimit`, is kept and compared with `RedoOverCommitCounter` [2088], and when flushing takes too long more times than the value of that parameter, any transactions that were not committed as a result of the flush timeout are aborted. When this occurs, the API node where any of these transactions originated handles the operations making up those transactions according to its `DefaultOperationRedoProblemAction` [2094] setting (it either queues the operations to be re-tried, or aborts them).

By default, `RedoOverCommitLimit` is 20 seconds. Set to 0 to disable checking for redo log flush timeouts. This parameter was added in MySQL Cluster NDB 7.1.10.

Controlling restart attempts. It is possible to exercise finely-grained control over restart attempts by data nodes when they fail to start using the `MaxStartFailRetries` [2089] and `StartFailRetryDelay` [2089] data node configuration parameters.

`MaxStartFailRetries` [2089] limits the total number of retries made before giving up on starting the data node, `StartFailRetryDelay` [2089] sets the number of seconds between retry attempts. These parameters are described in more detail in the next few paragraphs.

`StartFailRetryDelay` [2089]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.2.19	unsigned	0	0 - 4G
Restart Type: N			

Use this parameter to set the number of seconds between restart attempts by the data node in the event on failure on startup. The default is 0 (no delay).



Note

This parameter is ignored unless `StopOnError` [2058] is equal to 0.

`MaxStartFailRetries` [2089]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.2.19	unsigned	3	0 - 4G
Restart Type: N			

Use this parameter to limit the number restart attempts made by the data node in the event that it fails on startup. The default is 3 attempts.



Note

This parameter is ignored unless `StopOnError` [2058] is equal to 0.

17.3.2.7. Defining SQL and Other API Nodes in a MySQL Cluster

The `[mysqld]` and `[api]` sections in the `config.ini` file define the behavior of the MySQL servers (SQL nodes) and other applications (API nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, any host can use this SQL or API node.

Generally speaking, a `[mysqld]` section is used to indicate a MySQL server providing an SQL interface to the cluster, and an `[api]` section is used for applications other than `mysqld` processes accessing cluster data, but the two designations are actually synonymous; you can, for instance, list parameters for a MySQL server acting as an SQL node in an `[api]` section.



Note

For a discussion of MySQL server options for MySQL Cluster, see [Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”](#); for information about MySQL server system variables relating to MySQL Cluster, see [Section 17.3.4.3, “MySQL Cluster System Variables”](#).

- `Id`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.1.1	unsigned	[none]	1 - 255
Restart Type: N			

The `Id` is an integer value used to identify the node in all cluster internal messages. The permitted range of values is 1 to 255 inclusive. This value must be unique for each node in the cluster, regardless of the type of node.



Note

Data node IDs must be less than 49, regardless of the MySQL Cluster version used. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for API nodes (and management nodes) to values greater than 48.

`NodeId` [2090] is the preferred parameter name to use when identifying API nodes. (`Id` continues to be supported for backward compatibility, but is now deprecated and generates a warning when used. It is also subject to future removal.)

- `NodeId`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.1.1	unsigned	[none]	1 - 255
Restart Type: N			

The `NodeId` is an integer value used to identify the node in all cluster internal messages. The permitted range of values is 1 to 255 inclusive. This value must be unique for each node in the cluster, regardless of the type of node.



Note

Data node IDs must be less than 49, regardless of the MySQL Cluster version used. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for API nodes (and management nodes) to values greater than 48.

[NodeId \[2090\]](#) is the preferred parameter name to use when identifying management nodes in MySQL Cluster NDB 7.2 and later. Previously, [Id](#) was used for this purpose and this continues to be supported for backward compatibility. [Id](#) is now deprecated and generates a warning when used; it is subject to removal in a future release of MySQL Cluster.

- [ExecuteOnComputer](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name	[none]	...
Restart Type: S			

This refers to the [Id](#) set for one of the computers (hosts) defined in a [\[computer\]](#) section of the configuration file.

- [HostName](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...
Restart Type: S			

Specifying this parameter defines the hostname of the computer on which the SQL node (API node) is to reside. To specify a hostname, either this parameter or [ExecuteOnComputer](#) is required.

If no [HostName](#) or [ExecuteOnComputer](#) is specified in a given [\[mysql\]](#) or [\[api\]](#) section of the [config.ini](#) file, then an SQL or API node may connect using the corresponding “slot” from any host which can establish a network connection to the management server host machine. *This differs from the default behavior for data nodes, where [localhost](#) is assumed for [HostName](#) unless otherwise specified.*

- [ArbitrationRank](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	0-2	0	0 - 2
Restart Type: N			

This parameter defines which nodes can act as arbitrators. Both MGM nodes and SQL nodes can be arbitrators. A value of 0 means that the given node is never used as an arbitrator, a value of 1 gives the node high priority as an arbitrator, and a value of 2 gives it low priority. A normal configuration uses the management server as arbitrator, setting its [ArbitrationRank](#) to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

By setting [ArbitrationRank](#) to 0 on all management and SQL nodes, you can disable arbitration completely. You can also control arbitration by overriding this parameter; to do so, set the [Arbitration \[2068\]](#) parameter in the [\[ndbd default\]](#) section of the [config.ini](#) global configuration file.

- [ArbitrationDelay](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	milliseconds	0	0 - 4G
Restart Type: N			

Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests will be delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- [BatchByteSize \[2092\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	32K	1024 - 1M
NDB 7.2.1	bytes	16K	1024 - 1M
Restart Type: N			

For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records ([BatchSize \[2092\]](#)) and in terms of bytes ([BatchByteSize \[2092\]](#)). The actual batch size is limited by both parameters.

The speed at which queries are performed can vary by more than 40% depending upon how this parameter is set. In future releases, MySQL Server will make educated guesses on how to set parameters relating to batch size, based on the query type.

This parameter is measured in bytes and by default is equal to 16KB in MySQL Cluster NDB 7.2.1 and later. (Previously, the default was 32K.)

- [BatchSize \[2092\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	records	64	1 - 992
NDB 7.2.1	records	256	1 - 992
Restart Type: N			

This parameter is measured in number of records and is by default set to 256 (MySQL Cluster NDB 7.2.1 and later; previously, the default was 64). The maximum size is 992.

- [HeartbeatThreadPriority \[2092\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.2	string	[none]	...
Restart Type: S			

Use this parameter to set the scheduling policy and priority of heartbeat threads for management and API nodes. The syntax for setting this parameter is shown here:

```
HeartbeatThreadPriority = policy[, priority]
```

```
policy:  
{FIFO | RR}
```

When setting this parameter, you must specify a policy. This is one of [FIFO](#) (first in, first in) or [RR](#) (round robin). This followed optionally by the priority (an integer).

- [MaxScanBatchSize](#) [2093]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	256K	32K - 16M
Restart Type: N			

The batch size is the size of each batch sent from each data node. Most scans are performed in parallel to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

The default value of this parameter is set to 256KB. Its maximum size is 16MB.

- [TotalSendBufferMemory](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.0	bytes	256K	0 - 4G
Restart Type: N			

This parameter is available beginning with MySQL Cluster NDB 6.4.0. It is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

If this parameter is set, its minimum permitted value is 256KB; the maximum is 4294967039. For more detailed information about the behavior and use of [TotalSendBufferMemory](#) and configuring send buffer memory parameters in MySQL Cluster, see [Section 17.3.2.12, “Configuring MySQL Cluster Send Buffer Parameters”](#).

- [AutoReconnect](#) [2093]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.7	boolean	false	true, false
Restart Type: N			

This parameter is `false` by default. This forces disconnected API nodes (including MySQL Servers acting as SQL nodes) to use a new connection to the cluster rather than attempting to re-use an existing one, as re-use of connections can cause problems when using dynamically-allocated node IDs. (Bug #45921)

**Note**

This parameter can be overridden using the NDB API. For more information, see `Ndb_cluster_connection::set_auto_reconnect()`, and `Ndb_cluster_connection::get_auto_reconnect()`.

- `DefaultOperationRedoProblemAction` [2094]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.1.10	enumeration	ABORT	ABORT, QUEUE
NDB 7.2.1	enumeration	ABORT	ABORT, QUEUE
NDB 7.2.10	enumeration	QUEUE	ABORT, QUEUE
Restart Type: S			

This parameter (along with `RedoOverCommitLimit` [2088] and `RedoOverCommitCounter` [2088]) controls the data node's handling of operations when too much time is taken flushing redo logs to disk. This occurs when a given redo log flush takes longer than `RedoOverCommitLimit` [2088] seconds, more than `RedoOverCommitCounter` [2088] times, causing any pending transactions to be aborted.

When this happens, the node can respond in either of two ways, according to the value of `DefaultOperationRedoProblemAction`, listed here:

- **ABORT**: Any pending operations from aborted transactions are also aborted.
- **QUEUE**: Pending operations from transactions that were aborted are queued up to be re-tried.

Prior to MySQL Cluster NDB 7.2.10, setting this parameter did not have any effect. (Bug #15855588)

You can obtain some information from a MySQL server running as a Cluster SQL node using `SHOW STATUS` in the `mysql` client, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_cluster_node_id | 5 |
| Ndb_config_from_host | 192.168.0.112 |
| Ndb_config_from_port | 1186 |
| Ndb_number_of_storage_nodes | 4 |
+-----+-----+
4 rows in set (0.02 sec)
```

For information about these Cluster system status variables, see [Section 5.1.6, “Server Status Variables”](#).

**Note**

To add new SQL or API nodes to the configuration of a running MySQL Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

17.3.2.8. MySQL Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for all connections between nodes in a MySQL Cluster. Normally it is not necessary to define TCP/IP connections; MySQL Cluster automatically sets up such connections for all data nodes, management nodes, and SQL or API nodes.



Note

For an exception to this rule, see [Section 17.3.2.9, “MySQL Cluster TCP/IP Connections Using Direct Connections”](#).

To override the default connection parameters, it is necessary to define a connection using one or more `[tcp]` sections in the `config.ini` file. Each `[tcp]` section explicitly defines a TCP/IP connection between two MySQL Cluster nodes, and must contain at a minimum the parameters `NodeId1` [2095] and `NodeId2` [2095], as well as any connection parameters to override.

It is also possible to change the default values for these parameters by setting them in the `[tcp default]` section.



Important

Any `[tcp]` sections in the `config.ini` file should be listed *last*, following all other sections in the file. However, this is not required for a `[tcp default]` section. This requirement is a known issue with the way in which the `config.ini` file is read by the MySQL Cluster management server.

Connection parameters which can be set in `[tcp]` and `[tcp default]` sections of the `config.ini` file are listed here:

- `NodeId1`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	numeric	[none]	...
Restart Type: N			

- `NodeId2`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	numeric	[none]	...
Restart Type: N			

To identify a connection between two nodes it is necessary to provide their node IDs in the `[tcp]` section of the configuration file. These are the same unique `NodeId` (or `Id`) values for each of these nodes as described in [Section 17.3.2.7, “Defining SQL and Other API Nodes in a MySQL Cluster”](#).

- `HostName1`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...

Effective Version	Type/Units	Default	Range or Permitted Values
Restart Type: N			

[HostName2](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...
Restart Type: N			

The [HostName1](#) and [HostName2](#) parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be hostnames or IP addresses.

- [OverloadLimit](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.0	bytes	0	0 - 4G
Restart Type: N			

When more than this many unsent bytes are in the send buffer, the connection is considered overloaded.

This parameter can be used to determine the amount of unsent data that must be present in the send buffer before the connection is considered overloaded. See [Section 17.3.2.12, “Configuring MySQL Cluster Send Buffer Parameters”](#), for more information.

In some older releases, the effective value of this parameter was limited by the size of [SendBufferMemory \[2096\]](#); in MySQL Cluster NDB 7.2, the actual value for [OverloadLimit](#) (up to the stated maximum of 4G) is used instead.

- [SendBufferMemory \[2096\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	unsigned	2M	256K - 4G
Restart Type: N			

TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these are also sent when a round of messages have been executed. To handle temporary overload situations it is also possible to define a bigger send buffer.

If this parameter is set explicitly, then the memory is not dedicated to each transporter; instead, the value used denotes the hard limit for how much memory (out of the total available memory—that is, [TotalSendBufferMemory](#)) that may be used by a single transporter. For more information about configuring dynamic transporter send buffer memory allocation in MySQL Cluster, see [Section 17.3.2.12, “Configuring MySQL Cluster Send Buffer Parameters”](#).

The default size of the send buffer in MySQL Cluster NDB 7.2 is 2MB, which is the size recommended in most situations. The minimum size is 64 KB; the theoretical maximum is 4 GB.

- [SendSignalId](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	boolean	[see text]	true, false
Restart Type: N			

To be able to retrace a distributed message datagram, it is necessary to identify each message. When this parameter is set to **Y**, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- [Checksum](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	boolean	false	true, false
Restart Type: N			

This parameter is a boolean parameter (enabled by setting it to **Y** or **1**, disabled by setting it to **N** or **0**). It is disabled by default. When it is enabled, checksums for all messages are calculated before they placed in the send buffer. This feature ensures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- [PortNumber](#) (*OBSOLETE*)

This formerly specified the port number to be used for listening for connections from other nodes. This parameter should no longer be used; use the [ServerPort](#) [2040] data node configuration parameter for this purpose instead.

- [ReceiveBufferMemory](#) [2097]

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.0.4	bytes	2M	16K - 4G
Restart Type: N			

Specifies the size of the buffer used when receiving data from the TCP/IP socket.

The default value of this parameter is 2MB. The minimum possible value is 16KB; the theoretical maximum is 4GB.

17.3.2.9. MySQL Cluster TCP/IP Connections Using Direct Connections

Setting up a cluster using direct connections between data nodes requires specifying explicitly the crossover IP addresses of the data nodes so connected in the `[tcp]` section of the cluster `config.ini` file.

In the following example, we envision a cluster with at least four hosts, one each for a management server, an SQL node, and two data nodes. The cluster as a whole resides on the `172.23.72.*` subnet of a LAN.

In addition to the usual network connections, the two data nodes are connected directly using a standard crossover cable, and communicate with one another directly using IP addresses in the `1.1.0.*` address range as shown:

```
# Management Server
[ndb_mgmd]
Id=1
HostName=172.23.72.20

# SQL Node
[mysqld]
Id=2
HostName=172.23.72.21

# Data Nodes
[ndbd]
Id=3
HostName=172.23.72.22

[ndbd]
Id=4
HostName=172.23.72.23

# TCP/IP Connections
[tcp]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

The `HostName1` [2095] and `HostName2` [2096] parameters are used only when specifying direct connections.

The use of direct TCP connections between data nodes can improve the cluster's overall efficiency by enabling the data nodes to bypass an Ethernet device such as a switch, hub, or router, thus cutting down on the cluster's latency. It is important to note that to take the best advantage of direct connections in this fashion with more than two data nodes, you must have a direct connection between each data node and every other data node in the same node group.

17.3.2.10. MySQL Cluster Shared-Memory Connections

MySQL Cluster attempts to use the shared memory transporter and configure it automatically where possible. `[shm]` sections in the `config.ini` file explicitly define shared-memory connections between nodes in the cluster. When explicitly defining shared memory as the connection method, it is necessary to define at least `NodeId1` [2098], `NodeId2` [2099], and `ShmKey` [2099]. All other parameters have default values that should work well in most cases.



Important

SHM functionality is considered experimental only. It is not officially supported in any current MySQL Cluster release, and testing results indicate that SHM performance is not appreciably greater than when using TCP/IP for the transporter.

For these reasons, you must determine for yourself or by using our free resources (forums, mailing lists) whether SHM can be made to work correctly in your specific case.

- `NodeId1`

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	numeric	[none]	...
Restart Type: N			

[NodeId2](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	numeric	[none]	...
Restart Type: N			

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as [NodeId1](#) and [NodeId2](#).

- [HostName1](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...
Restart Type: N			

[HostName2](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...
Restart Type: N			

The [HostName1](#) and [HostName2](#) parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be hostnames or IP addresses.

- [ShmKey \[2099\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	[none]	0 - 4G
Restart Type: N			

When setting up shared memory segments, a node ID, expressed as an integer, is used to identify uniquely the shared memory segment to use for the communication. There is no default value.

- [ShmSize \[2099\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	bytes	2099 1M	64K - 4G

Effective Version	Type/Units	Default	Range or Permitted Values
Restart Type: N			

Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by [ShmSize \[2099\]](#). The default value is 1MB.

- [SendSignalId](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	boolean	false	true, false
Restart Type: N			

To retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to `y` causes these message IDs to be transported over the network as well. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- [Checksum](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	boolean	true	true, false
Restart Type: N			

This parameter is a boolean (`y/N`) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before being placed in the send buffer.

This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

- [SigNum \[2100\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	[none]	0 - 4G
Restart Type: N			

When using the shared memory transporter, a process sends an operating system signal to the other process when there is new data available in the shared memory. Should that signal conflict with with an existing signal, this parameter can be used to change it. This is a possibility when using SHM due to the fact that different operating systems use different signal numbers.

The default value of [SigNum \[2100\]](#) is 0; therefore, it must be set to avoid errors in the cluster log when using the shared memory transporter. Typically, this parameter is set to 10 in the `[shm default]` section of the `config.ini` file.

17.3.2.11. SCI Transport Connections in MySQL Cluster

`[sci]` sections in the `config.ini` file explicitly define SCI (Scalable Coherent Interface) connections between cluster nodes. Using SCI transporters in MySQL Cluster is supported only when the MySQL

binaries are built using `--with-ndb-sci=/your/path/to/SCI`. The `path` should point to a directory that contains at a minimum `lib` and `include` directories containing SISC libraries and header files. (See [Section 17.3.5, “Using High-Speed Interconnects with MySQL Cluster”](#) for more information about SCI.)

In addition, SCI requires specialized hardware.

It is strongly recommended to use SCI Transporters only for communication between `ndbd` processes. Note also that using SCI Transporters means that the `ndbd` processes never sleep. For this reason, SCI Transporters should be used only on machines having at least two CPUs dedicated for use by `ndbd` processes. There should be at least one CPU per `ndbd` process, with at least one CPU left in reserve to handle operating system activities.

- [NodeId1](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	numeric	[none]	...
Restart Type: N			

[NodeId2](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	numeric	[none]	...
Restart Type: N			

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as [NodeId1](#) and [NodeId2](#).

- [Host1SciId0 \[2101\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	[none]	0 - 4G
Restart Type: N			

This identifies the SCI node ID on the first Cluster node (identified by [NodeId1 \[2101\]](#)).

- [Host1SciId1 \[2101\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	0	0 - 4G
Restart Type: N			

It is possible to set up SCI Transporters for failover between two SCI cards which then should use separate networks between the nodes. This identifies the node ID and the second SCI card to be used on the first node.

- [Host2SciId0 \[2101\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	[none]	0 - 4G
Restart Type: N			

This identifies the SCI node ID on the second Cluster node (identified by [NodeId2 \[2101\]](#)).

- [Host2SciId1 \[2102\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	0	0 - 4G
Restart Type: N			

When using two SCI cards to provide failover, this parameter identifies the second SCI card to be used on the second node.

- [HostName1](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...
Restart Type: N			

[HostName2](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	name or IP address	[none]	...
Restart Type: N			

The [HostName1](#) and [HostName2](#) parameters can be used to specify specific network interfaces to be used for a given SCI connection between two nodes. The values used for these parameters can be hostnames or IP addresses.

- [SharedBufferSize \[2102\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	10M	64K - 4G
Restart Type: N			

Each SCI transporter has a shared memory segment used for communication between the two nodes. Setting the size of this segment to the default value of 1MB should be sufficient for most applications. Using a smaller value can lead to problems when performing many parallel inserts; if the shared buffer is too small, this can also result in a crash of the [ndbd](#) process.

- [SendLimit \[2102\]](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	unsigned	8K	128 - 32K
Restart Type: N			

A small buffer in front of the SCI media stores messages before transmitting them over the SCI network. By default, this is set to 8KB. Our benchmarks show that performance is best at 64KB but 16KB reaches within a few percent of this, and there was little if any advantage to increasing it beyond 8KB.

- [SendSignalId](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.3.23	boolean	true	true, false
Restart Type: N			

To trace a distributed message it is necessary to identify each message uniquely. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- [Checksum](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 7.2.0	boolean	false	true, false
Restart Type: N			

This parameter is a boolean value, and is disabled by default. When [Checksum](#) is enabled, checksums are calculated for all messages before they are placed in the send buffer. This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

- [OverloadLimit](#)

Effective Version	Type/Units	Default	Range or Permitted Values
NDB 6.4.0	bytes	0	0 - 4G
Restart Type: N			

When more than this many unsent bytes are in the send buffer, the connection is considered overloaded. See [Section 17.3.2.12, “Configuring MySQL Cluster Send Buffer Parameters”](#), for more information.

17.3.2.12. Configuring MySQL Cluster Send Buffer Parameters

Formerly, the NDB kernel employed a send buffer whose size was fixed at 2MB for each node in the cluster, this buffer being allocated when the node started. Because the size of this buffer could not be changed after the cluster was started, it was necessary to make it large enough in advance to accommodate the maximum possible load on any transporter socket. However, this was an inefficient use of memory, since much of it often went unused, and could result in large amounts of resources being wasted when scaling up to many API nodes.

This problem was eventually solved (in MySQL Cluster NDB 7.0) by employing a unified send buffer whose memory is allocated dynamically from a pool shared by all transporters. This means that the size of the send buffer can be adjusted as necessary. Configuration of the unified send buffer can be accomplished by setting the following parameters:

- **TotalSendBufferMemory.** This parameter can be set for all types of MySQL Cluster nodes—that is, it can be set in the `[ndbd]`, `[mgm]`, and `[api]` (or `[mysql]`) sections of the `config.ini` file. It represents the total amount of memory (in bytes) to be allocated by each node for which it is set for use among all configured transporters. If set, its minimum is 256KB; the maximum is 4294967039.

To be backward-compatible with existing configurations, this parameter takes as its default value the sum of the maximum send buffer sizes of all configured transporters, plus an additional 32KB (one page) per transporter. The maximum depends on the type of transporter, as shown in the following table:

Transporter	Maximum Send Buffer Size (bytes)
TCP	<code>SendBufferMemory</code> [2096] (default = 2M)
SCI	<code>SendLimit</code> [2102] (default = 8K) plus 16K
SHM	20K

This enables existing configurations to function in close to the same way as they did with MySQL Cluster NDB 6.3 and earlier, with the same amount of memory and send buffer space available to each transporter. However, memory that is unused by one transporter is not available to other transporters.

- **OverloadLimit.** This parameter is used in the `config.ini` file `[tcp]` section, and denotes the amount of unsent data (in bytes) that must be present in the send buffer before the connection is considered overloaded. When such an overload condition occurs, transactions that affect the overloaded connection fail with NDB API Error 1218 (`Send Buffers overloaded in NDB kernel`) until the overload status passes. The default value is 0, in which case the effective overload limit is calculated as `SendBufferMemory * 0.8` for a given connection. The maximum value for this parameter is 4G.
- **SendBufferMemory.** This value denotes a hard limit for the amount of memory that may be used by a single transporter out of the entire pool specified by `TotalSendBufferMemory` [2088]. However, the sum of `SendBufferMemory` for all configured transporters may be greater than the `TotalSendBufferMemory` [2088] that is set for a given node. This is a way to save memory when many nodes are in use, as long as the maximum amount of memory is never required by all transporters at the same time.

17.3.3. Overview of MySQL Cluster Configuration Parameters

The next four sections provide summary tables of MySQL Cluster configuration parameters used in the `config.ini` file to govern the cluster's functioning. Each table lists the parameters for one of the Cluster node process types (`ndbd`, `ndb_mgmd`, and `mysqld`), and includes the parameter's type as well as its default, minimum, and maximum values as applicable.

These tables also indicate what type of restart is required (node restart or system restart)—and whether the restart must be done with `--initial`—to change the value of a given configuration parameter.

When performing a node restart or an initial node restart, all of the cluster's data nodes must be restarted in turn (also referred to as a *rolling restart*). It is possible to update cluster configuration parameters marked as `node` online—that is, without shutting down the cluster—in this fashion. An initial node restart requires restarting each `ndbd` process with the `--initial` option.

A system restart requires a complete shutdown and restart of the entire cluster. An initial system restart requires taking a backup of the cluster, wiping the cluster file system after shutdown, and then restoring from the backup following the restart.

In any cluster restart, all of the cluster's management servers must be restarted for them to read the updated configuration parameter values.



Important

Values for numeric cluster parameters can generally be increased without any problems, although it is advisable to do so progressively, making such adjustments in relatively small increments. Many of these can be increased online, using a rolling restart.

However, decreasing the values of such parameters—whether this is done using a node restart, node initial restart, or even a complete system restart of the cluster—is not to be undertaken lightly; it is recommended that you do so only after careful planning and testing. This is especially true with regard to those parameters that relate to memory usage and disk space, such as [MaxNoOfTables \[2054\]](#), [MaxNoOfOrderedIndexes \[2055\]](#), and [MaxNoOfUniqueHashIndexes \[2055\]](#). In addition, it is generally the case that configuration parameters relating to memory and disk usage can be raised using a simple node restart, but they require an initial node restart to be lowered.

Because some of these parameters can be used for configuring more than one type of cluster node, they may appear in more than one of the tables.



Note

`4294967039` often appears as a maximum value in these tables. This value is defined in the `NDBCLUSTER` sources as `MAX_INT_RNIL` and is equal to `0xFFFFFFFF`, or $2^{32} - 2^8 - 1$.

17.3.3.1. MySQL Cluster Data Node Configuration Parameters

The summary table in this section provides information about parameters used in the `[ndbd]` or `[ndbd default]` sections of a `config.ini` file for configuring MySQL Cluster data nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”](#).

These parameters also apply to `ndbmt_d`, the multi-threaded version of `ndbd`. For more information, see [Section 17.4.3, “ndbmt_d — The MySQL Cluster Data Node Daemon \(Multi-Threaded\)”](#).

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- **N**—Node restart: The parameter can be updated using a rolling restart (see [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#)).
- **S**—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- **I**—Initial restart: Data nodes must be restarted using the `--initial [2178]` option.

For more information about restart types, see [Section 17.3.3, “Overview of MySQL Cluster Configuration Parameters”](#).

MySQL Cluster NDB 7.2 supports the addition of new data node groups online, to a running cluster. For more information, see [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#).

Table 17.1. Data Node Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
Arbitration [2068]	enumeration	N	NDB 7.2.0
	Default		
	Default, Disabled, WaitExternal		
ArbitrationTimeout [2068]	milliseconds	N	NDB 7.2.0
	7500		
	10 / 4G		
BackupDataBufferSize [2074]	bytes	N	NDB 7.2.0
	16M		
	0 / 4G		
BackupDataDir [2042]	path	IN	NDB 7.2.0
	FileSystemPath		
	...		
BackupLogBufferSize [2074]	bytes	N	NDB 7.2.0
	16M		
	0 / 4G		
BackupMaxWriteSize [2076]	bytes	N	NDB 7.2.1
	1M		
	2K / 4G		
BackupMemory [2075]	bytes	N	NDB 7.2.0
	32M		
	0 / 4G		
BackupReportFrequency [2075]	seconds	N	NDB 7.2.0
	0		
	0 / 4G		
BackupWriteSize [2075]	bytes	N	NDB 7.2.0
	256K		
	2K / 4G		
BatchSizePerLocalScan [2050]	integer	N	NDB 7.2.1
	256		
	1 / 992		
BuildIndexThreads [2078]	numeric	S	NDB 7.2.0

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	0		
	0 / 128		
CompressedBackup [2059]	boolean	N	NDB 7.2.0
	false		
	true, false		
CompressedLCP [2060]	boolean	N	NDB 7.2.0
	false		
	true, false		
ConnectCheckIntervalDelay [2064]	string	N	NDB 7.2.1
	0		
	0 / 4G		
CrashOnCorruptedTuple [2058]	boolean	S	NDB 7.2.1
	true		
	true, false		
DataDir [2042]	path	IN	NDB 7.2.0
	.		
	...		
DataMemory [2043]	bytes	N	NDB 7.2.0
	80M		
	1M / 1024G		
DictTrace	bytes	N	NDB 7.2.0
	undefined		
	0 / 100		
DiskCheckpointSpeed [2067]	bytes	N	NDB 7.2.0
	10M		
	1M / 4G		
DiskCheckpointSpeedInRestart [2068]	bytes	N	NDB 7.2.0
	100M		
	1M / 4G		
DiskIOThreadPool [2083]	threads	N	NDB 7.2.0
	2		
	0 / 4G		
Diskless [2058]	true false (1 0)	IS	NDB 7.2.0
	false		

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	true, false		
DiskPageBufferMemory [2083]	bytes	N	NDB 7.2.0
	64M		
	4M / 1T		
DiskSyncSize [2067]	bytes	N	NDB 7.2.0
	4M		
	32K / 4G		
ExecuteOnComputer [2039]	name	S	NDB 7.2.0
	[none]		
	...		
ExtraSendBufferMemory [2087]	bytes	N	NDB 7.2.0
	0		
	0 / 32G		
FileSystemPath [2042]	path	IN	NDB 7.2.0
	DataDir		
	...		
FileSystemPathDataFiles [2084]	filename	IN	NDB 7.2.0
	[see text]		
	...		
FileSystemPathDD [2084]	filename	IN	NDB 7.2.0
	FileSystemPath		
	...		
FileSystemPathUndoFiles [2085]	filename	IN	NDB 7.2.0
	[see text]		
	...		
FragmentLogFileSize [2052]	bytes	IN	NDB 7.2.0
	16M		
	4M / 1G		
HeartbeatIntervalDbApi [2062]	milliseconds	N	NDB 7.2.0
	1500		
	100 / 4G		
HeartbeatIntervalDbDb [2062]	milliseconds	N	NDB 7.2.0
	5000		
	10 / 4G		

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
HeartbeatOrder [2063]	numeric	S	NDB 7.2.0
	0		
	0 / 65535		
HostName [2040]	name or IP address	S	NDB 7.2.0
	localhost		
	...		
Id [2039]	unsigned	N	NDB 7.2.0
	[none]		
	1 / 48		
IndexMemory [2044]	bytes	N	NDB 7.2.0
	18M		
	1M / 1T		
InitFragmentLogFiles [2052]	[see values]	IN	NDB 7.2.0
	SPARSE		
	SPARSE, FULL		
InitialLogFileGroup [2085]	string	S	NDB 7.2.0
	[see text]		
	...		
InitialNoOfOpenFiles [2053]	files	N	NDB 7.2.0
	27		
	20 / 4G		
InitialTablespace [2086]	string	S	NDB 7.2.0
	[see text]		
	...		
IOThreadPool [2083]	threads	N	NDB 7.2.0
	8		
	0 / 4G		
LockExecuteThreadToCPU [2076]	CPU ID	N	NDB 7.2.0
	64K		
	0 / 64K		
LockMaintThreadsToCPU [2077]	CPU ID	N	NDB 7.2.0
	[none]		
	0 / 64K		

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
LockPagesInMainMemory [2057]	numeric	N	NDB 7.2.0
	0		
	0 / 2		
LogLevelCheckpoint [2072]	log level	N	NDB 7.2.0
	0		
	0 / 15		
LogLevelCongestion [2072]	levelr	N	NDB 7.2.0
	0		
	0 / 15		
LogLevelConnection [2072]	integer	N	NDB 7.2.0
	0		
	0 / 15		
LogLevelError [2072]	integer	N	NDB 7.2.0
	0		
	0 / 15		
LogLevelInfo [2073]	integer	N	NDB 7.2.0
	0		
	0 / 15		
LogLevelNodeRestart [2072]	integer	N	NDB 7.2.0
	0		
	0 / 15		
LogLevelShutdown [2071]	integer	N	NDB 7.2.0
	0		
	0 / 15		
LogLevelStartup [2071]	integer	N	NDB 7.2.0
	1		
	0 / 15		
LogLevelStatistic [2071]	integer	N	NDB 7.2.0
	0		
	0 / 15		
LongMessageBuffer [2050]	bytes	N	NDB 7.2.0
	4M		
	512K / 4G		
MaxAllocate [2051]	unsigned	N	NDB 7.2.0

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	32M		
	1M / 1G		
MaxBufferedEpochs [2066]	epochs	N	NDB 7.2.0
	100		
	0 / 100000		
MaxDMLOperationsPerTransaction [2048]	operations (DML)	N	NDB 7.2.0
	4294967295		
	32 / 4294967295		
MaxLCPStartDelay [2053]	seconds	N	NDB 7.2.0
	0		
	0 / 600		
MaxNoOfAttributes [2054]	integer	N	NDB 7.2.0
	1000		
	32 / 4G		
MaxNoOfConcurrentIndexOperations [2048]	integer	N	NDB 7.2.0
	8K		
	0 / 4G		
MaxNoOfConcurrentOperations [2047]	integer	N	NDB 7.2.0
	32K		
	32 / 4G		
MaxNoOfConcurrentScans [2049]	integer	N	NDB 7.2.0
	256		
	2 / 500		
MaxNoOfConcurrentSubOperations [2057]	unsigned	N	NDB 7.2.0
	256		
	0 / 4G		
MaxNoOfConcurrentTransactions [2046]	integer	S	NDB 7.2.0
	4096		
	32 / 4G		
MaxNoOfFiredTriggers [2049]	integer	N	NDB 7.2.0
	4000		
	0 / 4G		

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
MaxNoOfLocalOperations [2048]	integer	N	NDB 7.2.0
	UNDEFINED		
	32 / 4G		
MaxNoOfLocalScans [2050]	integer	N	NDB 7.2.0
	UNDEFINED		
	32 / 4G		
MaxNoOfOpenFiles [2053]	unsigned	N	NDB 7.2.0
	0		
	20 / 4G		
MaxNoOfOrderedIndexes [2055]	integer	N	NDB 7.2.0
	128		
	0 / 4G		
MaxNoOfSavedMessages [2053]	integer	N	NDB 7.2.0
	25		
	0 / 4G		
MaxNoOfSubscribers [2057]	unsigned	N	NDB 7.2.0
	0		
	0 / 4G		
MaxNoOfSubscriptions [2056]	unsigned	N	NDB 7.2.0
	0		
	0 / 4G		
MaxNoOfTables [2054]	integer	N	NDB 7.2.0
	128		
	8 / 20320		
MaxNoOfTriggers [2056]	integer	N	NDB 7.2.0
	768		
	0 / 4G		
MaxNoOfUniqueHashIndexes [2055]	integer	N	NDB 7.2.0
	64		
	0 / 4G		
MaxParallelScansPerFragment [2051]	bytes	N	NDB 7.2.0
	256		
	1 / 1G		
MaxStartFailRetries [2089]	unsigned	N	NDB 7.2.0

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	3		
	0 / 4G		
MemReportFrequency [2073]	unsigned	N	NDB 7.2.0
	0		
	0 / 4G		
MinFreePct [2045]	unsigned	N	NDB 7.2.3
	5		
	0 / 100		
NodeGroup [2040]	[none]	IS	NDB 7.2.0
	0 / 65536		
NodeId [2039]	unsigned	N	NDB 7.2.0
	[none]		
	1 / 48		
NoOfFragmentLogFiles [2051]	integer	IN	NDB 7.2.0
	16		
	3 / 4G		
NoOfReplicas [2041]	integer	IS	NDB 7.2.0
	2		
	1 / 4		
Numa [2078]	boolean	N	NDB 7.2.0
	false		
	...		
ODirect [2059]	boolean	N	NDB 7.2.0
	false		
	true, false		
RealtimeScheduler [2077]	boolean	N	NDB 7.2.0
	false		
	true, false		
RedoBuffer [2070]	bytes	N	NDB 7.2.0
	32M		
	1M / 4G		
RedoOverCommitCounter [2088]	numeric	N	NDB 7.2.0
	3		

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	0 / 4G		
RedoOverCommitLimit [2088]	seconds	N	NDB 7.2.0
	20		
	0 / 4G		
ReservedSendBufferMemory [2088]	bytes	N	NDB 7.2.0
	256K		
	0 / 4G		
RestartOnErrorInsert [2059]	error code	N	NDB 7.2.0
	2		
	0 / 4		
SchedulerExecutionTimer [2077]	µsec	N	NDB 7.2.0
	50		
	0 / 11000		
SchedulerSpinTimer [2077]	µsec	N	NDB 7.2.0
	0		
	0 / 500		
ServerPort [2040]	unsigned	N	NDB 7.2.0
	[none]		
	1 / 64K		
SharedGlobalMemory [2083]	bytes	N	NDB 7.2.0
	128M		
	0 / 64T		
StartFailRetryDelay [2089]	unsigned	N	NDB 7.2.0
	0		
	0 / 4G		
StartFailureTimeout [2061]	milliseconds	N	NDB 7.2.0
	0		
	0 / 4G		
StartNoNodeGroupTimeout [2062]	milliseconds	N	NDB 7.2.0
	15000		
	0 / 4294967039		
StartPartialTimeout [2061]	milliseconds	N	NDB 7.2.0
	30000		
	0 / 4G		

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
StartPartitionedTimeout [2061]	milliseconds	N	NDB 7.2.0
	60000		
	0 / 4G		
StartupStatusReportFrequency [2073]	seconds	N	NDB 7.2.0
	[none]		
	...		
StopOnError [2058]	boolean	N	NDB 7.2.0
	true		
	true, false		
StringMemory [2044]	% or bytes	S	NDB 7.2.1
	25		
	0 / 4G		
TcpBind_INADDR_ANY [2040]	boolean	N	NDB 7.2.0
	false		
	true, false		
TimeBetweenEpochs [2065]	milliseconds	N	NDB 7.2.0
	100		
	0 / 32000		
TimeBetweenEpochsTimeout [2065]	milliseconds	N	NDB 7.2.0
	4000		
	0 / 256000		
TimeBetweenGlobalCheckpoints [2065]	milliseconds	N	NDB 7.2.0
	2000		
	20 / 32000		
TimeBetweenInactiveTransactionAbortCheck [2066]	milliseconds	N	NDB 7.2.0
	1000		
	1000 / 4G		
TimeBetweenLocalCheckpoints [2064]	number of 4-byte words, as a base-2 logarithm	N	NDB 7.2.0
	20		
	0 / 31		
TimeBetweenWatchDogCheck [2060]	milliseconds	N	NDB 7.2.0

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	6000		
	70 / 4G		
TimeBetweenWatchDogCheckInitial [2060]	milliseconds	N	NDB 7.2.0
	6000		
	70 / 4G		
TotalSendBufferMemory [2088]	bytes	N	NDB 7.2.0
	256K		
	0 / 4G		
TransactionBufferMemory [2049]	bytes	N	NDB 7.2.0
	1M		
	1K / 4G		
TransactionDeadlockDetectionTimeout [2067]	milliseconds	N	NDB 7.2.0
	1200		
	50 / 4G		
TransactionInactiveTimeout [2066]	milliseconds	N	NDB 7.2.0
	4G		
	0 / 4G		
TwoPassInitialNodeRestartCopy [2078]	boolean	N	NDB 7.2.0
	false		
	true, false		
UndoDataBuffer [2070]	unsigned	N	NDB 7.2.0
	16M		
	1M / 4G		
UndoIndexBuffer [2069]	unsigned	N	NDB 7.2.0
	2M		
	1M / 4G		

Table 17.2. Multi-Threaded Data Node Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
MaxNoOfExecutionThreads [2079]	integer	N	NDB 7.2.5

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	2		
	2 / 36		
<code>NoOfFragmentLogParts</code> [2080]	numeric	IN	NDB 7.2.5
	4		
	4, 8, 12, 16		
<code>ThreadConfig</code> [2081]	string	N	NDB 7.2.3
	"		
	...		

17.3.3.2. MySQL Cluster Management Node Configuration Parameters

The summary table in this section provides information about parameters used in the `[ndb_mgmd]` or `[mgm]` sections of a `config.ini` file for configuring MySQL Cluster management nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 17.3.2.5, “Defining a MySQL Cluster Management Server”](#).

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- **N**—Node restart: The parameter can be updated using a rolling restart (see [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#)).
- **S**—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- **I**—Initial restart: Data nodes must be restarted using the `--initial` [2178] option.

For more information about restart types, see [Section 17.3.3, “Overview of MySQL Cluster Configuration Parameters”](#).

Table 17.3. Management Node Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
<code>ArbitrationDelay</code> [2037]	milliseconds	N	NDB 7.2.0
	0		
	0 / 4G		
<code>ArbitrationRank</code> [2037]	0-2	N	NDB 7.2.0
	1		

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	0 / 2		
DataDir [2037]	path	N	NDB 7.2.0
ExecuteOnComputer [2035]	name [none] ...	S	NDB 7.2.0
HeartbeatThreadPriority [2037]	string [none] ...	S	NDB 7.2.0
HostName [2035]	name or IP address [none] ...	S	NDB 7.2.0
Id [2034]	unsigned [none] 1 / 255	N	NDB 7.2.0
LogDestination [2036]	{CONSOLE SYSLOG FILE} [see text] ...	N	NDB 7.2.0
MaxNoOfSavedEvents	unsigned 100 0 / 4G	N	NDB 7.2.0
NodeId [2035]	unsigned [none] 1 / 255	N	NDB 7.2.0
PortNumber [2035]	unsigned 1186 0 / 64K	N	NDB 7.2.0
PortNumberStats	unsigned [none] 0 / 64K	N	NDB 7.2.0
TotalSendBufferMemory [2038]	bytes	N	NDB 7.2.0

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	256K		
	0 / 4G		
wan	boolean	N	NDB 7.2.0
	false		
	true, false		



Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect. See [Section 17.3.2.5, “Defining a MySQL Cluster Management Server”](#), for more information.

To add new management servers to a running MySQL Cluster, it is also necessary perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple management nodes, see [Section 17.1.6.10, “Limitations Relating to Multiple MySQL Cluster Nodes”](#).

17.3.3.3. MySQL Cluster SQL Node and API Node Configuration Parameters

The summary table in this section provides information about parameters used in the `[mysqld]` and `[api]` sections of a `config.ini` file for configuring MySQL Cluster SQL nodes and API nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 17.3.2.7, “Defining SQL and Other API Nodes in a MySQL Cluster”](#).



Note

For a discussion of MySQL server options for MySQL Cluster, see [Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”](#); for information about MySQL server system variables relating to MySQL Cluster, see [Section 17.3.4.3, “MySQL Cluster System Variables”](#).

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- **N**—Node restart: The parameter can be updated using a rolling restart (see [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#)).
- **S**—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- **I**—Initial restart: Data nodes must be restarted using the `--initial [2178]` option.

For more information about restart types, see [Section 17.3.3, “Overview of MySQL Cluster Configuration Parameters”](#).

Table 17.4. SQL Node / API Node Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
ArbitrationDelay [2091]	milliseconds	N	NDB 7.2.0
	0		
	0 / 4G		
ArbitrationRank [2091]	0-2	N	NDB 7.2.0
	0		
	0 / 2		
AutoReconnect [2093]	boolean	N	NDB 7.2.0
	false		
	true, false		
BatchByteSize [2092]	bytes	N	NDB 7.2.1
	16K		
	1024 / 1M		
BatchSize [2092]	records	N	NDB 7.2.1
	256		
	1 / 992		
ConnectionMap	string	N	NDB 7.2.0
	[none]		
	...		
DefaultOperationRedoProblemAction [2094]	enumeration	S	NDB 7.2.10
	QUEUE		
	ABORT, QUEUE		
ExecuteOnComputer [2091]	name	S	NDB 7.2.0
	[none]		
	...		
HeartbeatThreadPriority [2092]	string	S	NDB 7.2.0
	[none]		
	...		
HostName [2091]	name or IP address	S	NDB 7.2.0
	[none]		
	...		
Id [2090]	unsigned	N	NDB 7.2.0

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
	[none]		
	1 / 255		
MaxScanBatchSize [2093]	bytes	N	NDB 7.2.0
	256K		
	32K / 16M		
NodeId [2090]	unsigned	N	NDB 7.2.0
	[none]		
	1 / 255		
TotalSendBufferMemory [2093]	bytes	N	NDB 7.2.0
	256K		
	0 / 4G		
wan	boolean	N	NDB 7.2.0
	false		
	true, false		



Note

To add new SQL or API nodes to the configuration of a running MySQL Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new [mysqld] or [api] sections to the config.ini file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

17.3.3.4. Other MySQL Cluster Configuration Parameters

The summary tables in this section provide information about parameters used in the [computer], [tcp], [shm], and [sci] sections of a config.ini file for configuring MySQL Cluster management nodes. For detailed descriptions and other additional information about individual parameters, see Section 17.3.2.8, “MySQL Cluster TCP/IP Connections”, Section 17.3.2.10, “MySQL Cluster Shared-Memory Connections”, or Section 17.3.2.11, “SCI Transport Connections in MySQL Cluster”, as appropriate.

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary tables as follows:

- **N**—Node restart: The parameter can be updated using a rolling restart (see Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”).

- **S**—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- **I**—Initial restart: Data nodes must be restarted using the `--initial` [2178] option.

For more information about restart types, see [Section 17.3.3, “Overview of MySQL Cluster Configuration Parameters”](#).

Table 17.5. Computer Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
HostName [2034]	name or IP address	S	NDB 7.2.0
	[none]		
	...		
Id [2034]	string	IN	NDB 7.2.0
	[none]		
	...		

Table 17.6. TCP Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
Checksum [2097]	boolean	N	NDB 7.2.0
	false		
	true, false		
Group	unsigned	N	NDB 7.2.0
	55		
	0 / 200		
NodeId1 [2095]	numeric	N	NDB 7.2.0
	[none]		
	...		
NodeId2 [2095]	numeric	N	NDB 7.2.0
	[none]		
	...		
NodeIdServer	numeric	N	NDB 7.2.0
	[none]		
	...		

Overview of MySQL Cluster Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
OverloadLimit [2096]	bytes	N	NDB 7.2.0
	0		
	0 / 4G		
PortNumber [2097]	unsigned	N	NDB 7.2.0
	[none]		
	0 / 64K		
Proxy	string	N	NDB 7.2.0
	[none]		
	...		
ReceiveBufferMemory [2097]	bytes	N	NDB 7.2.0
	2M		
	16K / 4G		
SendBufferMemory [2096]	unsigned	N	NDB 7.2.0
	2M		
	256K / 4G		
SendSignalId [2097]	boolean	N	NDB 7.2.0
	[see text]		
	true, false		
TCP_MAXSEG_SIZE	unsigned	N	NDB 7.2.0
	0		
	0 / 2G		
TCP_RCV_BUF_SIZE	unsigned	N	NDB 7.2.0
	70080		
	1 / 2G		
TCP_SND_BUF_SIZE	unsigned	N	NDB 7.2.0
	71540		
	1 / 2G		
TcpBind_INADDR_ANY	boolean	N	NDB 7.2.0
	false		
	true, false		

Table 17.7. Shared Memory Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
Checksum [2100]	boolean	N	NDB 7.2.0
	true		
	true, false		
Group	unsigned	N	NDB 7.2.0
	35		
	0 / 200		
NodeId1 [2098]	numeric	N	NDB 7.2.0
	[none]		
	...		
NodeId2 [2099]	numeric	N	NDB 7.2.0
	[none]		
	...		
NodeIdServer	numeric	N	NDB 7.2.0
	[none]		
	...		
OverloadLimit	bytes	N	NDB 7.2.0
	0		
	0 / 4G		
PortNumber	unsigned	N	NDB 7.2.0
	[none]		
	0 / 64K		
SendSignalId [2100]	boolean	N	NDB 7.2.0
	false		
	true, false		
ShmKey [2099]	unsigned	N	NDB 7.2.0
	[none]		
	0 / 4G		
ShmSize [2099]	bytes	N	NDB 7.2.0
	1M		
	64K / 4G		
Signum [2100]	unsigned	N	NDB 7.2.0
	[none]		
	0 / 4G		

Table 17.8. SCI Configuration Parameters

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
Checksum [2103]	boolean	N	NDB 7.2.0
	false		
	true, false		
Group	unsigned	N	NDB 7.2.0
	15		
	0 / 200		
Host1SciId0 [2101]	unsigned	N	NDB 7.2.0
	[none]		
	0 / 4G		
Host1SciId1 [2101]	unsigned	N	NDB 7.2.0
	0		
	0 / 4G		
Host2SciId0 [2101]	unsigned	N	NDB 7.2.0
	[none]		
	0 / 4G		
Host2SciId1 [2102]	unsigned	N	NDB 7.2.0
	0		
	0 / 4G		
NodeId1 [2101]	numeric	N	NDB 7.2.0
	[none]		
	...		
NodeId2 [2101]	numeric	N	NDB 7.2.0
	[none]		
	...		
NodeIdServer	numeric	N	NDB 7.2.0
	[none]		
	...		
OverloadLimit [2103]	bytes	N	NDB 7.2.0
	0		
	0 / 4G		
PortNumber	unsigned	N	NDB 7.2.0
	[none]		
	0 / 64K		

Parameter Name	Type or Units	Restart Type	As of Version ... And Later
	Default Value		
	Minimum/Maximum or Permitted Values		
<code>SendLimit</code> [2102]	unsigned	N	NDB 7.2.0
	8K		
	128 / 32K		
<code>SendSignalId</code> [2103]	boolean	N	NDB 7.2.0
	true		
	true, false		
<code>SharedBufferSize</code> [2102]	unsigned	N	NDB 7.2.0
	10M		
	64K / 4G		

17.3.4. MySQL Server Options and Variables for MySQL Cluster

This section provides information about MySQL server options, server and status variables that are specific to MySQL Cluster. For general information on using these, and for other options and variables not specific to MySQL Cluster, see [Section 5.1, “The MySQL Server”](#).

For MySQL Cluster configuration parameters used in the cluster configuration file (usually named `config.ini`), see [Section 17.3, “MySQL Cluster Configuration”](#).

17.3.4.1. MySQL Cluster `mysqld` Option and Variable Reference

The following table provides a list of the command-line options, server and status variables applicable within `mysqld` when it is running as an SQL node in a MySQL Cluster. For a table showing *all* command-line options, server and status variables available for use with `mysqld`, see [Section 5.1.1, “Server Option and Variable Reference”](#).

Table 17.9. MySQL Cluster `mysqld` Options and Variables: MySQL 5.5

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
<code>Handler_discover</code> [2161]	N	N	Both
	N	Y	N
The number of times that tables have been discovered			
<code>Ndb_api_wait_exec_complete_count_session</code> [2161]	N	N	Session
	N	Y	N
Number of times thread has been blocked while waiting for execution of an operation to complete in this client session.			
<code>Ndb_api_wait_exec_complete_count_slave</code> [2161]	N	N	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
	N	Y	N
Number of times thread has been blocked while waiting for execution of an operation to complete by this slave.			
Ndb_api_wait_exec_complete_count [2162]	N	N	Global
	N	Y	N
Number of times thread has been blocked while waiting for execution of an operation to complete by this MySQL Server (SQL node).			
Ndb_api_wait_scan_result_count_session [2162]	N	N	Session
	N	Y	N
Number of times thread has been blocked while waiting for a scan-based signal in this client session.			
Ndb_api_wait_scan_result_count_slave [2162]	N	N	Global
	N	Y	N
Number of times thread has been blocked while waiting for a scan-based signal by this slave.			
Ndb_api_wait_scan_result_count [2162]	N	N	Global
	N	Y	N
Number of times thread has been blocked while waiting for a scan-based signal by this MySQL Server (SQL node).			
Ndb_api_wait_meta_request_count_session [2162]	N	N	Session
	N	Y	N
Number of times thread has been blocked waiting for a metadata-based signal in this client session.			
Ndb_api_wait_meta_request_count_slave [2162]	N	N	Global
	N	Y	N
Number of times thread has been blocked waiting for a metadata-based signal by this slave.			
Ndb_api_wait_meta_request_count [2163]	N	N	Global
	N	Y	N
Number of times thread has been blocked waiting for a metadata-based signal by this MySQL Server (SQL node).			
Ndb_api_wait_nanos_count_session [2163]	N	N	Session
	N	Y	N
Total time (in nanoseconds) spent waiting for some type of signal from the data nodes in this client session.			
Ndb_api_wait_nanos_count_slave [2163]	N	N	Global
	N	Y	N
Total time (in nanoseconds) spent waiting for some type of signal from the data nodes by this slave.			
Ndb_api_wait_nanos_count [2163]	N	N	Global
	N	Y	N

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
Total time (in nanoseconds) spent waiting for some type of signal from the data nodes by this MySQL Server (SQL node).			
Ndb_api_bytes_sent_count_session [2163]	N	N	Session
	N	Y	N
Amount of data (in bytes) sent to the data nodes in this client session.			
Ndb_api_bytes_sent_count_slave [2164]	N	N	Global
	N	Y	N
Amount of data (in bytes) sent to the data nodes by this slave.			
Ndb_api_bytes_sent_count [2164]	N	N	Global
	N	Y	N
Amount of data (in bytes) sent to the data nodes by this MySQL Server (SQL node).			
Ndb_api_bytes_received_count_session [2164]	N	N	Session
	N	Y	N
Amount of data (in bytes) received from the data nodes in this client session.			
Ndb_api_bytes_received_count_slave [2164]	N	N	Global
	N	Y	N
Amount of data (in bytes) received from the data nodes by this slave.			
Ndb_api_bytes_received_count [2164]	N	N	Global
	N	Y	N
Amount of data (in bytes) received from the data nodes by this MySQL Server (SQL node).			
Ndb_api_trans_start_count_session [2164]	N	N	Session
	N	Y	N
Number of transactions started in this client session.			
Ndb_api_trans_start_count_slave [2165]	N	N	Global
	N	Y	N
Number of transactions started by this slave.			
Ndb_api_trans_start_count [2165]	N	N	Global
	N	Y	N
Number of transactions started by this MySQL Server (SQL node).			
Ndb_api_trans_commit_count_session [2165]	N	N	Session
	N	Y	N
Number of transactions committed in this client session.			
Ndb_api_trans_commit_count_slave [2165]	N	N	Global
	N	Y	N
Number of transactions committed by this slave.			

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
Ndb_api_trans_commit_count [2165]	N	N	Global
	N	Y	N
Number of transactions committed by this MySQL Server (SQL node).			
Ndb_api_trans_abort_count_session [2165]	N	N	Session
	N	Y	N
Number of transactions aborted in this client session.			
Ndb_api_trans_abort_count_slave [2165]	N	N	Global
	N	Y	N
Number of transactions aborted by this slave.			
Ndb_api_trans_abort_count [2166]	N	N	Global
	N	Y	N
Number of transactions aborted by this MySQL Server (SQL node).			
Ndb_api_trans_close_count_session [2166]	N	N	Session
	N	Y	N
Number of transactions aborted (may be greater than the sum of TransCommitCount and TransAbortCount) in this client session.			
Ndb_api_trans_close_count_slave [2166]	N	N	Global
	N	Y	N
Number of transactions aborted (may be greater than the sum of TransCommitCount and TransAbortCount) by this slave.			
Ndb_api_trans_close_count [2166]	N	N	Global
	N	Y	N
Number of transactions aborted (may be greater than the sum of TransCommitCount and TransAbortCount) by this MySQL Server (SQL node).			
Ndb_api_pk_op_count_session [2166]	N	N	Session
	N	Y	N
Number of operations based on or using primary keys in this client session.			
Ndb_api_pk_op_count_slave [2167]	N	N	Global
	N	Y	N
Number of operations based on or using primary keys by this slave.			
Ndb_api_pk_op_count [2167]	N	N	Global
	N	Y	N
Number of operations based on or using primary keys by this MySQL Server (SQL node).			
Ndb_api_uk_op_count_session [2167]	N	N	Session
	N	Y	N

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
Number of operations based on or using unique keys in this client session.			
Ndb_api_uk_op_count_slave [2167]	N	N	Global
	N	Y	N
Number of operations based on or using unique keys by this slave.			
Ndb_api_uk_op_count [2167]	N	N	Global
	N	Y	N
Number of operations based on or using unique keys by this MySQL Server (SQL node).			
Ndb_api_table_scan_count_session [2167]	N	N	Session
	N	Y	N
Number of table scans that have been started, including scans of internal tables, in this client session.			
Ndb_api_table_scan_count_slave [2168]	N	N	Global
	N	Y	N
Number of table scans that have been started, including scans of internal tables, by this slave.			
Ndb_api_table_scan_count [2168]	N	N	Global
	N	Y	N
Number of table scans that have been started, including scans of internal tables, by this MySQL Server (SQL node).			
Ndb_api_range_scan_count_session [2168]	N	N	Session
	N	Y	N
Number of range scans that have been started in this client session.			
Ndb_api_range_scan_count_slave [2168]	N	N	Global
	N	Y	N
Number of range scans that have been started by this slave.			
Ndb_api_range_scan_count [2168]	N	N	Global
	N	Y	N
Number of range scans that have been started by this MySQL Server (SQL node).			
Ndb_api_pruned_scan_count_session [2168]	N	N	Session
	N	Y	N
Number of scans that have been pruned to a single partition in this client session.			
Ndb_api_pruned_scan_count_slave [2169]	N	N	Global
	N	Y	N
Number of scans that have been pruned to a single partition by this slave.			
Ndb_api_pruned_scan_count [2169]	N	N	Global
	N	Y	N
Number of scans that have been pruned to a single partition by this MySQL Server (SQL node).			

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
Ndb_api_scan_batch_count_session [2169]	N	N	Session
	N	Y	N
Number of batches of rows received in this client session.			
Ndb_api_scan_batch_count_slave [2169]	N	N	Global
	N	Y	N
Number of batches of rows received by this slave.			
Ndb_api_scan_batch_count [2169]	N	N	Global
	N	Y	N
Number of batches of rows received by this MySQL Server (SQL node).			
Ndb_api_read_row_count_session [2169]	N	N	Session
	N	Y	N
Total number of rows that have been read in this client session.			
Ndb_api_read_row_count_slave [2170]	N	N	Global
	N	Y	N
Total number of rows that have been read by this slave.			
Ndb_api_read_row_count [2170]	N	N	Global
	N	Y	N
Total number of rows that have been read by this MySQL Server (SQL node).			
Ndb_api_trans_local_read_row_count_session [2170]	N	N	Session
	N	Y	N
Total number of rows that have been read in this client session.			
Ndb_api_trans_local_read_row_count_slave [2170]	N	N	Global
	N	Y	N
Total number of rows that have been read by this slave.			
Ndb_api_trans_local_read_row_count [2170]	N	N	Global
	N	Y	N
Total number of rows that have been read by this MySQL Server (SQL node).			
Ndb_api_event_data_count_injector [2171]	N	N	Global
	N	Y	N
Number of row change events received by the NDB binlog injector thread.			
Ndb_api_event_data_count [2171]	N	N	Global
	N	Y	N
Number of row change events received by this MySQL Server (SQL node).			
Ndb_api_event_nondata_count_injector [2171]	N	N	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
	N	Y	N
Number of events received, other than row change events, by the NDB binlog injector thread.			
Ndb_api_event_nondata_count [2171]	N	N	Global
	N	Y	N
Number of events received, other than row change events, by this MySQL Server (SQL node).			
Ndb_api_event_bytes_count_injector [2171]	N	N	Global
	N	Y	N
Number of bytes of events received by the NDB binlog injector thread.			
Ndb_api_event_bytes_count [2171]	N	N	Global
	N	Y	N
Number of bytes of events received by this MySQL Server (SQL node).			
Ndb_number_of_data_nodes [2173]	N	N	Global
	N	Y	N
If the server is part of a MySQL Cluster, the value of this variable is the number of data nodes in the cluster			
Ndb_conflict_fn_max [2172]	N	N	Global
	N	Y	N
If the server is part of a MySQL Cluster involved in cluster replication, the value of this variable indicates the number of times that conflict resolution based on "greater timestamp wins" has been applied			
Ndb_conflict_fn_old [2172]	N	N	Global
	N	Y	N
If the server is part of a MySQL Cluster involved in cluster replication, the value of this variable indicates the number of times that "same timestamp wins" conflict resolution has been applied			
Ndb_conflict_fn_epoch [2172]	N	N	Global
	N	Y	N
Number of rows that have been found in conflict by the NDB\$EPOCH() conflict detection function			
Ndb_conflict_fn_epoch_trans [2172]	N	N	Global
	N	Y	N
Number of rows that have been found in conflict by the NDB\$EPOCH_TRANS() conflict detection function			
Ndb_conflict_trans_row_conflict_count [2172]	N	N	Global
	N	Y	N
Number of rows found to be in conflict by a transactional conflict function			
Ndb_conflict_trans_row_reject_count [2173]	N	N	Global
	N	Y	N

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
Total number of rows realigned after being found in conflict by a transactional conflict function. Includes Ndb_conflict_trans_row_conflict_count and any rows included in or dependent on conflicting transactions.			
Ndb_conflict_trans_reject_count [2173]	N	N	Global
	N	Y	N
Number of transactions rejected after being found in conflict by a transactional conflict function.			
Ndb_conflict_trans_detect_iter_count [2173]	N	N	Global
	N	Y	N
The number of internal iterations required to commit an epoch transaction. Should be (slightly) greater than or equal to Ndb_conflict_trans_conflict_commit_count.			
Ndb_conflict_trans_conflict_commit_count [2173]	N	N	Global
	N	Y	N
The number of epoch transactions committed after requiring transactional conflict handling.			
ndb_optimization_delay [2147]	N	Y	Global
	N	N	Y
Sets the number of milliseconds to wait between processing sets of rows by OPTIMIZE TABLE on NDB tables.			
ndb_table_temporary [2157]	N	Y	Session
	N	N	Y
NDB tables are not persistent on disk: no schema files are created and the tables are not logged			
ndb_table_no_logging [2156]	N	Y	Session
	N	N	Y
NDB tables created when this setting is enabled are not checkpointed to disk (although table schema files are created). The setting in effect when the table is created with or altered to use NDBCLUSTER persists for the lifetime of the table.			
ndb_autoincrement_prefetch_sz [2148]	Y	Y	Both
	Y	N	Y
NDB auto-increment prefetch size			
ndb-batch-size [2138]	Y	Y	Global
	Y	N	N
The size (in bytes) to use for NDB transaction batches			
ndb-blob-read-batch-bytes [2140]	Y	Y	Both
	Y	N	Y
Specifies the size in bytes that large BLOB reads should be batched into. 0 = no limit.			
ndb-blob-write-batch-bytes [2140]	Y	Y	Both
	Y	N	Y
Specifies the size in bytes that large BLOB writes should be batched into. 0 = no limit.			

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
ndb_cache_check_time [2149]	Y	Y	Global
	Y	N	Y
The number of milliseconds between checks of cluster SQL nodes made by the MySQL query cache			
ndb-cluster-connection-pool [2139]	Y	Y	Global
	Y	Y	N
The number of connections to the cluster used by MySQL			
ndb_join_pushdown [2151]	N	Y	Global
	N	N	N
Enables pushing down of joins to data nodes			
Ndb_execute_count [2173]	N	N	Global
	N	Y	N
Provides the number of round trips to the NDB kernel made by operations			
Ndb_scan_count [2174]	N	N	Global
	N	Y	N
The total number of scans executed by NDB since the cluster was last started			
Ndb_pruned_scan_count [2174]	N	N	Global
	N	Y	N
The number of scans executed by NDB since the cluster was last started where partition pruning could be used			
ndb-connectstring [2141]	Y	N	
	Y	N	N
The management server that distributes the cluster configuration			
ndb_extra_logging [2150]	Y	Y	Global
	Y	N	Y
Controls logging of MySQL Cluster schema, connection, and data distribution events in the MySQL error log			
ndb_force_send [2150]	Y	Y	Both
	Y	N	Y
Forces sending of buffers to NDB immediately, without waiting for other threads			
ndb_use_exact_count [2157]	N	Y	Both
	N	N	Y
Use exact row count when planning queries			
ndb-log-apply-status [2142]	Y	Y	Global
	Y	N	N

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
Cause a MySQL server acting as a slave to log mysql.ndb_apply_status updates received from its immediate master in its own binary log, using its own server ID. Effective only if the server is started with the --ndbcluster option.			
ndb_log_bin [2153]	Y	Y	Both
	N	N	Y
Write updates to NDB tables in the binary log. Effective only if binary logging is enabled with --log-bin.			
ndb_log_binlog_index [2153]	Y	Y	Global
	N	N	Y
Insert mapping between epochs and binary log positions into the ndb_binlog_index table. Defaults to ON. Effective only if binary logging is enabled on the server.			
ndb_log_orig	N	Y	Global
	N	N	N
Log originating server id and epoch in mysql.ndb_binlog_index table.			
ndb-log-transaction-id [2143]	Y	Y	Global
	Y	N	N
Write NDB transaction IDs in the binary log. Requires --log-bin-v1-events=OFF.			
ndb_log_transaction_id [2154]	N	Y	Global
	N	N	N
Whether NDB transaction IDs are written into the binary log. (Read-only.)			
ndb_log_updated_only [2372]	Y	Y	Global
	Y	N	Y
Log complete rows (ON) or updates only (OFF)			
ndb-log-update-as-write [2373]	Y	Y	Global
	Y	N	Y
Toggles logging of updates on the master between updates (OFF) and writes (ON)			
ndb-log-empty-epochs [2144]	Y	Y	Global
	Y	N	Y
When enabled, causes epochs in which there were no changes to be written to the ndb_apply_status and ndb_binlog_index tables, even when --log-slave-updates is enabled.			
ndb_log_empty_epochs [2160]	Y	Y	Global
	Y	N	Y
When enabled, epochs in which there were no changes are written to the ndb_apply_status and ndb_binlog_index tables, even when log_slave_updates is enabled.			
ndb-transid-mysql-connection-map [2145]	Y	N	
	N	N	N

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
Enable or disable the <code>ndb_transid_mysql_connection_map</code> plugin; that is, enable or disable the <code>INFORMATION_SCHEMA</code> table having that name.			
<code>Ndb_pushed_reads</code> [2174]	N	N	Global
	N	Y	N
Number of reads executed on the data nodes by pushed-down joins			
<code>Ndb_pushed_queries_defined</code> [2173]	N	N	Global
	N	Y	N
Number of joins that API nodes have attempted to push down to the data nodes			
<code>Ndb_pushed_queries_dropped</code> [2173]	N	N	Global
	N	Y	N
Number of joins that API nodes have tried to push down, but failed			
<code>Ndb_pushed_queries_executed</code> [2174]	N	N	Global
	N	Y	N
Number of joins successfully pushed down and executed on the data nodes			
<code>ndb-nodeid</code> [2143]	Y	N	Global
	Y	Y	N
MySQL Cluster node ID for this MySQL server			
<code>ndb-mgmd-host</code> [2142]	Y	N	
	Y	N	N
Set the host (and port, if desired) for connecting to the management server			
<code>ndb-wait-connected</code> [2146]	Y	Y	Global
	Y	N	N
Time (in seconds) for the MySQL server to wait for connection to cluster management and data nodes before accepting MySQL client connections.			
<code>ndb-wait-setup</code> [2146]	Y	Y	Global
	Y	N	N
Time (in seconds) for the MySQL server to wait for NDB engine setup to complete.			
<code>ndbcluster</code> [2142]	Y	N	
	Y	N	N
Enable NDB Cluster (if this version of MySQL supports it)			
<code>ndbinfo_database</code> [2158]	N	Y	Global
	N	N	N
The name used for the NDB information database; read only.			
<code>ndbinfo_max_bytes</code> [2159]	Y	Y	Both
	N	N	Y

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
Used for debugging only.			
ndbinfo_max_rows [2159]	Y	Y	Both
	N	N	Y
Used for debugging only.			
ndbinfo_show_hidden [2159]	Y	Y	Both
	N	N	Y
Whether to show ndbinfo internal base tables in the mysql client. The default is OFF.			
ndbinfo_table_prefix [2160]	Y	Y	Both
	N	N	Y
The prefix to use for naming ndbinfo internal base tables. Defaults to 'ndb\$'.			
ndbinfo_version [2160]	N	Y	Global
	N	N	N
The version of the ndbinfo engine; read only.			
server-id-bits [2144]	Y	Y	Global
	Y	N	N
Sets the number of least significant bits in the server_id actually used for identifying the server, permitting NDB API applications to store application data in the most significant bits. server_id must be less than 2 to the power of this value.			
slave_allow_batching [2155]	Y	Y	Global
	Y	N	Y
Turns update batching on and off for a replication slave			
have_ndbcluster [2147]	N	Y	Global
	N	N	N
Whether mysqld supports NDB Cluster tables (set by --ndbcluster option)			
ndb-deferred-constraints [2141]	Y	Y	Both
	Y	N	Y
Specifies that constraint checks on unique indexes (where these are supported) should be deferred until commit time. Not normally needed or used; for testing purposes only.			
ndb_deferred_constraints [2149]	Y	Y	Both
	Y	N	Y
Specifies that constraint checks should be deferred (where these are supported). Not normally needed or used; for testing purposes only.			
ndb-distribution [2141]	Y	Y	Both
	Y	N	Y
Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH)			

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Notes			
<code>ndb_distribution</code> [2150]	Y	Y	Both
	Y	N	Y
Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH)			
<code>ndb_optimized_node_selection</code> [2154]	Y	N	
	Y	N	N
Determines how an SQL node chooses a cluster data node to use as transaction coordinator			
<code>ndb_report_thresh_binlog_epoch_slip</code> [2155]	Y	N	
	Y	N	N
This is a threshold on the number of epochs to be behind before reporting binlog status			
<code>ndb_report_thresh_binlog_mem_usage</code> [2155]	Y	N	
	Y	N	N
This is a threshold on the percentage of free memory remaining before reporting binlog status			
<code>ndb_use_transactions</code> [2157]	Y	Y	Both
	Y	N	Y
Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query			
<code>Ndb_cluster_node_id</code> [2171]	N	N	Both
	N	Y	N
If the server is acting as a MySQL Cluster node, then the value of this variable its node ID in the cluster			
<code>Ndb_config_from_host</code> [2172]	N	N	Both
	N	Y	N
The host name or IP address of the Cluster management server. Formerly <code>Ndb_connected_host</code>			
<code>Com_show_ndb_status</code> [597]	N	N	Both
	N	Y	N
Count of SHOW NDB STATUS statements			

17.3.4.2. MySQL Server Options for MySQL Cluster

This section provides descriptions of `mysqld` server options relating to MySQL Cluster. For information about `mysqld` options not specific to MySQL Cluster, and for general information about the use of options with `mysqld`, see [Section 5.1.3, “Server Command Options”](#).

For information about command-line options used with other MySQL Cluster processes (`ndbd`, `ndb_mgmd`, and `ndb_mgm`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#). For information about command-line options used with NDB utility programs (such as `ndb_desc`, `ndb_size.pl`, and `ndb_show_tables`), see [Section 17.4, “MySQL Cluster Programs”](#).

- `--ndb-batch-size=#`

Command-Line Format	<code>--ndb-batch-size</code>
Option-File Format	<code>ndb-batch-size</code>
Variable Name	<code>ndb_batch_size</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>numeric</code>
	Default <code>32768</code>
	Range <code>0 .. 31536000</code>

This sets the size in bytes that is used for NDB transaction batches.

- `--ndb-cluster-connection-pool=#`

Command-Line Format	<code>--ndb-cluster-connection-pool</code>
Option-File Format	<code>ndb-cluster-connection-pool</code>
Variable Name	<code>ndb_cluster_connection_pool</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>numeric</code>
	Default <code>1</code>
	Range <code>1 .. 63</code>

By setting this option to a value greater than 1 (the default), a `mysqld` process can use multiple connections to the cluster, effectively mimicking several SQL nodes. Each connection requires its own `[api]` or `[mysqld]` section in the cluster configuration (`config.ini`) file, and counts against the maximum number of API connections supported by the cluster.

Suppose that you have 2 cluster host computers, each running an SQL node whose `mysqld` process was started with `--ndb-cluster-connection-pool=4`; this means that the cluster must have 8 API slots available for these connections (instead of 2). All of these connections are set up when the SQL node connects to the cluster, and are allocated to threads in a round-robin fashion.

This option is useful only when running `mysqld` on host machines having multiple CPUs, multiple cores, or both. For best results, the value should be smaller than the total number of cores available on the host machine. Setting it to a value greater than this is likely to degrade performance severely.



Important

Because each SQL node using connection pooling occupies multiple API node slots—each slot having its own node ID in the cluster—you must *not* use a node ID as part of the cluster connectstring when starting any `mysqld` process that employs connection pooling.

Setting a node ID in the connectstring when using the `--ndb-cluster-connection-pool` option causes node ID allocation errors when the SQL node attempts to connect to the cluster.

**Note**

In some older releases of MySQL Cluster prior to MySQL Cluster NDB 7.2, there was also a separate status variable corresponding to this option; however, the status variable was removed as redundant as of these versions. (Bug #60119)

- `--ndb-blob-read-batch-bytes=bytes`

Command-Line Format	<code>--ndb-blob-read-batch-bytes</code>
Option-File Format	<code>ndb-blob-read-batch-bytes</code>
Variable Name	<code>ndb_blob_read_batch_bytes</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>65535</code>
	Range <code>0 .. 4294967295</code>

This option can be used to set the size (in bytes) for batching of **BLOB** data reads in MySQL Cluster applications. When this batch size is exceeded by the amount of **BLOB** data to be read within the current transaction, any pending **BLOB** read operations are immediately executed.

The maximum value for this option is 4294967295; the default is 65535. Setting it to 0 has the effect of disabling **BLOB** read batching.

**Note**

In NDB API applications, you can control **BLOB** write batching with the `setMaxPendingBlobReadBytes()` and `getMaxPendingBlobReadBytes()` methods.

- `--ndb-blob-write-batch-bytes=bytes`

Command-Line Format	<code>--ndb-blob-write-batch-bytes</code>
Option-File Format	<code>ndb-blob-write-batch-bytes</code>
Variable Name	<code>ndb_blob_write_batch_bytes</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>65535</code>
	Range <code>0 .. 4294967295</code>

This option can be used to set the size (in bytes) for batching of **BLOB** data writes in MySQL Cluster applications. When this batch size is exceeded by the amount of **BLOB** data to be written within the current transaction, any pending **BLOB** write operations are immediately executed.

The maximum value for this option is 4294967295; the default is 65535. Setting it to 0 has the effect of disabling `BLOB` write batching.



Note

In NDB API applications, you can control `BLOB` write batching with the `setMaxPendingBlobWriteBytes()` and `getMaxPendingBlobWriteBytes()` methods.

- `--ndb-connectstring=connect_string`

Command-Line Format	<code>--ndb-connectstring</code>	
Option-File Format	<code>ndb-connectstring</code>	
	Permitted Values	
	Type	<code>string</code>

When using the `NDBCLUSTER` storage engine, this option specifies the management server that distributes cluster configuration data. See [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#), for syntax.

- `--ndb-deferred-constraints=[TRUE|FALSE]`

Command-Line Format	<code>--ndb-deferred-constraints</code>	
Option-File Format	<code>ndb-deferred-constraints</code>	
Option Sets Variable	Yes, <code>ndb_deferred_constraints</code> [2149]	
Variable Name	<code>ndb-deferred-constraints</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>false</code>

Controls whether or not constraint checks on unique indexes are deferred until commit time, where such checks are supported. `OFF` is the default.

This option is not normally needed for operation of MySQL Cluster or MySQL Cluster Replication, and is intended primarily for use in testing.

- `--ndb-distribution=[KEYHASH|LINHASH]`

Command-Line Format	<code>--ndb-distribution={KEYHASH LINHASH}</code>	
Option-File Format	<code>ndb-distribution</code>	
Option Sets Variable	Yes, <code>ndb_distribution</code> [2150]	
Variable Name	<code>ndb-distribution</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	

	Type	enumeration
	Default	KEYHASH
	Valid Values	LINHASH KEYHASH

Controls the default distribution method for `NDB` tables. Can be set to either of `KEYHASH` (key hashing) or `LINHASH` (linear hashing). `KEYHASH` is the default.

- `--ndb-mgmd-host=host[:port]`

Command-Line Format	<code>--ndb-mgmd-host=host[:port]</code>	
Option-File Format	<code>ndb-mgmd-host</code>	
	Permitted Values	
	Type	string
	Default	localhost:1186

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` [2141] option instead.

- `--ndbcluster` [2142]

Command-Line Format	<code>--ndbcluster</code>	
Option-File Format	<code>ndbcluster</code>	
Option Sets Variable	Yes, <code>have_ndbcluster</code> [2147]	
Disabled by	<code>skip-ndbcluster</code>	
	Permitted Values	
	Type	boolean
	Default	FALSE

The `NDBCLUSTER` storage engine is necessary for using MySQL Cluster. If a `mysqld` binary includes support for the `NDBCLUSTER` storage engine, the engine is disabled by default. Use the `--ndbcluster` [2142] option to enable it. Use `--skip-ndbcluster` to explicitly disable the engine.

- `--ndb-log-apply-status`

Command-Line Format	<code>--ndb-log-apply-status</code>	
Option-File Format	<code>ndb-log-apply-status</code>	
Option Sets Variable	Yes, <code>ndb_log_apply_status</code> [2153]	
Variable Name	<code>ndb_log_apply_status</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

Causes a slave `mysqld` to log any updates received from its immediate master to the `mysql.ndb_apply_status` table in its own binary log using its own server ID rather than the server ID of the master. In a circular or chain replication setting, this allows such updates to propagate to the `mysql.ndb_apply_status` tables of any MySQL servers configured as slaves of the current `mysqld`.

In a chain replication setup, using this option allows downstream (slave) clusters to be aware of their positions relative to all of their upstream contributors (masters).

In a circular replication setup, this option causes changes to `ndb_apply_status` tables to complete the entire circuit, eventually propagating back to the originating MySQL Cluster. This also allows a cluster acting as a master to see when its changes (epochs) have been applied to the other clusters in the circle.

This option has no effect unless the MySQL server is started with the `--ndbcluster` [2142] option.

- `--ndb-log-transaction-id`

Version Introduced	5.5.15-ndb-7.2.1	
Command-Line Format	<code>--ndb-log-transaction-id[={0 1}]</code>	
Option-File Format	<code>ndb-log-transaction-id</code>	
Option Sets Variable	Yes, <code>ndb_log_transaction_id</code> [2154]	
Variable Name	<code>ndb_log_transaction_id</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

Causes a slave `mysqld` to write the NDB transaction ID in each row of the binary log. Such logging requires the use of the Version 2 event format for the binary log; thus, `--log-bin-use-v1-row-events` [1893] must be set to `FALSE` in order to use this option.

This option is available beginning with MySQL Cluster NDB 7.2.1 (and is not supported in mainline MySQL Server 5.5). It is required to enable MySQL Cluster Replication conflict detection and resolution using the `NDB$EPOCH_TRANS()` function introduced in the same MySQL Cluster release.

The default value is `FALSE`.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `--ndb-nodeid=#`

Command-Line Format	<code>--ndb-nodeid=#</code>
Option-File Format	<code>ndb-nodeid</code>
Variable Name	<code>Ndb_cluster_node_id</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	numeric
	Range	1 .. 255

Set this MySQL server's node ID in a MySQL Cluster.

The `--ndb-nodeid` option overrides any node ID set with `--ndb-connectstring` [2141], regardless of the order in which the two options are used.

In addition, if `--ndb-nodeid` is used, then either a matching node ID must be found in a `[mysqld]` or `[api]` section of `config.ini`, or there must be an “open” `[mysqld]` or `[api]` section in the file (that is, a section without a `NodeId` or `Id` parameter specified). This is also true if the node ID is specified as part of the connectstring.

Regardless of how the node ID is determined, its is shown as the value of the global status variable `Ndb_cluster_node_id` in the output of `SHOW STATUS`, and as `cluster_node_id` in the `connection` row of the output of `SHOW ENGINE NDBCLUSTER STATUS`.

For more information about node IDs for MySQL Cluster SQL nodes, see [Section 17.3.2.7, “Defining SQL and Other API Nodes in a MySQL Cluster”](#).

- `--ndb-log-empty-epochs=[0|1]`

Command-Line Format	<code>--ndb-log-empty-epochs</code>	
Option-File Format	<code>ndb-log-empty-epochs</code>	
Variable Name	<code>ndb_log_empty_epochs</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Causes epochs during which there were no changes to be written to the `ndb_apply_status` and `ndb_binlog_index` tables, even when `--log-slave-updates` is enabled.

By default this option is disabled. Disabling `--ndb-log-empty-epochs` causes epoch transactions with no changes not to be written to the binary log, although a row is still written even for an empty epoch in `ndb_binlog_index`.

Because `--ndb-log-empty-epochs=1` causes the size of `ndb_binlog_index` table to increase independently of the size of the binary log, users should be prepared to manage the growth of this table, even if they expect the cluster to be idle a large part of the time.

- `--server-id-bits=#`

Command-Line Format	<code>--server-id-bits=#</code>
Option-File Format	<code>server-id-bits</code>
Option Sets Variable	Yes, <code>server_id_bits</code> [2160]
Variable Name	<code>server_id_bits</code>
Variable Scope	Global
Dynamic Variable	No

	Permitted Values	
Type	numeric	
Default	32	
Range	7 .. 32	

This option indicates the number of least significant bits within the 32-bit `server_id` [551] which actually identify the server. Indicating that the server is actually identified by fewer than 32 bits makes it possible for some of the remaining bits to be used for other purposes, such as storing user data generated by applications using the NDB API's Event API within the `AnyValue` of an `OperationOptions` structure (MySQL Cluster uses the `AnyValue` to store the server ID).

When extracting the effective server ID from `server_id` [551] for purposes such as detection of replication loops, the server ignores the remaining bits. The `--server-id-bits` option is used to mask out any irrelevant bits of `server_id` [551] in the IO and SQL threads when deciding whether an event should be ignored based on the server ID.

This data can be read from the binary log by `mysqlbinlog`, provided that it is run with its own `--server-id-bits` [376] option set to 32 (the default).

The value of `server_id` [551] must be less than $2^{\text{server_id_bits}}$ [2160]; otherwise, `mysqld` refuses to start.

This system variable is supported only by MySQL Cluster. It is not supported in the standard MySQL 5.5 Server.

- `--skip-ndbcluster`

Command-Line Format	<code>--skip-ndbcluster</code>
Option-File Format	<code>skip-ndbcluster</code>

Disable the `NDBCLUSTER` storage engine. This is the default for binaries that were built with `NDBCLUSTER` storage engine support; the server allocates memory and other resources for this storage engine only if the `--ndbcluster` [2142] option is given explicitly. See [Section 17.3.1, "Quick Test Setup of MySQL Cluster"](#), for an example.

- `ndb-transid-mysql-connection-map=state`

Version Introduced	5.5.16-ndb-7.2.2	
Command-Line Format	<code>--ndb-transid-mysql-connection-map[=state]</code>	5.1.16-ndb-7.1.17
	Permitted Values	
Type	enumeration	
Default	ON	
Valid Values	OFF FORCE	

Enables or disables the plugin that handles the `ndb_transid_mysql_connection_map` table in the `INFORMATION_SCHEMA` database. Takes one of the values `ON`, `OFF`, or `FORCE`. `ON` (the default) enables the plugin. `OFF` disables the plugin, which makes `ndb_transid_mysql_connection_map` inaccessible. `FORCE` keeps the MySQL Server from starting if the plugin fails to load and start.

You can see whether the `ndb_transid_mysql_connection_map` table plugin is running by checking the output of `SHOW PLUGINS`.

This option was added in MySQL Cluster NDB 7.2.2.

- `--ndb-wait-connected=seconds`

Command-Line Format	<code>--ndb-wait-connected=#</code>	
Option-File Format	<code>ndb-wait-connected</code>	
Variable Name	<code>ndb-wait-connected</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 31536000</code>
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. 31536000</code>
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>30</code>
	Range	<code>0 .. 31536000</code>
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>30</code>
	Range	<code>0 .. 31536000</code>

This option sets the period of time that the MySQL server waits for connections to MySQL Cluster management and data nodes to be established before accepting MySQL client connections. The time is specified in seconds. The default value is `30`.

- `--ndb-wait-setup=seconds`

Command-Line Format	<code>--ndb-wait-setup=#</code>
Option-File Format	<code>ndb-wait-setup</code>
Variable Name	<code>ndb-wait-setup</code>
Variable Scope	Global

Dynamic Variable	No
	Permitted Values
Type	numeric
Default	15
Range	0 .. 31536000
	Permitted Values
Type	numeric
Default	15
Range	0 .. 31536000
	Permitted Values
Type	numeric
Default	30
Range	0 .. 31536000
	Permitted Values
Type	numeric
Default	30
Range	0 .. 31536000

This variable shows the period of time that the MySQL server waits for the [NDB](#) storage engine to complete setup before timing out and treating [NDB](#) as unavailable. The time is specified in seconds. The default value is 30.

- `--ndb_optimization_delay=milliseconds`

Variable Name	<code>ndb_optimization_delay</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
Type	numeric
Default	10
Range	0 .. 100000

Set the number of milliseconds to wait between sets of rows by `OPTIMIZE TABLE` statements on [NDB](#) tables. The default is 15.

17.3.4.3. MySQL Cluster System Variables

This section provides detailed information about MySQL server system variables that are specific to MySQL Cluster and the [NDB](#) storage engine. For system variables not specific to MySQL Cluster, see [Section 5.1.4, “Server System Variables”](#). For general information on using system variables, see [Section 5.1.5, “Using System Variables”](#).

- `have_ndbcluster` [\[2147\]](#)

Variable Name	<code>have_ndbcluster</code>
Variable Scope	Global

Dynamic Variable	No	
	Permitted Values	
	Type	boolean

YES if `mysqld` supports `NDBCLUSTER` tables. `DISABLED` if `--skip-ndbcluster` is used.

This variable is deprecated and is removed in MySQL 5.6. Use `SHOW ENGINES` instead.

- `multi_range_count`

Command-Line Format	<code>--multi_range_count=#</code>	
Option-File Format	<code>multi_range_count</code>	
Option Sets Variable	Yes, <code>multi_range_count</code> [2148]	
Variable Name	<code>multi_range_count</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	256
	Range	1 .. 4294967295

The maximum number of ranges to send to a table handler at once during range selects. The default value is 256. Sending multiple ranges to a handler at once can improve the performance of certain selects dramatically. This is especially true for the `NDBCLUSTER` table handler, which needs to send the range requests to all nodes. Sending a batch of those requests at once reduces communication costs significantly.

This variable is deprecated in MySQL 5.1, and is no longer supported in MySQL 5.5, in which arbitrarily long lists of ranges can be processed.

- `ndb_autoincrement_prefetch_sz` [2148]

Command-Line Format	<code>--ndb_autoincrement_prefetch_sz</code>	
Option-File Format	<code>ndb_autoincrement_prefetch_sz</code>	
Option Sets Variable	Yes, <code>ndb_autoincrement_prefetch_sz</code> [2148]	
Variable Name	<code>ndb_autoincrement_prefetch_sz</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1
	Range	1 .. 65536

Determines the probability of gaps in an autoincremented column. Set it to 1 to minimize this. Setting it to a high value for optimization makes inserts faster, but decreases the likelihood that consecutive autoincrement numbers will be used in a batch of inserts. The minimum and default value is 1. The maximum value for `ndb_autoincrement_prefetch_sz` is 65536.

This variable affects only the number of `AUTO_INCREMENT` IDs that are fetched between statements; within a given statement, at least 32 IDs are obtained at a time. The default value for `ndb_autoincrement_prefetch_sz` [2148] is 1, to increase the speed of statements inserting single rows.



Important

This variable does not affect inserts performed using `INSERT ... SELECT`.

- `ndb_cache_check_time` [2149]

Command-Line Format	<code>--ndb_cache_check_time</code>
Option-File Format	<code>ndb_cache_check_time</code>
Option Sets Variable	Yes, <code>ndb_cache_check_time</code> [2149]
Variable Name	<code>ndb_cache_check_time</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>numeric</code>
	Default <code>0</code>

The number of milliseconds that elapse between checks of MySQL Cluster SQL nodes by the MySQL query cache. Setting this to 0 (the default and minimum value) means that the query cache checks for validation on every query.

The recommended maximum value for this variable is 1000, which means that the check is performed once per second. A larger value means that the check is performed and possibly invalidated due to updates on different SQL nodes less often. It is generally not desirable to set this to a value greater than 2000.

- `ndb_deferred_constraints` [2149]

Command-Line Format	<code>--ndb-deferred-constraints</code>
Option-File Format	<code>ndb_deferred_constraints</code>
Option Sets Variable	Yes, <code>ndb_deferred_constraints</code> [2149]
Variable Name	<code>ndb_deferred_constraints</code>
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values
	Type <code>boolean</code>
	Default <code>false</code>
	Range <code>false .. true</code>

Controls whether or not constraint checks are deferred, where these are supported. `OFF` is the default.

This variable is not normally needed for operation of MySQL Cluster or MySQL Cluster Replication, and is intended primarily for use in testing.

- [ndb_distribution \[2150\]](#)

Command-Line Format	<code>--ndb-distribution={KEYHASH LINHASH}</code>	
Option-File Format	<code>ndb_distribution</code>	
Variable Name	<code>ndb_distribution={KEYHASH LINHASH}</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	KEYHASH
	Valid Values	LINHASH KEYHASH

Controls the default distribution method for **NDB** tables. Can be set to either of **KEYHASH** (key hashing) or **LINHASH** (linear hashing). **KEYHASH** is the default.

- [ndb_extra_logging \[2150\]](#)

Command-Line Format	<code>ndb_extra_logging=#</code>	
Option-File Format	<code>ndb_extra_logging</code>	
Variable Name	<code>ndb_extra_logging</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0

This variable can be used to enable recording in the MySQL error log of information specific to the **NDB** storage engine. It is normally of interest only when debugging **NDB** storage engine code.

The default value is 0, which means that the only **NDB**-specific information written to the MySQL error log relates to transaction handling. If the value is greater than 0 but less than 10, **NDB** table schema and connection events are also logged, as well as whether or not conflict resolution is in use, and other **NDB** errors and information. If the value is set to 10 or more, information about **NDB** internals, such as the progress of data distribution among cluster nodes, is also written to the MySQL error log.

- [ndb_force_send \[2150\]](#)

Command-Line Format	<code>--ndb-force-send</code>	
Option-File Format	<code>ndb_force_send</code>	
Option Sets Variable	Yes, ndb_force_send [2150]	
Variable Name	<code>ndb_force_send</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	

	Type	boolean
	Default	TRUE

Forces sending of buffers to [NDB](#) immediately, without waiting for other threads. Defaults to [ON](#).

- [ndb_join_pushdown \[2151\]](#)

Variable Name	ndb_join_pushdown	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	TRUE

Added in MySQL Cluster NDB 7.2.0, this variable controls whether joins on [NDB](#) tables are pushed down to the NDB kernel (data nodes). Previously, a join was handled using multiple accesses of [NDB](#) by the SQL node; however, when [ndb_join_pushdown \[2151\]](#) is enabled, a pushable join is sent in its entirety to the data nodes, where it can be distributed among the data nodes and executed in parallel on multiple copies of the data, with a single, merged result being returned to `mysqld`. This can reduce greatly the number of round trips between an SQL node and the data nodes required to handle such a join.

By default, [ndb_join_pushdown \[2151\]](#) is enabled.

In order for a join to be pushable, it must meet the following conditions:

1. Only columns can be compared, and all columns to be joined must use *exactly* the same data type.

This means that expressions such as `t1.a = t2.a + constant` cannot be pushed down, and that (for example) a join on an [INT](#) column and a [BIGINT](#) column also cannot be pushed down.
2. Queries referencing [BLOB](#) or [TEXT](#) columns are not supported.
3. Explicit locking is not supported; however, the [NDB](#) storage engine's characteristic implicit row-based locking is enforced.

This means that a join using `FOR UPDATE` cannot be pushed down.

4. In order for a join to be pushed down, child tables in the join must be accessed using one of the [ref \[837\]](#), [eq_ref \[837\]](#), or [const \[837\]](#) access methods, or some combination of these methods.

Outer joined child tables can only be pushed using [eq_ref \[837\]](#).

If the root of the pushed join is an [eq_ref \[837\]](#) or [const \[837\]](#), only child tables joined by [eq_ref \[837\]](#) can be appended. (A table joined by [ref \[837\]](#) is likely to become the root of another pushed join.)

If the query optimizer decides on `Using join cache` for a candidate child table, that table cannot be pushed as a child. However, it may be the root of another set of pushed tables.

5. Joins referencing tables explicitly partitioned by [[LINEAR](#)] [HASH](#), [LIST](#), or [RANGE](#) currently cannot be pushed down.

You can see whether a given join can be pushed down by checking it with `EXPLAIN`; when the join can be pushed down, you can see references to the `pushed join` in the `Extra` column of the output, as shown in this example:

```
mysql> EXPLAIN
-> SELECT e.first_name, e.last_name, t.title, d.dept_name
-> FROM employees e
-> JOIN dept_emp de ON e.emp_no=de.emp_no
-> JOIN departments d ON d.dept_no=de.dept_no
-> JOIN titles t ON e.emp_no=t.emp_no\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: d
         type: ALL
possible_keys: PRIMARY
         key: NULL
        key_len: NULL
         ref: NULL
         rows: 9
      Extra: Parent of 4 pushed join@1
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: de
         type: ref
possible_keys: PRIMARY,emp_no,dept_no
         key: dept_no
        key_len: 4
         ref: employees.d.dept_no
         rows: 5305
      Extra: Child of 'd' in pushed join@1
***** 3. row *****
      id: 1
  select_type: SIMPLE
        table: e
         type: eq_ref
possible_keys: PRIMARY
         key: PRIMARY
        key_len: 4
         ref: employees.de.emp_no
         rows: 1
      Extra: Child of 'de' in pushed join@1
***** 4. row *****
      id: 1
  select_type: SIMPLE
        table: t
         type: ref
possible_keys: PRIMARY,emp_no
         key: emp_no
        key_len: 4
         ref: employees.de.emp_no
         rows: 19
      Extra: Child of 'e' in pushed join@1
4 rows in set (0.00 sec)
```



Note

If inner joined child tables are joined by `ref` [837], and the result is ordered or grouped by a sorted index, this index cannot provide sorted rows, which forces writing to a sorted tempfile.

Two additional sources of information about pushed join performance are available:

1. The status variables `Ndb_pushed_queries_defined` [2173], `Ndb_pushed_queries_dropped` [2173], `Ndb_pushed_queries_executed` [2174], and `Ndb_pushed_reads` [2174] (all introduced in MySQL Cluster NDB 7.2.0).
 2. The counters in the `ndbinfo.counters` table that belong to the `DBSPJ` kernel block. (These counters and the `DBSPJ` block were also introduced in MySQL Cluster NDB 7.2.0). See [Section 17.5.10.7, “The ndbinfo counters Table”](#), for information about these counters. See also [The DBSPJ Block](#), in the *MySQL Cluster API Developer Guide*.
- `ndb_log_apply_status` [2153]

Command-Line Format	<code>--ndb-log-apply-status</code>	
Option-File Format	<code>ndb-log-apply-status</code>	
Option Sets Variable	Yes, <code>ndb_log_apply_status</code> [2153]	
Variable Name	<code>ndb_log_apply_status</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

A read-only variable which shows whether the server was started with the `--ndb-log-apply-status` [2142] option.

- `ndb_log_bin` [2153]

Command-Line Format	<code>--ndb-log-bin={1 0}</code>	
Option Sets Variable	Yes, <code>ndb_log_bin</code> [2153]	
Variable Name	<code>ndb_log_bin</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>ON</code>

Causes updates to `NDB` tables to be written to the binary log. Setting this variable has no effect if binary logging is not already enabled for the server using `log_bin` [514]. `ndb_log_bin` defaults to 1 (ON); normally, there is never any need to change this value in a production environment.

- `ndb_log_binlog_index` [2153]

Command-Line Format	<code>--ndb-log-binlog-index={1 0}</code>	
Option Sets Variable	Yes, <code>ndb_log_binlog_index</code> [2153]	
Variable Name	<code>ndb_log_binlog_index</code>	
Variable Scope	Global	
Dynamic Variable	Yes	

	Permitted Values	
Type	boolean	
Default	ON	

Causes a mapping of epochs to positions in the binary log to be inserted into the `ndb_binlog_index` table. Setting this variable has no effect if binary logging is not already enabled for the server using `log_bin` [514]. (In addition, `ndb_log_bin` [2153] must not be disabled.) `ndb_log_binlog_index` defaults to 1 (ON); normally, there is never any need to change this value in a production environment.

- `ndb_log_transaction_id` [2154]

Version Introduced	5.5.15-ndb-7.2.1	
Variable Name	<code>ndb_log_transaction_id</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
Type	boolean	
Default	OFF	

This read-only, Boolean system variable shows whether a slave `mysqld` writes NDB transaction IDs in the binary log (required to use “active-active” MySQL Cluster Replication with `NDB$EPOCH_TRANS()` conflict detection). To change the setting, use the `--ndb-log-transaction-id` [2143] option.

`ndb_log_transaction_id` [2154] is available in MySQL Cluster NDB 7.2.1 and later. It is not supported in mainline MySQL Server 5.5.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `ndb_optimized_node_selection` [2154]

Command-Line Format	<code>--ndb-optimized-node-selection=#</code>	
Option-File Format	<code>ndb_optimized_node_selection</code>	
	Permitted Values	
Type	boolean	
Default	ON	
	Permitted Values	
Type	numeric	
Default	3	
Range	0 .. 3	

There are two forms of optimized node selection, described here:

1. The SQL node uses *proximity* to determine the transaction coordinator; that is, the “closest” data node to the SQL node is chosen as the transaction coordinator. For this purpose, a data node having a shared memory connection with the SQL node is considered to be “closest” to the SQL node; the next closest (in order of decreasing proximity) are: TCP connection to `localhost`; SCI connection; TCP connection from a host other than `localhost`.

2. The SQL thread uses *distribution awareness* to select the data node. That is, the data node housing the cluster partition accessed by the first statement of a given transaction is used as the transaction coordinator for the entire transaction. (This is effective only if the first statement of the transaction accesses no more than one cluster partition.)

This option takes one of the integer values 0, 1, 2, or 3. 3 is the default. These values affect node selection as follows:

- 0: Node selection is not optimized. Each data node is employed as the transaction coordinator 8 times before the SQL thread proceeds to the next data node.
 - 1: Proximity to the SQL node is used to determine the transaction coordinator.
 - 2: Distribution awareness is used to select the transaction coordinator. However, if the first statement of the transaction accesses more than one cluster partition, the SQL node reverts to the round-robin behavior seen when this option is set to 0.
 - 3: If distribution awareness can be employed to determine the transaction coordinator, then it is used; otherwise proximity is used to select the transaction coordinator. (This is the default behavior.)
- [ndb_report_thresh_binlog_epoch_slip \[2155\]](#)

Command-Line Format	<code>--ndb_report_thresh_binlog_epoch_slip</code>	
Option-File Format	<code>ndb_report_thresh_binlog_epoch_slip</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>3</code>
	Range	<code>0 .. 256</code>

This is a threshold on the number of epochs to be behind before reporting binary log status. For example, a value of 3 (the default) means that if the difference between which epoch has been received from the storage nodes and which epoch has been applied to the binary log is 3 or more, a status message will be sent to the cluster log.

- [ndb_report_thresh_binlog_mem_usage \[2155\]](#)

Command-Line Format	<code>--ndb_report_thresh_binlog_mem_usage</code>	
Option-File Format	<code>ndb_report_thresh_binlog_mem_usage</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>10</code>
	Range	<code>0 .. 10</code>

This is a threshold on the percentage of free memory remaining before reporting binary log status. For example, a value of 10 (the default) means that if the amount of available memory for receiving binary log data from the data nodes falls below 10%, a status message will be sent to the cluster log.

- [slave_allow_batching \[2155\]](#)

Command-Line Format	<code>--slave-allow-batching</code>
----------------------------	-------------------------------------

Option-File Format	slave_allow_batching	
Option Sets Variable	Yes, slave_allow_batching [2155]	
Variable Name	slave_allow_batching	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	off

Whether or not batched updates are enabled on MySQL Cluster replication slaves.

Currently, this variable is available for `mysqld` only as supplied with MySQL Cluster or built from the MySQL Cluster sources. For more information, see [Section 17.6.6, “Starting MySQL Cluster Replication \(Single Replication Channel\)”](#).

Setting this variable had no effect in MySQL Cluster NDB 7.2 prior to MySQL Cluster NDB 7.2.10. (Bug #15953530)

- [ndb_table_no_logging \[2156\]](#)

Variable Name	ndb_table_no_logging	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	FALSE

When this variable is set to `ON` or `1`, it causes `NDB` tables not to be checkpointed to disk. More specifically, this setting applies to tables which are created or altered using `ENGINE NDB` when [ndb_table_no_logging \[2156\]](#) is enabled, and continues to apply for the lifetime of the table, even if [ndb_table_no_logging \[2156\]](#) is later changed. Suppose that `A`, `B`, `C`, and `D` are tables that we create (and perhaps also alter), and that we also change the setting for [ndb_table_no_logging \[2156\]](#) as shown here:

```
SET @@ndb_table_no_logging = 1;

CREATE TABLE A ... ENGINE NDB;

CREATE TABLE B ... ENGINE MYISAM;
CREATE TABLE C ... ENGINE MYISAM;

ALTER TABLE B ENGINE NDB;

SET @@ndb_table_no_logging = 0;

CREATE TABLE D ... ENGINE NDB;
ALTER TABLE C ENGINE NDB;

SET @@ndb_table_no_logging = 1;
```

After the previous sequence of events, tables `A` and `B` are not checkpointed; `A` was created with `ENGINE NDB` and `B` was altered to use `NDB`, both while [ndb_table_no_logging](#) was enabled. However, tables `C` and `D` are logged; `C` was altered to use `NDB` and `D` was created using `ENGINE NDB`, both while

`ndb_table_no_logging` [2156] was disabled. Setting `ndb_table_no_logging` [2156] back to 1 or ON does *not* cause table C or D to be checkpointed.



Note

`ndb_table_no_logging` [2156] has no effect on the creation of NDB table schema files; to suppress these, use `ndb_table_temporary` [2157] instead.

- `ndb_table_temporary` [2157]

Variable Name	<code>ndb_table_temporary</code>	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

When set to ON or 1, this variable causes NDB tables not to be written to disk: This means that no table schema files are created, and that the tables are not logged.



Note

Setting this variable currently has no effect in MySQL Cluster NDB 7.0 and later. This is a known issue; see BUG#34036.

- `ndb_use_copying_alter_table` [2157]

Variable Name	<code>ndb_use_copying_alter_table</code>
Variable Scope	Global, Session
Dynamic Variable	No

Forces NDB to use copying of tables in the event of problems with online ALTER TABLE operations. The default value is OFF.

- `ndb_use_exact_count` [2157]

Variable Name	<code>ndb_use_exact_count</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>ON</code>

Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query. The default value is ON. For faster queries overall, disable this feature by setting the value of `ndb_use_exact_count` [2157] to OFF.

- `ndb_use_transactions` [2157]

Command-Line Format	<code>--ndb_use_transactions</code>
----------------------------	-------------------------------------

Option-File Format	ndb_use_transactions	
Variable Name	ndb_use_transactions	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

You can disable **NDB** transaction support by setting this variable's values to **OFF** (not recommended). The default is **ON**.

- [transaction_allow_batching \[2158\]](#)

Variable Name	transaction_allow_batching	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	FALSE

When set to **1** or **ON**, this variable enables batching of statements within the same transaction. To use this variable, [autocommit \[484\]](#) must first be disabled by setting it to **0** or **OFF**; otherwise, setting [transaction_allow_batching \[2158\]](#) has no effect.

It is safe to use this variable with transactions that performs writes only, as having it enabled can lead to reads from the “before” image. You should ensure that any pending transactions are committed (using an explicit **COMMIT** if desired) before issuing a **SELECT**.



Important

[transaction_allow_batching \[2158\]](#) should not be used whenever there is the possibility that the effects of a given statement depend on the outcome of a previous statement within the same transaction.

This variable is currently supported for MySQL Cluster only.



Important

Due an issue in the MySQL Cluster NDB 7.2 codebase (Bug #64697) prior to General Availability, this variable is not available prior to MySQL Cluster NDB 7.2.6.

The system variables in the following list all relate to the [ndbinfo](#) information database.

- [ndbinfo_database \[2158\]](#)

Variable Name	ndbinfo_database	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	

	Type	string
	Default	ndbinfo

Shows the name used for the NDB information database; the default is `ndbinfo`. This is a read-only variable whose value is determined at compile time; you can set it by starting the server using `--ndbinfo-database=name`, which sets the value shown for this variable but does not actually change the name used for the NDB information database.

- [ndbinfo_max_bytes \[2159\]](#)

Command-Line Format	<code>--ndbinfo-max-bytes=#</code>	
Option Sets Variable	Yes, ndbinfo_max_bytes [2159]	
Variable Name	<code>ndbinfo_max_bytes</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0

Used in testing and debugging only.

- [ndbinfo_max_rows \[2159\]](#)

Command-Line Format	<code>--ndbinfo-max-rows=#</code>	
Option Sets Variable	Yes, ndbinfo_max_rows [2159]	
Variable Name	<code>ndbinfo_max_rows</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	10

Used in testing and debugging only.

- [ndbinfo_show_hidden \[2159\]](#)

Command-Line Format	<code>--ndbinfo-show-hidden={0 1}</code>	
Option Sets Variable	Yes, ndbinfo_show_hidden [2159]	
Variable Name	<code>ndbinfo_show_hidden</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Whether or not the `ndbinfo` database's underlying internal tables are shown in the `mysql` client. The default is `OFF`.

- [ndbinfo_table_prefix \[2160\]](#)

Command-Line Format	<code>--ndbinfo-table-prefix=name</code>	
Option Sets Variable	Yes, <code>ndbinfo_table_prefix</code> [2160]	
Variable Name	<code>ndbinfo_table_prefix</code>	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>ndb\$</code>

The prefix used in naming the `ndbinfo` database's base tables (normally hidden, unless exposed by setting `ndbinfo_show_hidden` [2159]). This is a read-only variable whose default value is “`ndb$`”. You can start the server with the `--ndbinfo-table-prefix` option, but this merely sets the variable and does not change the actual prefix used to name the hidden base tables; the prefix itself is determined at compile time.

- [ndbinfo_version \[2160\]](#)

Variable Name	<code>ndbinfo_version</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>string</code>
	Default	

Shows the version of the `ndbinfo` engine in use; read-only.

- [ndb_log_empty_epochs \[2160\]](#)

Command-Line Format	<code>--ndb-log-empty-epochs</code>	
Option-File Format	<code>ndb_log_empty_epochs</code>	
Variable Name	<code>ndb_log_empty_epochs</code>	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>OFF</code>

When this variable is set to 0, epoch transactions with no changes are not written to the binary log, although a row is still written even for an empty epoch in `ndb_binlog_index`.

- [server_id_bits \[2160\]](#)

Command-Line Format	<code>--server-id-bits=#</code>	
Option-File Format	<code>server-id-bits</code>	
Option Sets Variable	Yes, <code>server_id_bits</code> [2160]	
Variable Name	<code>server_id_bits</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>32</code>
	Range	<code>7 .. 32</code>

The effective value of `server_id` if the server was started with the `--server-id-bits` [2144] option set to a nondefault value.

If the value of `server_id` [551] greater than or equal to 2 to the power of `server_id_bits` [2160], `mysqld` refuses to start.

This system variable is supported only by MySQL Cluster. `server_id_bits` [2160] is not supported by the standard MySQL Server.

17.3.4.4. MySQL Cluster Status Variables

This section provides detailed information about MySQL server status variables that relate to MySQL Cluster and the `NDB` storage engine. For status variables not specific to MySQL Cluster, and for general information on using status variables, see [Section 5.1.6, “Server Status Variables”](#).

- `Handler_discover` [2161]

The MySQL server can ask the `NDBCLUSTER` storage engine if it knows about a table with a given name. This is called discovery. `Handler_discover` [2161] indicates the number of times that tables have been discovered using this mechanism.

- `Ndb_api_wait_exec_complete_count_session` [2161]

The number of times a thread has been blocked in this client session while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit implicit executes for blob and auto-increment operations not visible to clients.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_wait_exec_complete_count_slave` [2161]

The number of times a thread has been blocked by this slave while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit implicit executes for blob and auto-increment operations not visible to clients.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use `NDB` tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_wait_exec_complete_count \[2162\]](#)

The number of times a thread has been blocked by this MySQL Server (SQL node) while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit implicit executes for blob and auto-increment operations not visible to clients.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_wait_scan_result_count_session \[2162\]](#)

The number of times a thread has been blocked in this client session while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_wait_scan_result_count_slave \[2162\]](#)

The number of times a thread has been blocked by this slave while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_wait_scan_result_count \[2162\]](#)

The number of times a thread has been blocked by this MySQL Server (SQL node) while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_wait_meta_request_count_session \[2162\]](#)

The number of times a thread has been blocked in this client session waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_wait_meta_request_count_slave \[2162\]](#)

The number of times a thread has been blocked by this slave waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_wait_meta_request_count` [2163]

The number of times a thread has been blocked by this MySQL Server (SQL node) waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_wait_nanos_count_session` [2163]

Total time (in nanoseconds) spent in this client session waiting for any type of signal from the data nodes.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_wait_nanos_count_slave` [2163]

Total time (in nanoseconds) spent by this slave waiting for any type of signal from the data nodes.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_wait_nanos_count` [2163]

Total time (in nanoseconds) spent by this MySQL Server (SQL node) waiting for any type of signal from the data nodes.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_bytes_sent_count_session` [2163]

Amount of data (in bytes) sent to the data nodes in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_bytes_sent_count_slave` [2164]

Amount of data (in bytes) sent to the data nodes by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_bytes_sent_count` [2164]

Amount of data (in bytes) sent to the data nodes by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_bytes_received_count_session` [2164]

Amount of data (in bytes) received from the data nodes in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_bytes_received_count_slave` [2164]

Amount of data (in bytes) received from the data nodes by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_bytes_received_count` [2164]

Amount of data (in bytes) received from the data nodes by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_trans_start_count_session` [2164]

The number of transactions started in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_trans_start_count_slave \[2165\]](#)

The number of transactions started by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_trans_start_count \[2165\]](#)

The number of transactions started by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_trans_commit_count_session \[2165\]](#)

The number of transactions committed in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_trans_commit_count_slave \[2165\]](#)

The number of transactions committed by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_trans_commit_count \[2165\]](#)

The number of transactions committed by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_trans_abort_count_session \[2165\]](#)

The number of transactions aborted in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_trans_abort_count_slave \[2165\]](#)

The number of transactions aborted by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_trans_abort_count` [2166]

The number of transactions aborted by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_trans_close_count_session` [2166]

The number of transactions closed in this client session. This value may be greater than the sum of `Ndb_api_trans_commit_count_session` [2165] and `Ndb_api_trans_abort_count_session` [2165], since some transactions may have been rolled back.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_trans_close_count_slave` [2166]

The number of transactions closed by this slave. This value may be greater than the sum of `Ndb_api_trans_commit_count_slave` [2165] and `Ndb_api_trans_abort_count_slave` [2165], since some transactions may have been rolled back.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_trans_close_count` [2166]

The number of transactions closed by this MySQL Server (SQL node). This value may be greater than the sum of `Ndb_api_trans_commit_count` [2165] and `Ndb_api_trans_abort_count` [2166], since some transactions may have been rolled back.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_pk_op_count_session` [2166]

The number of operations in this client session based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_pk_op_count_slave \[2167\]](#)

The number of operations by this slave based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_pk_op_count \[2167\]](#)

The number of operations by this MySQL Server (SQL node) based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_uk_op_count_session \[2167\]](#)

The number of operations in this client session based on or using unique keys.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_uk_op_count_slave \[2167\]](#)

The number of operations by this slave based on or using unique keys.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_uk_op_count \[2167\]](#)

The number of operations by this MySQL Server (SQL node) based on or using unique keys.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_table_scan_count_session \[2167\]](#)

The number of table scans that have been started in this client session, including scans of internal tables,.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_table_scan_count_slave` [2168]

The number of table scans that have been started by this slave, including scans of internal tables,.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_table_scan_count` [2168]

The number of table scans that have been started by this MySQL Server (SQL node), including scans of internal tables,.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_range_scan_count_session` [2168]

The number of range scans that have been started in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_range_scan_count_slave` [2168]

The number of range scans that have been started by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_range_scan_count` [2168]

The number of range scans that have been started by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_pruned_scan_count_session` [2168]

The number of scans in this client session that have been pruned to a single partition.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_pruned_scan_count_slave` [2169]

The number of scans by this slave that have been pruned to a single partition.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_pruned_scan_count` [2169]

The number of scans by this MySQL Server (SQL node) that have been pruned to a single partition.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_scan_batch_count_session` [2169]

The number of batches of rows received in this client session. 1 batch is defined as 1 set of scan results from a single fragment.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_scan_batch_count_slave` [2169]

The number of batches of rows received by this slave. 1 batch is defined as 1 set of scan results from a single fragment.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_scan_batch_count` [2169]

The number of batches of rows received by this MySQL Server (SQL node). 1 batch is defined as 1 set of scan results from a single fragment.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_read_row_count_session` [2169]

The total number of rows that have been read in this client session. This includes all rows read by any primary key, unique key, or scan operation made in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_read_row_count_slave` [2170]

The total number of rows that have been read by this slave. This includes all rows read by any primary key, unique key, or scan operation made by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_read_row_count` [2170]

The total number of rows that have been read by this MySQL Server (SQL node). This includes all rows read by any primary key, unique key, or scan operation made by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_trans_local_read_row_count_session` [2170]

The total number of rows that have been read in this client session. This includes all rows read by any primary key, unique key, or scan operation made in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_trans_local_read_row_count_slave` [2170]

The total number of rows that have been read by this slave. This includes all rows read by any primary key, unique key, or scan operation made by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- `Ndb_api_trans_local_read_row_count` [2170]

The total number of rows that have been read by this MySQL Server (SQL node). This includes all rows read by any primary key, unique key, or scan operation made by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_event_data_count_injector \[2171\]](#)

The number of row change events received by the NDB binlog injector thread.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_event_data_count \[2171\]](#)

The number of row change events received by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_event_nondata_count_injector \[2171\]](#)

The number of events received, other than row change events, by the NDB binlog injector thread.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_event_nondata_count \[2171\]](#)

The number of events received, other than row change events, by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_event_bytes_count_injector \[2171\]](#)

The number of bytes of events received by the NDB binlog injector thread.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_api_event_bytes_count \[2171\]](#)

The number of bytes of events received by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see [Section 17.5.15, “NDB API Statistics Counters and Variables”](#).

- [Ndb_cluster_node_id \[2171\]](#)

If the server is acting as a MySQL Cluster node, then the value of this variable is its node ID in the cluster.

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

- `Ndb_config_from_host` [2172]

If the server is part of a MySQL Cluster, the value of this variable is the host name or IP address of the Cluster management server from which it gets its configuration data.

If the server is not part of a MySQL Cluster, then the value of this variable is an empty string.

- `Ndb_config_from_port` [2172]

If the server is part of a MySQL Cluster, the value of this variable is the number of the port through which it is connected to the Cluster management server from which it gets its configuration data.

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

- `Ndb_conflict_fn_max`

Used in MySQL Cluster Replication conflict resolution, this variable shows the number of times that a row was not applied on the current SQL node due to “greatest timestamp wins” conflict resolution since the last time that this `mysqld` was started.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_conflict_fn_old`

Used in MySQL Cluster Replication conflict resolution, this variable shows the number of times that a row was not applied as the result of “same timestamp wins” conflict resolution on a given `mysqld` since the last time it was restarted.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_conflict_fn_epoch`

Used in MySQL Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH()` conflict resolution on a given `mysqld` since the last time it was restarted.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_conflict_fn_epoch_trans`

Used in MySQL Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH_TRANS()` conflict resolution on a given `mysqld` since the last time it was restarted.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_conflict_trans_row_conflict_count`

Used in MySQL Cluster Replication conflict resolution, this status variable shows the number of rows found to be directly in-conflict by a transactional conflict function on a given `mysqld` since the last time it was restarted.

Currently, the only transactional conflict detection function supported by MySQL Cluster is `NDB$EPOCH_TRANS()`, so this status variable is effectively the same as `Ndb_conflict_fn_epoch_trans` [2172].

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_conflict_trans_row_reject_count`

Used in MySQL Cluster Replication conflict resolution, this status variable shows the total number of rows realigned due to being determined as conflicting by a transactional conflict detection function. This includes not only `Ndb_conflict_trans_row_conflict_count` [2172], but any rows in or dependent on conflicting transactions.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_conflict_trans_reject_count`

Used in MySQL Cluster Replication conflict resolution, this status variable shows the number of transactions found to be in conflict by a transactional conflict detection function.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_conflict_trans_detect_iter_count`

Used in MySQL Cluster Replication conflict resolution, this shows the number of internal iterations required to commit an epoch transaction. Should be (slightly) greater than or equal to `Ndb_conflict_trans_conflict_commit_count` [2173].

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_conflict_trans_conflict_commit_count`

Used in MySQL Cluster Replication conflict resolution, this shows the number of epoch transactions committed after they required transactional conflict handling.

For more information, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

- `Ndb_execute_count` [2173]

Provides the number of round trips to the NDB kernel made by operations.

- `Ndb_number_of_data_nodes` [2173]

If the server is part of a MySQL Cluster, the value of this variable is the number of data nodes in the cluster.

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

- `Ndb_pushed_queries_defined` [2173]

The total number of joins pushed down to the NDB kernel for distributed handling on the data nodes. Note that joins tested using `EXPLAIN` that can be pushed down contribute to this number. Added in MySQL Cluster NDB 7.2.0.

- `Ndb_pushed_queries_dropped` [2173]

The number of joins that were pushed down to the NDB kernel but that could not be handled there. Added in MySQL Cluster NDB 7.2.0.

- [Ndb_pushed_queries_executed \[2174\]](#)

The number of joins successfully pushed down to `NDB` and executed there. Added in MySQL Cluster NDB 7.2.0.

- [Ndb_pushed_reads \[2174\]](#)

The number of rows returned to `mysqld` from the NDB kernel by joins that were pushed down. Note that executing `EXPLAIN` on joins that can be pushed down to `NDB` does not add to this number. Added in MySQL Cluster NDB 7.2.0.

- [Ndb_pruned_scan_count \[2174\]](#)

This variable holds a count of the number of scans executed by `NDBCLUSTER` since the MySQL Cluster was last started where `NDBCLUSTER` was able to use partition pruning.

Using this variable together with [Ndb_scan_count \[2174\]](#) can be helpful in schema design to maximize the ability of the server to prune scans to a single table partition, thereby involving only a single data node.

- [Ndb_scan_count \[2174\]](#)

This variable holds a count of the total number of scans executed by `NDBCLUSTER` since the MySQL Cluster was last started.

17.3.5. Using High-Speed Interconnects with MySQL Cluster

Even before design of `NDBCLUSTER` began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, `NDBCLUSTER` was designed from the very beginning to permit the use of a number of different data transport mechanisms. In this Manual, we use the term *transporter* for these.

The MySQL Cluster codebase provides for four different transporters:

- *TCP/IP using 100 Mbps or gigabit Ethernet*, as discussed in [Section 17.3.2.8, “MySQL Cluster TCP/IP Connections”](#).
- *Direct (machine-to-machine) TCP/IP*; although this transporter uses the same TCP/IP protocol as mentioned in the previous item, it requires setting up the hardware differently and is configured differently as well. For this reason, it is considered a separate transport mechanism for MySQL Cluster. See [Section 17.3.2.9, “MySQL Cluster TCP/IP Connections Using Direct Connections”](#), for details.
- *Shared memory (SHM)*. For more information about SHM, see [Section 17.3.2.10, “MySQL Cluster Shared-Memory Connections”](#).



Note

SHM is considered experimental only, and is not officially supported.

- *Scalable Coherent Interface (SCI)*, as described in the next section of this chapter, [Section 17.3.2.11, “SCI Transport Connections in MySQL Cluster”](#).

Most users today employ TCP/IP over Ethernet because it is ubiquitous. TCP/IP is also by far the best-tested transporter for use with MySQL Cluster.

We are working to make sure that communication with the `ndbd` process is made in “chunks” that are as large as possible because this benefits all types of data transmission.

For users who desire it, it is also possible to use cluster interconnects to enhance performance even further. There are two ways to achieve this: Either a custom transporter can be designed to handle this case, or you can use socket implementations that bypass the TCP/IP stack to one extent or another. We have experimented with both of these techniques using the SCI (Scalable Coherent Interface) technology developed by [Dolphin Interconnect Solutions](#).

17.3.5.1. Configuring MySQL Cluster to use SCI Sockets

It is possible employing Scalable Coherent Interface (SCI) technology to achieve a significant increase in connection speeds and throughput between MySQL Cluster data and SQL nodes. To use SCI, it is necessary to obtain and install Dolphin SCI network cards and to use the drivers and other software supplied by Dolphin. You can get information on obtaining these, from [Dolphin Interconnect Solutions](#). SCI SuperSocket or SCI Transporter support is available for 32-bit and 64-bit Linux, Solaris, Windows, and other platforms. See the Dolphin documentation referenced later in this section for more detailed information regarding platforms supported for SCI.

Once you have acquired the required Dolphin hardware and software, you can obtain detailed information on how to adapt a MySQL Cluster configured for normal TCP/IP communication to use SCI from the from the [Dolphin SCI online documentation](#).

17.3.5.2. MySQL Cluster Interconnects and Performance

The `ndbd` process has a number of simple constructs which are used to access the data in a MySQL Cluster. We have created a very simple benchmark to check the performance of each of these and the effects which various interconnects have on their performance.

There are four access methods:

- **Primary key access.** This is access of a record through its primary key. In the simplest case, only one record is accessed at a time, which means that the full cost of setting up a number of TCP/IP messages and a number of costs for context switching are borne by this single request. In the case where multiple primary key accesses are sent in one batch, those accesses share the cost of setting up the necessary TCP/IP messages and context switches. If the TCP/IP messages are for different destinations, additional TCP/IP messages need to be set up.
- **Unique key access.** Unique key accesses are similar to primary key accesses, except that a unique key access is executed as a read on an index table followed by a primary key access on the table. However, only one request is sent from the MySQL Server, and the read of the index table is handled by `ndbd`. Such requests also benefit from batching.
- **Full table scan.** When no indexes exist for a lookup on a table, a full table scan is performed. This is sent as a single request to the `ndbd` process, which then divides the table scan into a set of parallel scans on all cluster `ndbd` processes. In future versions of MySQL Cluster, an SQL node will be able to filter some of these scans.
- **Range scan using ordered index.** When an ordered index is used, it performs a scan in the same manner as the full table scan, except that it scans only those records which are in the range used by the query transmitted by the MySQL server (SQL node). All partitions are scanned in parallel when all bound index attributes include all attributes in the partitioning key.

With benchmarks developed internally by MySQL for testing simple and batched primary and unique key accesses, we have found that using SCI sockets improves performance by approximately 100% over TCP/IP, except in rare instances when communication performance is not an issue. This can occur when scan filters make up most of processing time or when very large batches of primary key accesses are achieved. In that case, the CPU processing in the `ndbd` processes becomes a fairly large part of the overhead.

Using the SCI transporter instead of SCI Sockets is only of interest in communicating between `ndbd` processes. Using the SCI transporter is also only of interest if a CPU can be dedicated to the `ndbd` process because the SCI transporter ensures that this process will never go to sleep. It is also important to ensure that the `ndbd` process priority is set in such a way that the process does not lose priority due to running for an extended period of time, as can be done by locking processes to CPUs in Linux 2.6. If such a configuration is possible, the `ndbd` process will benefit by 10–70% as compared with using SCI sockets. (The larger figures will be seen when performing updates and probably on parallel scan operations as well.)

There are several other optimized socket implementations for computer clusters, including Myrinet, Gigabit Ethernet, Infiniband and the VIA interface. However, we have tested MySQL Cluster so far only with SCI sockets. See [Section 17.3.5.1, “Configuring MySQL Cluster to use SCI Sockets”](#), for information on how to set up SCI sockets using ordinary TCP/IP for MySQL Cluster.

17.4. MySQL Cluster Programs

Using and managing a MySQL Cluster requires several specialized programs, which we describe in this chapter. We discuss the purposes of these programs in a MySQL Cluster, how to use the programs, and what startup options are available for each of them.

These programs include the MySQL Cluster data, management, and SQL node processes (`ndbd`, `ndbmt_d`, `ndb_mgmd`, and `mysqld`) and the management client (`ndb_mgm`).

For information about using `mysqld` as a MySQL Cluster process, see [Section 17.5.4, “MySQL Server Usage for MySQL Cluster”](#).

Other NDB utility, diagnostic, and example programs are included with the MySQL Cluster distribution. These include `ndb_restore`, `ndb_show_tables`, and `ndb_config`. These programs are also covered in this section.

The final portion of this section contains tables of options that are common to all the various MySQL Cluster programs.

17.4.1. `ndbd` — The MySQL Cluster Data Node Daemon

`ndbd` is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process that empowers a data node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In a MySQL Cluster, a set of `ndbd` processes cooperate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between data nodes and Cluster hosts is completely configurable.

The following table includes command options specific to the MySQL Cluster data node program `ndbd`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndbd`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.10. `ndbd` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--initial [2178]</code>	Perform initial start of <code>ndbd</code> , including cleaning the file system. Consult the documentation before using this option	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--nostart [2180], -n [2180]</code>	Don't start ndbd immediately; ndbd waits for command to start from ndb_mgmd	All MySQL 5.5 based releases
<code>--daemon [2178], -d [2178]</code>	Start ndbd as daemon (default); override with <code>--nodaemon</code>	All MySQL 5.5 based releases
<code>--nodaemon [2178]</code>	Do not start ndbd as daemon; provided for testing purposes	All MySQL 5.5 based releases
<code>--foreground [2178]</code>	Run ndbd in foreground, provided for debugging purposes (implies <code>--nodaemon</code>)	All MySQL 5.5 based releases
<code>--nowait-nodes=list [2180]</code>	Do not wait for these data nodes to start (takes comma-separated list of node IDs). Also requires <code>--ndb-nodeid</code> to be used.	All MySQL 5.5 based releases
<code>--initial-start [2179]</code>	Perform partial initial start (requires <code>--nowait-nodes</code>)	All MySQL 5.5 based releases
<code>--bind-address=name [2177]</code>	Local bind address	All MySQL 5.5 based releases
<code>--install[=name] [2180]</code>	Used to install the data node process as a Windows service. Does not apply on non-Windows platforms.	All MySQL 5.5 based releases
<code>--remove[=name] [2181]</code>	Used to remove a data node process that was previously installed as a Windows service. Does not apply on non-Windows platforms.	All MySQL 5.5 based releases
<code>--connect-retries=# [2181]</code>	Number of times to try contacting the management server; set to -1 to keep trying indefinitely	ADDED: NDB 7.2.9
<code>--connect-delay=# [2181]</code>	Time to wait between attempts to contact a management server, in seconds	ADDED: NDB 7.2.9



Note

All of these options also apply to the multi-threaded version of this program (`ndbmt.d`) and you may substitute “`ndbmt.d`” for “`ndbd`” wherever the latter occurs in this section.

- `--bind-address`

Command-Line Format	<code>--bind-address=name</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

Causes `ndbd` to bind to a specific network interface (host name or IP address). This option has no default value.

- `--daemon, -d`

Command-Line Format	<code>--daemon</code>	
	<code>-d</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>TRUE</code>

Instructs `ndbd` or `ndbmt` to execute as a daemon process. This is the default behavior. `--nodaemon` [2178] can be used to prevent the process from running as a daemon.

This option has no effect when running `ndbd` or `ndbmt` on Windows platforms.

- `--nodaemon`

Command-Line Format	<code>--nodaemon</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Prevents `ndbd` or `ndbmt` from executing as a daemon process. This option overrides the `--daemon` [2178] option. This is useful for redirecting output to the screen when debugging the binary.

The default behavior for `ndbd` and `ndbmt` on Windows is to run in the foreground, making this option unnecessary on Windows platforms, where it has no effect.

- `--foreground`

Command-Line Format	<code>--foreground</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Causes `ndbd` or `ndbmt` to execute as a foreground process, primarily for debugging purposes. This option implies the `--nodaemon` [2178] option.

This option has no effect when running `ndbd` or `ndbmt` on Windows platforms.

- `--initial`

Command-Line Format	<code>--initial</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Instructs `ndbd` to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of `ndbd`. It also re-creates recovery log files. Note that on some operating systems this process can take a substantial amount of time.

An `--initial` [2178] start is to be used *only* when starting the `ndbd` process under very special circumstances; this is because this option causes all files to be removed from the MySQL Cluster file system and all redo log files to be re-created. These circumstances are listed here:

- When performing a software upgrade which has changed the contents of any files.
- When restarting the node with a new version of `ndbd`.
- As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case, be aware that this node can no longer be used to restore data due to the destruction of the data files.

Use of this option prevents the `StartPartialTimeout` [2061] and `StartPartitionedTimeout` [2061] configuration parameters from having any effect.



Important

This option does *not* affect either of the following types of files:

- Backup files that have already been created by the affected node
- MySQL Cluster Disk Data files (see [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#)).

This option also has no effect on recovery of data by a data node that is just starting (or restarting) from data nodes that are already running. This recovery of data occurs automatically, and requires no user intervention in a MySQL Cluster that is running normally.

It is permissible to use this option when starting the cluster for the very first time (that is, before any data node files have been created); however, it is *not* necessary to do so.

- `--initial-start`

Command-Line Format	<code>--initial-start</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

This option is used when performing a partial initial start of the cluster. Each node should be started with this option, as well as `--nowait-nodes` [2180].

Suppose that you have a 4-node cluster whose data nodes have the IDs 2, 3, 4, and 5, and you wish to perform a partial initial start using only nodes 2, 4, and 5—that is, omitting node 3:

```
shell> ndbd --ndb-nodeid=2 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=4 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=5 --nowait-nodes=3 --initial-start
```

When using this option, you must also specify the node ID for the data node being started with the `--ndb-nodeid` [2242] option.



Important

Do not confuse this option with the `--nowait-nodes` [2191] option for `ndb_mgmd`, which can be used to enable a cluster configured with multiple management servers to be started without all management servers being online.

- `--nowait-nodes=node_id_1[, node_id_2[, ...]]`

Command-Line Format	<code>--nowait-nodes=list</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

This option takes a list of data nodes which for which the cluster will not wait for before starting.

This can be used to start the cluster in a partitioned state. For example, to start the cluster with only half of the data nodes (nodes 2, 3, 4, and 5) running in a 4-node cluster, you can start each `ndbd` process with `--nowait-nodes=3,5`. In this case, the cluster starts as soon as nodes 2 and 4 connect, and does *not* wait `StartPartitionedTimeout` [2061] milliseconds for nodes 3 and 5 to connect as it would otherwise.

If you wanted to start up the same cluster as in the previous example without one `ndbd` (say, for example, that the host machine for node 3 has suffered a hardware failure) then start nodes 2, 4, and 5 with `--nowait-nodes=3`. Then the cluster will start as soon as nodes 2, 4, and 5 connect and will not wait for node 3 to start.

- `--nostart, -n`

Command-Line Format	<code>--nostart</code>	
	<code>-n</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Instructs `ndbd` not to start automatically. When this option is used, `ndbd` connects to the management server, obtains configuration data from it, and initializes communication objects. However, it does not actually start the execution engine until specifically requested to do so by the management server. This can be accomplished by issuing the proper `START` command in the management client (see Section 17.5.2, “Commands in the MySQL Cluster Management Client”).

- `--install[=name]`

Command-Line Format	<code>--install[=name]</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>ndbd</code>

Causes `ndbd` to be installed as a Windows service. Optionally, you can specify a name for the service; if not set, the service name defaults to `ndbd`. Although it is preferable to specify other `ndbd` program options in a `my.ini` or `my.cnf` configuration file, it is possible to use together with `--install`.

However, in such cases, the `--install` option must be specified first, before any other options are given, for the Windows service installation to succeed.

It is generally not advisable to use this option together with the `--initial` [2178] option, since this causes the data node file system to be wiped and rebuilt every time the service is stopped and started. Extreme care should also be taken if you intend to use any of the other `ndbd` options that affect the starting of data nodes—including `--initial-start` [2179], `--nostart` [2180], and `--nowait-nodes` [2180]—together with `--install` [2180], and you should make absolutely certain you fully understand and allow for any possible consequences of doing so.

The `--install` [2180] option has no effect on non-Windows platforms.

- `--remove[=name]` [2181]

Command-Line Format	<code>--remove[=name]</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>ndbd</code>

Causes an `ndbd` process that was previously installed as a Windows service to be removed. Optionally, you can specify a name for the service to be uninstalled; if not set, the service name defaults to `ndbd`.

The `--remove` [2181] option has no effect on non-Windows platforms.

- `--connect-retries=#`

Version Introduced	5.5.27-ndb-7.2.9	
Command-Line Format	<code>--connect-retries=#</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>12</code>
	Range	<code>-1 .. 65535</code>

Determines the number of times that the data node attempts to contact a management server when starting. Setting this option to -1 causes the data node to keep trying to make contact indefinitely. The default is 12 attempts. The time to wait between attempts is controlled by the `--connect-delay` [2181] option.

This option was added in MySQL Cluster NDB 7.2.9.

- `--connect-delay=#`

Version Introduced	5.5.27-ndb-7.2.9	
Command-Line Format	<code>--connect-delay=#</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>5</code>
	Range	<code>0 .. 3600</code>

Determines the time to wait between attempts to contact a management server when starting (the time between attempts is controlled by the `--connect-retries` [2181] option). The default is 5 attempts.

This option was added in MySQL Cluster NDB 7.2.9.

`ndbd` generates a set of log files which are placed in the directory specified by `DataDir` [2042] in the `config.ini` configuration file.

These log files are listed below. `node_id` is the node's unique identifier. Note that `node_id` represents the node's unique identifier. For example, `ndb_2_error.log` is the error log generated by the data node whose node ID is 2.

- `ndb_node_id_error.log` is a file containing records of all crashes which the referenced `ndbd` process has encountered. Each record in this file contains a brief error string and a reference to a trace file for this crash. A typical entry in this file might appear as shown here:

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

Listings of possible `ndbd` exit codes and messages generated when a data node process shuts down prematurely can be found in [ndbd Error Messages](#).



Important

*The last entry in the error log file is not necessarily the newest one (nor is it likely to be). Entries in the error log are *not* listed in chronological order; rather, they correspond to the order of the trace files as determined in the `ndb_node_id_trace.log.next` file (see below). Error log entries are thus overwritten in a cyclical and not sequential fashion.*

- `ndb_node_id_trace.log.trace_id` is a trace file describing exactly what happened just before the error occurred. This information is useful for analysis by the MySQL Cluster development team.

It is possible to configure the number of these trace files that will be created before old files are overwritten. `trace_id` is a number which is incremented for each successive trace file.

- `ndb_node_id_trace.log.next` is the file that keeps track of the next trace file number to be assigned.
- `ndb_node_id_out.log` is a file containing any data output by the `ndbd` process. This file is created only if `ndbd` is started as a daemon, which is the default behavior.
- `ndb_node_id.pid` is a file containing the process ID of the `ndbd` process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.
- `ndb_node_id_signal.log` is a file used only in debug versions of `ndbd`, where it is possible to trace all incoming, outgoing, and internal messages with their data in the `ndbd` process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the `.pid` file remains in effect even after the process has terminated.

To start `ndbd`, it may also be necessary to specify the host name of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

See [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#), for additional information about this issue. [Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”](#), describes other options for `ndbd`.

When `ndbd` starts, it actually initiates two processes. The first of these is called the “angel process”; its only job is to discover when the execution process has been completed, and then to restart the `ndbd` process if it is configured to do so. Thus, if you attempt to kill `ndbd` using the Unix `kill` command, it is necessary to kill both processes, beginning with the angel process. The preferred method of terminating an `ndbd` process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent actions. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the `ndbd` process. In a multi-processor system performing a large number of operations (including updates), the `ndbd` process can consume up to 2 CPUs if permitted to do so.

For a machine with many CPUs it is possible to use several `ndbd` processes which belong to different node groups; however, such a configuration is still considered experimental and is not supported for MySQL 5.5 in a production setting. See [Section 17.1.6, “Known Limitations of MySQL Cluster”](#).

17.4.2. ndbinfo_select_all — Select From ndbinfo Tables

`ndbinfo_select_all` is a client program that selects all rows and columns from one or more tables in the `ndbinfo` database.

Not all `ndbinfo` tables can be accessed by this program. `ndbinfo_select_all` can access the `counters`, `diskpagebuffer`, `logbuffers`, `logspaces`, `nodes`, `resources`, `threadblocks`, `threadstat`, and `transporters` tables.

To select from one or more `ndbinfo` tables using `ndbinfo_select_all`, it is necessary to supply the names of the tables when invoking the program as shown here:

```
shell> ndbinfo_select_all table_name1 [table_name2] [...]
```

For example:

```
shell> ndbinfo_select_all logbuffers logspaces
== logbuffers ==
node_id log_type      log_id  log_part    total  used   high
5       0       0       0     33554432  262144  0
6       0       0       0     33554432  262144  0
7       0       0       0     33554432  262144  0
8       0       0       0     33554432  262144  0
== logspaces ==
node_id log_type      log_id  log_part    total  used   high
5       0       0       0     268435456  0      0
5       0       0       1     268435456  0      0
5       0       0       2     268435456  0      0
5       0       0       3     268435456  0      0
6       0       0       0     268435456  0      0
6       0       0       1     268435456  0      0
6       0       0       2     268435456  0      0
6       0       0       3     268435456  0      0
7       0       0       0     268435456  0      0
7       0       0       1     268435456  0      0
```

```

7      0      0      2      268435456      0      0
7      0      0      3      268435456      0      0
8      0      0      0      268435456      0      0
8      0      0      1      268435456      0      0
8      0      0      2      268435456      0      0
8      0      0      3      268435456      0      0
shell>

```

The following table includes options that are specific to `ndbinfo_select_all`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndbinfo_select_all`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.11. `ndbinfo_select_all` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--delay=# [2184]</code>	Set the delay in seconds between loops. Default is 5.	All MySQL 5.5 based releases
<code>--loops=# [2184], -l [2184]</code>	Set the number of times to perform the select. Default is 1.	All MySQL 5.5 based releases
<code>--database=db_name [2230], -d [2230]</code>	Name of the database where the table located.	All MySQL 5.5 based releases
<code>--parallelism=# [2230], -p [2230]</code>	Set the degree of parallelism.	All MySQL 5.5 based releases

- `--delay=seconds`

Command-Line Format	<code>--delay=#</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>5</code>
	Range	<code>0 .. MAX_INT</code>

This option sets the number of seconds to wait between executing loops. Has no effect if `--loops [2184]` is set to 0 or 1.

- `--loops=number, -l number`

Command-Line Format	<code>--loops=#</code>	
	<code>-l</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>1</code>
	Range	<code>0 .. MAX_INT</code>

This option sets the number of times to execute the select. Use `--delay [2184]` to set the time between loops.

17.4.3. `ndbmt d` — The MySQL Cluster Data Node Daemon (Multi-Threaded)

`ndbmtbd` is a multi-threaded version of `ndbd`, the process that is used to handle all the data in tables using the `NDBCLUSTER` storage engine. `ndbmtbd` is intended for use on host computers having multiple CPU cores. Except where otherwise noted, `ndbmtbd` functions in the same way as `ndbd`; therefore, in this section, we concentrate on the ways in which `ndbmtbd` differs from `ndbd`, and you should consult [Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”](#), for additional information about running MySQL Cluster data nodes that apply to both the single-threaded and multi-threaded versions of the data node process.

Command-line options and configuration parameters used with `ndbd` also apply to `ndbmtbd`. For more information about these options and parameters, see [Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”](#), and [Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”](#), respectively.

`ndbmtbd` is also file system-compatible with `ndbd`. In other words, a data node running `ndbd` can be stopped, the binary replaced with `ndbmtbd`, and then restarted without any loss of data. (However, when doing this, you must make sure that `MaxNoOfExecutionThreads` [\[2079\]](#) is set to an appropriate value before restarting the node if you wish for `ndbmtbd` to run in multi-threaded fashion.) Similarly, an `ndbmtbd` binary can be replaced with `ndbd` simply by stopping the node and then starting `ndbd` in place of the multi-threaded binary. It is not necessary when switching between the two to start the data node binary using `--initial` [\[2178\]](#).

Using `ndbmtbd` differs from using `ndbd` in two key respects:

1. Because `ndbmtbd` runs by default in single-threaded mode (that is, it behaves like `ndbd`), you must configure it to use multiple threads. This can be done by setting an appropriate value in the `config.ini` file for the `MaxNoOfExecutionThreads` [\[2079\]](#) configuration parameter or (in MySQL Cluster NDB 7.2.3 and later) the `ThreadConfig` [\[2081\]](#) configuration parameter. Using `MaxNoOfExecutionThreads` is simpler, but `ThreadConfig` offers more flexibility. For more information about these configuration parameters and their use, see [Multi-Threading Configuration Parameters](#) (`ndbmtbd`).
2. Trace files are generated by critical errors in `ndbmtbd` processes in a somewhat different fashion from how these are generated by `ndbd` failures. These differences are discussed in more detail in the next few paragraphs.

Like `ndbd`, `ndbmtbd` generates a set of log files which are placed in the directory specified by `DataDir` [\[2042\]](#) in the `config.ini` configuration file. Except for trace files, these are generated in the same way and have the same names as those generated by `ndbd`.

In the event of a critical error, `ndbmtbd` generates trace files describing what happened just prior to the error's occurrence. These files, which can be found in the data node's `DataDir` [\[2042\]](#), are useful for analysis of problems by the MySQL Cluster Development and Support teams. One trace file is generated for each `ndbmtbd` thread. The names of these files have the following pattern:

```
ndb_node_id_trace.log.trace_id_tthread_id,
```

In this pattern, `node_id` stands for the data node's unique node ID in the cluster, `trace_id` is a trace sequence number, and `thread_id` is the thread ID. For example, in the event of the failure of an `ndbmtbd` process running as a MySQL Cluster data node having the node ID 3 and with `MaxNoOfExecutionThreads` [\[2079\]](#) equal to 4, four trace files are generated in the data node's data directory. If this is the first time this node has failed, then these files are named `ndb_3_trace.log.1_t1`, `ndb_3_trace.log.1_t2`, `ndb_3_trace.log.1_t3`, and `ndb_3_trace.log.1_t4`. Internally, these trace files follow the same format as `ndbd` trace files.

The `ndbd` exit codes and messages that are generated when a data node process shuts down prematurely are also used by `ndbmtbd`. See [ndbd Error Messages](#), for a listing of these.

**Note**

It is possible to use `ndbd` and `ndbmt` concurrently on different data nodes in the same MySQL Cluster. However, such configurations have not been tested extensively; thus, we cannot recommend doing so in a production setting at this time.

17.4.4. `ndb_mgmd` — The MySQL Cluster Management Server Daemon

The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

The following table includes options that are specific to the MySQL Cluster management server program `ndb_mgmd`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_mgmd`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.12. `ndb_mgmd` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--config-file=file</code> [2188], <code>-f</code> [2188], <code>-c</code> [2188]	Specify the cluster configuration file; in NDB-6.4.0 and later, needs <code>--reload</code> or <code>--initial</code> to override configuration cache if present	All MySQL 5.5 based releases
<code>--configdir=directory</code> [2187]	Specify the cluster management server's configuration cache directory	All MySQL 5.5 based releases
<code>--bind-address=ip_address</code> [2187]	Local bind address	All MySQL 5.5 based releases
<code>--print-full-config</code> [2190], <code>-P</code> [2190]	Print full configuration and exit	All MySQL 5.5 based releases
<code>--daemon</code> [2189], <code>-d</code> [2189]	Run <code>ndb_mgmd</code> in daemon mode (default)	All MySQL 5.5 based releases
<code>--nodaemon</code> [2190]	Do not run <code>ndb_mgmd</code> as a daemon	All MySQL 5.5 based releases
<code>--interactive</code> [2189]	Run <code>ndb_mgmd</code> in interactive mode (not officially supported in production; for testing purposes only)	All MySQL 5.5 based releases
<code>--log-name=name</code> [2190]	A name to use when writing messages applying to this node in the cluster log.	All MySQL 5.5 based releases
<code>--no-nodeid-checks</code> [2187]	Do not provide any node id checks	All MySQL 5.5 based releases
<code>--mycnf</code> [2189]	Read cluster configuration data from the <code>my.cnf</code> file	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--reload [2191]</code>	Causes the management server to compare the configuration file with its configuration cache	All MySQL 5.5 based releases
<code>--initial [2190]</code>	Causes the management server reload its configuration data from the configuration file, bypassing the configuration cache	All MySQL 5.5 based releases
<code>--nowait-nodes=list [2191]</code>	Do not wait for these management nodes when starting this management server. Also requires <code>--ndb-nodeid</code> to be used.	All MySQL 5.5 based releases
<code>--config-cache=value [2188]</code>	Enable the management server configuration cache; ON by default.	All MySQL 5.5 based releases
<code>--install[=name] [2193]</code>	Used to install the management server process as a Windows service. Does not apply on non-Windows platforms.	All MySQL 5.5 based releases
<code>--remove[=name] [2194]</code>	Used to remove a management server process that was previously installed as a Windows service, optionally specifying the name of the service to be removed. Does not apply on non-Windows platforms.	All MySQL 5.5 based releases

- `--bind-address=host[:port]`

Command-Line Format	<code>--bind-address=ip_address</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>[none]</code>

When specified, this option limits management server connections by management clients to clients at the specified host name or IP address (and possibly port, if this is also specified). In such cases, a management client attempting to connect to the management server from any other address fails with the error `Unable to setup port: host:port!`

If the `port` is not specified, the management client attempts to use port 1186.

- `--no-nodeid-checks`

Command-Line Format	<code>--no-nodeid-checks</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Do not perform any checks of node IDs.

- `--configdir=path`

Command-Line Format	<code>--configdir=directory</code>	
	Permitted Values	
	Type	<code>file name</code>
	Default	<code>\$INSTALLDIR/mysql-cluster</code>

Specifies the cluster management server's configuration cache directory. `--config-dir` is an alias for this option.

- This option, whose default value is `1` (or `TRUE`, or `ON`), can be used to disable the management server's configuration cache, so that it reads its configuration from `config.ini` every time it starts (see Section 17.3.2, “MySQL Cluster Configuration Files”). You can do this by starting the `ndb_mgmd` process with any one of the following options:

- `--config-cache=0`
- `--config-cache=FALSE`
- `--config-cache=OFF`
- `--skip-config-cache`

Using one of the options just listed is effective only if the management server has no stored configuration at the time it is started. If the management server finds any configuration cache files, then the `--config-cache` option or the `--skip-config-cache` option is ignored. Therefore, to disable configuration caching, the option should be used the *first* time that the management server is started. Otherwise—that is, if you wish to disable configuration caching for a management server that has *already* created a configuration cache—you must stop the management server, delete any existing configuration cache files manually, then restart the management server with `--skip-config-cache` (or with `--config-cache` set equal to `0`, `OFF`, or `FALSE`).

Configuration cache files are normally created in a directory named `mysql-cluster` under the installation directory (unless this location has been overridden using the `--configdir` [2187] option). Each time the management server updates its configuration data, it writes a new cache file. The files are named sequentially in order of creation using the following format:

```
ndb_node-id_config.bin.seq-number
```

`node-id` is the management server's node ID; `seq-number` is a sequence number, beginning with 1. For example, if the management server's node ID is 5, then the first three configuration cache files would, when they are created, be named `ndb_5_config.bin.1`, `ndb_5_config.bin.2`, and `ndb_5_config.bin.3`.

If your intent is to purge or reload the configuration cache without actually disabling caching, you should start `ndb_mgmd` with one of the options `--reload` [2191] or `--initial` [2190] instead of `--skip-config-cache`.

To re-enable the configuration cache, simply restart the management server, but without the `--config-cache` or `--skip-config-cache` option that was used previously to disable the configuration cache.

Beginning with MySQL Cluster NDB 7.2.5, `ndb_mgmd` no longer checks for the configuration directory (`--configdir` [2187]) or attempts to create one when `--skip-config-cache` is used. (Bug #13428853)

- `--config-file=filename, -f filename`

Command-Line Format	<code>--config-file=file</code>	
	<code>-f</code>	
	<code>-c</code>	
	Permitted Values	
	Type	file name
	Default	<code>./config.ini</code>

Instructs the management server as to which file it should use for its configuration file. By default, the management server looks for a file named `config.ini` in the same directory as the `ndb_mgmd` executable; otherwise the file name and location must be specified explicitly.

This option is ignored unless the management server is forced to read the configuration file, either because `ndb_mgmd` was started with the `--reload` [2191] or `--initial` [2190] option, or because the management server could not find any configuration cache. This option is also read if `ndb_mgmd` was started with `--config-cache=OFF` [2188]. See Section 17.3.2, “MySQL Cluster Configuration Files”, for more information.

- `--mycnf`

Command-Line Format	<code>--mycnf</code>	
	Permitted Values	
	Type	boolean
	Default	<code>FALSE</code>

Read configuration data from the `my.cnf` file.

- `--daemon, -d`

Command-Line Format	<code>--daemon</code>	
	<code>-d</code>	
	Permitted Values	
	Type	boolean
	Default	<code>TRUE</code>

Instructs `ndb_mgmd` to start as a daemon process. This is the default behavior.

This option has no effect when running `ndb_mgmd` on Windows platforms.

- `--interactive`

Command-Line Format	<code>--interactive</code>	
	Permitted Values	
	Type	boolean
	Default	<code>FALSE</code>

Starts `ndb_mgmd` in interactive mode; that is, an `ndb_mgm` client session is started as soon as the management server is running. This option does not start any other MySQL Cluster nodes.

- `--initial`

Command-Line Format	<code>--initial</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Configuration data is cached internally, rather than being read from the cluster global configuration file each time the management server is started (see [Section 17.3.2, “MySQL Cluster Configuration Files”](#)). Using the `--initial` option overrides this behavior, by forcing the management server to delete any existing cache files, and then to re-read the configuration data from the cluster configuration file and to build a new cache.

This differs in two ways from the `--reload` [2191] option. First, `--reload` forces the server to check the configuration file against the cache and reload its data only if the contents of the file are different from the cache. Second, `--reload` does not delete any existing cache files.

If `ndb_mgmd` is invoked with `--initial` but cannot find a global configuration file, the management server cannot start.

When a management server starts, it checks for another management server in the same MySQL Cluster and tries to use the other management server's configuration data; `ndb_mgmd` ignores `--initial` [2190] unless it is the only management server running. This behavior also has implications when performing a rolling restart of a MySQL Cluster with multiple management nodes. See [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#), for more information.

- `--log-name=name`

Command-Line Format	<code>--log-name=name</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>MgmtSrvr</code>

Provides a name to be used for this node in the cluster log.

- `--nodaemon`

Command-Line Format	<code>--nodaemon</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Instructs `ndb_mgmd` not to start as a daemon process.

The default behavior for `ndb_mgmd` on Windows is to run in the foreground, making this option unnecessary on Windows platforms.

- `--print-full-config, -P`

Command-Line Format	<code>--print-full-config</code>	
	<code>-P</code>	

	Permitted Values	
Type	boolean	
Default	FALSE	

Shows extended information regarding the configuration of the cluster. With this option on the command line the `ndb_mgmd` process prints information about the cluster setup including an extensive list of the cluster configuration sections as well as parameters and their values. Normally used together with the `--config-file [2188] (-f)` option.

- `--reload`

Command-Line Format	<code>--reload</code>	
	Permitted Values	
Type	boolean	
Default	FALSE	

In MySQL Cluster NDB 7.2, configuration data is stored internally rather than being read from the cluster global configuration file each time the management server is started (see [Section 17.3.2, “MySQL Cluster Configuration Files”](#)). Using this option forces the management server to check its internal data store against the cluster configuration file and to reload the configuration if it finds that the configuration file does not match the cache. Existing configuration cache files are preserved, but not used.

This differs in two ways from the `--initial [2190]` option. First, `--initial` causes all cache files to be deleted. Second, `--initial` forces the management server to re-read the global configuration file and construct a new cache.

If the management server cannot find a global configuration file, then the `--reload` option is ignored.

When a management server starts, it checks for another management server in the same MySQL Cluster and tries to use the other management server's configuration data; `ndb_mgmd` ignores `--reload [2191]` unless it is the only management server running. This behavior also has implications when performing a rolling restart of a MySQL Cluster with multiple management nodes. See [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#), for more information.

- `--nowait-nodes`

Command-Line Format	<code>--nowait-nodes=list</code>	
	Permitted Values	
Type	numeric	
Default		
Range	1 .. 255	

When starting a MySQL Cluster is configured with two management nodes, each management server normally checks to see whether the other `ndb_mgmd` is also operational and whether the other management server's configuration is identical to its own. However, it is sometimes desirable to start the cluster with only one management node (and perhaps to allow the other `ndb_mgmd` to be started later). This option causes the management node to bypass any checks for any other management nodes whose node IDs are passed to this option, permitting the cluster to start as though configured to use only the management node that was started.

For purposes of illustration, consider the following portion of a `config.ini` file (where we have omitted most of the configuration parameters that are not relevant to this example):

```
[ndbd]
NodeId = 1
HostName = 192.168.0.101

[ndbd]
NodeId = 2
HostName = 192.168.0.102

[ndbd]
NodeId = 3
HostName = 192.168.0.103

[ndbd]
NodeId = 4
HostName = 192.168.0.104

[ndb_mgmd]
NodeId = 10
HostName = 192.168.0.150

[ndb_mgmd]
NodeId = 11
HostName = 192.168.0.151

[api]
NodeId = 20
HostName = 192.168.0.200

[api]
NodeId = 21
HostName = 192.168.0.201
```

Assume that you wish to start this cluster using only the management server having node ID 10 and running on the host having the IP address 192.168.0.150. (Suppose, for example, that the host computer on which you intend to the other management server is temporarily unavailable due to a hardware failure, and you are waiting for it to be repaired.) To start the cluster in this way, use a command line on the machine at 192.168.0.150 to enter the following command:

```
shell> ndb_mgmd --ndb-nodeid=10 --nowait-nodes=11
```

As shown in the preceding example, when using `--nowait-nodes` [2191], you must also use the `--ndb-nodeid` [2242] option to specify the node ID of this `ndb_mgmd` process.

You can then start each of the cluster's data nodes in the usual way. If you wish to start and use the second management server in addition to the first management server at a later time without restarting the data nodes, you must start each data node with a connectstring that references both management servers, like this:

```
shell> ndbd -c 192.168.0.150,192.168.0.151
```

The same is true with regard to the connectstring used with any `mysqld` processes that you wish to start as MySQL Cluster SQL nodes connected to this cluster. See [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#), for more information.

When used with `ndb_mgmd`, this option affects the behavior of the management node with regard to other management nodes only. Do not confuse it with the `--nowait-nodes` [2180] option used with `ndbd` or `ndbmtid` to permit a cluster to start with fewer than its full complement of data nodes; when used with data nodes, this option affects their behavior only with regard to other data nodes.

Multiple management node IDs may be passed to this option as a comma-separated list. Each node ID must be no less than 1 and no greater than 255. In practice, it is quite rare to use more than two management servers for the same MySQL Cluster (or to have any need for doing so); in most cases you need to pass to this option only the single node ID for the one management server that you do not wish to use when starting the cluster.



Note

When you later start the “missing” management server, its configuration must match that of the management server that is already in use by the cluster. Otherwise, it fails the configuration check performed by the existing management server, and does not start.

It is not strictly necessary to specify a connectstring when starting the management server. However, if you are using more than one management server, a connectstring should be provided and each node in the cluster should specify its node ID explicitly.

See [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#), for information about using connectstrings. [Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”](#), describes other options for `ndb_mgmd`.

The following files are created or used by `ndb_mgmd` in its starting directory, and are placed in the `DataDir [2042]` as specified in the `config.ini` configuration file. In the list that follows, `node_id` is the unique node identifier.

- `config.ini` is the configuration file for the cluster as a whole. This file is created by the user and read by the management server. [Section 17.3, “MySQL Cluster Configuration”](#), discusses how to set up this file.
- `ndb_node_id_cluster.log` is the cluster events log file. Examples of such events include checkpoint startup and completion, node startup events, node failures, and levels of memory usage. A complete listing of cluster events with descriptions may be found in [Section 17.5, “Management of MySQL Cluster”](#).

When the size of the cluster log reaches one million bytes, the file is renamed to `ndb_node_id_cluster.log.seq_id`, where `seq_id` is the sequence number of the cluster log file. (For example: If files with the sequence numbers 1, 2, and 3 already exist, the next log file is named using the number 4.)

- `ndb_node_id_out.log` is the file used for `stdout` and `stderr` when running the management server as a daemon.
- `ndb_node_id.pid` is the process ID file used when running the management server as a daemon.
- `--install[=name]`

Command-Line Format	<code>--install[=name]</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>ndb_mgmd</code>

Causes `ndb_mgmd` to be installed as a Windows service. Optionally, you can specify a name for the service; if not set, the service name defaults to `ndb_mgmd`. Although it is preferable to specify other `ndb_mgmd` program options in a `my.ini` or `my.cnf` configuration file, it is possible to use them together

with `--install` [2193]. However, in such cases, the `--install` [2193] option must be specified first, before any other options are given, for the Windows service installation to succeed.

It is generally not advisable to use this option together with the `--initial` [2178] option, since this causes the configuration cache to be wiped and rebuilt every time the service is stopped and started. Care should also be taken if you intend to use any other `ndb_mgmd` options that affect the starting of the management server, and you should make absolutely certain you fully understand and allow for any possible consequences of doing so.

The `--install` [2193] option has no effect on non-Windows platforms.

- `--remove[=name]` [2194]

Command-Line Format	<code>--remove[=name]</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>ndb_mgmd</code>

Causes an `ndb_mgmd` process that was previously installed as a Windows service to be removed. Optionally, you can specify a name for the service to be uninstalled; if not set, the service name defaults to `ndb_mgmd`.

The `--remove` [2194] option has no effect on non-Windows platforms.

17.4.5. ndb_mgm — The MySQL Cluster Management Client

The `ndb_mgm` management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client accesses the management server using a C API. Advanced users can also employ this API for programming dedicated management processes to perform tasks similar to those performed by `ndb_mgm`.

To start the management client, it is necessary to supply the host name and port number of the management server:

```
shell> ndb_mgm [host_name [port_num]]
```

For example:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

The default host name and port number are `localhost` and 1186, respectively.

The following table includes options that are specific to the MySQL Cluster management client program `ndb_mgm`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_mgm`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.13. ndb_mgm Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--try-reconnect=#</code> [2195], <code>-t</code> [2195]	Specify number of tries for connecting to <code>ndb_mgmd</code> (0 = infinite)	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--execute=name [2195],</code> <code>-e [2195]</code>	Execute command and exit	All MySQL 5.5 based releases

- `--execute=command, -e command`

Command-Line Format	
	<code>--execute=name</code>
	<code>-e</code>

This option can be used to send a command to the MySQL Cluster management client from the system shell. For example, either of the following is equivalent to executing `SHOW` in the management client:

```
shell> ndb_mgm -e "SHOW"
shell> ndb_mgm --execute="SHOW"
```

This is analogous to how the `--execute [278]` or `-e` option works with the `mysql` command-line client. See [Section 4.2.3.1, “Using Options on the Command Line”](#).



Note

If the management client command to be passed using this option contains any space characters, then the command *must* be enclosed in quotation marks. Either single or double quotation marks may be used. If the management client command contains no space characters, the quotation marks are optional.

- `--try-reconnect=number`

Command-Line Format	<code>--try-reconnect=#</code>	
	<code>-t</code>	
	Permitted Values	
	Type	boolean
	Default	TRUE

If the connection to the management server is broken, the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to `number` before giving up and reporting an error instead.

Additional information about using `ndb_mgm` can be found in [Section 17.5.2, “Commands in the MySQL Cluster Management Client”](#).

17.4.6. ndb_config — Extract MySQL Cluster Configuration Information

This tool extracts current configuration information for data nodes, SQL nodes, and API nodes from one of a number of sources: a MySQL Cluster management node, or its `config.ini` or `my.cnf` file. By default, the management node is the source for the configuration data; to override the default, execute `ndb_config` with the `--config-file [2197]` or `--mycnf [2198]` option. It is also possible to use a data node as the source by specifying its node ID with `--config_from_node=node_id [2197]`.

`ndb_config` can also provide an offline dump of all configuration parameters which can be used, along with their default, maximum, and minimum values and other information. The dump can be produced in either text or XML format; for more information, see the discussion of the `--configinfo [2200]` and `--xml [2201]` options later in this section).

You can filter the results by section (`DB`, `SYSTEM`, or `CONNECTIONS`) using one of the options `--nodes [2199]`, `--system [2199]`, or `--connections [2199]`.

The following table includes options that are specific to `ndb_config`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_config`), see Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.14. `ndb_config` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--nodes [2199]</code>	Print node information (DB section) only.	All MySQL 5.5 based releases
<code>--connections [2199]</code>	Print CONNECTIONS section information only. Cannot be used with <code>--nodes</code> or <code>--system</code> option.	All MySQL 5.5 based releases
<code>--query=string [2198]</code> , <code>-q [2198]</code>	One or more query options (attributes)	All MySQL 5.5 based releases
<code>--host=name [2198]</code>	Specify host	All MySQL 5.5 based releases
<code>--type=name [2199]</code>	Specify node type	All MySQL 5.5 based releases
<code>--nodeid [2199]</code> , <code>--id [2199]</code>	Get configuration of node with this ID	All MySQL 5.5 based releases
<code>--fields=string [2200]</code> , <code>-f [2200]</code>	Field separator	All MySQL 5.5 based releases
<code>--rows=string [2200]</code> , <code>-r [2200]</code>	Row separator	All MySQL 5.5 based releases
<code>--config-file=path [2197]</code>	Set the path to config.ini file	All MySQL 5.5 based releases
<code>--mycnf [2198]</code>	Read configuration data from my.cnf file	All MySQL 5.5 based releases
<code>-c [2241]</code>	Short form for <code>--ndb-connectstring</code>	All MySQL 5.5 based releases
<code>--configinfo [2200]</code>	Dumps information about all NDB configuration parameters in text format with default, maximum, and minimum values. Use with <code>--xml</code> to obtain XML output.	All MySQL 5.5 based releases
<code>--configinfo --xml [2201]</code>	Use <code>--xml</code> with <code>--configinfo</code> to obtain a dump of all NDB configuration parameters in XML format with default, maximum, and minimum values.	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--system [2199]</code>	Print SYSTEM section information only. Cannot be used with <code>--nodes</code> or <code>--connections</code> option.	All MySQL 5.5 based releases
<code>--config_from_node=# [2197]</code>	Obtain configuration data from the node having this ID (must be a data node).	All MySQL 5.5 based releases

- `--usage`, `--help`, or `-?`

Command-Line Format	<code>--help</code>
	<code>--usage</code>
	<code>-?</code>

Causes `ndb_config` to print a list of available options, and then exit.

- `--config_from_node=#`

Command-Line Format	<code>--config_from_node=#</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>none</code>
	Range	<code>1 .. 48</code>

Obtain the cluster's configuration data from the data node that has this ID.

If the node having this ID is not a data node, `ndb_config` fails with an error. (To obtain configuration data from the management node instead, simply omit this option.)

- `--version`, `-V`

Command-Line Format	<code>--version</code>
	<code>-V</code>

Causes `ndb_config` to print a version information string, and then exit.

- `--ndb-connectstring=connect_string`, `-c connect_string`

Specifies the connectstring to use in connecting to the management server. The format for the connectstring is the same as described in [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#), and defaults to `localhost:1186`.

- `--config-file=path-to-file`

Command-Line Format	<code>--config-file=path</code>	
	Permitted Values	
	Type	<code>file name</code>
	Default	

Gives the path to the management server's configuration file (`config.ini`). This may be a relative or absolute path. If the management node resides on a different host from the one on which `ndb_config` is invoked, then an absolute path must be used.

- `--mycnf`

Command-Line Format	<code>--mycnf</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Read configuration data from the `my.cnf` file.

- `--query=query-options, -q query-options`

Command-Line Format	<code>--query=string</code>	
	<code>-q</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

This is a comma-delimited list of *query options*—that is, a list of one or more node attributes to be returned. These include `id` (node ID), `type` (node type—that is, `ndbd`, `mysqld`, or `ndb_mgmd`), and any configuration parameters whose values are to be obtained.

For example, `--query=id,type,indexmemory,damemory` returns the node ID, node type, `DataMemory` [2043], and `IndexMemory` [2044] for each node.



Note

If a given parameter is not applicable to a certain type of node, than an empty string is returned for the corresponding value. See the examples later in this section for more information.

- `--host=hostname`

Command-Line Format	<code>--host=name</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

Specifies the host name of the node for which configuration information is to be obtained.



Note

While the hostname `localhost` usually resolves to the IP address `127.0.0.1`, this may not necessarily be true for all operating platforms and configurations. This means that it is possible, when `localhost` is used in `config.ini`, for `ndb_config --host=localhost` to fail if `ndb_config` is run on a different host where `localhost` resolves to a different address (for example, on some versions of SUSE Linux, this is `127.0.0.2`). In general, for best results, you

should use numeric IP addresses for all MySQL Cluster configuration values relating to hosts, or verify that all MySQL Cluster hosts handle `localhost` in the same fashion.

- `--id=node_id`

`--nodeid=node_id`

Either of these options can be used to specify the node ID of the node for which configuration information is to be obtained. `--nodeid` is the preferred form.

- `--nodes`

Command-Line Format	<code>--nodes</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Tells `ndb_config` to print information from parameters defined in `DB` sections only. This option cannot be used together with `--connections` [2199] or `--system` [2199].

- `--connections`

Command-Line Format	<code>--connections</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Tells `ndb_config` to print `CONNECTIONS` information only. This option cannot be used together with `--nodes` [2199] or `--system` [2199].

- `--system`

Command-Line Format	<code>--system</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Tells `ndb_config` to print `SYSTEM` information only.

This option cannot be used together with the `--nodes` [2199] or `--system` [2199] options.

- `--type=node_type`

Command-Line Format	<code>--type=name</code>	
	Permitted Values	
	Type	<code>enumeration</code>
	Default	
	Valid Values	<code>ndbd</code> <code>mysqld</code>

		ndb_mgmd
--	--	----------

Filters results so that only configuration values applying to nodes of the specified *node_type* (*ndbd*, *mysqld*, or *ndb_mgmd*) are returned.

- `--fields=delimiter, -f delimiter`

Command-Line Format	<code>--fields=string</code>	
	<code>-f</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

Specifies a *delimiter* string used to separate the fields in the result. The default is “,” (the comma character).



Note

If the *delimiter* contains spaces or escapes (such as `\n` for the linefeed character), then it must be quoted.

- `--rows=separator, -r separator`

Command-Line Format	<code>--rows=string</code>	
	<code>-r</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

Specifies a *separator* string used to separate the rows in the result. The default is a space character.



Note

If the *separator* contains spaces or escapes (such as `\n` for the linefeed character), then it must be quoted.

- `--configinfo`

The `--configinfo` option causes *ndb_config* to dump a list of each MySQL Cluster configuration parameter supported by the MySQL Cluster distribution of which *ndb_config* is a part, including the following information:

- A brief description of each parameter's purpose, effects, and usage
- The section of the `config.ini` file where the parameter may be used
- The parameter's data type or unit of measurement
- Where applicable, the parameter's default, minimum, and maximum values
- A brief description of the parameter's purpose, effects, and usage

- MySQL Cluster release version and build information

By default, this output is in text format. Part of this output is shown here:

```
shell> ndb_config --configinfo

***** SYSTEM *****

Name (String)
Name of system (NDB Cluster)
MANDATORY

PrimaryMGMNode (Non-negative Integer)
Node id of Primary ndb_mgmd(MGM) node
Default: 0 (Min: 0, Max: 4294967039)

ConfigGenerationNumber (Non-negative Integer)
Configuration generation number
Default: 0 (Min: 0, Max: 4294967039)

***** DB *****

MaxNoOfSubscriptions (Non-negative Integer)
Max no of subscriptions (default 0 == MaxNoOfTables)
Default: 0 (Min: 0, Max: 4294967039)

MaxNoOfSubscribers (Non-negative Integer)
Max no of subscribers (default 0 == 2 * MaxNoOfTables)
Default: 0 (Min: 0, Max: 4294967039)

...
```

`--configinfo --xml`

Command-Line Format	<code>--configinfo --xml</code>	
	Permitted Values	
	Type	boolean
	Default	false

You can obtain the output of `ndb_config --configinfo [2200]` as XML by adding the `--xml` option. A portion of the resulting output is shown in this example:

```
shell> ndb_config --configinfo --xml

<configvariables protocolversion="1" ndbversionstring="5.5.29-ndb-7.2.10"
  ndbversion="458758" ndbversionmajor="7" ndbversionminor="0"
  ndbversionbuild="6">
  <section name="SYSTEM">
    <param name="Name" comment="Name of system (NDB Cluster)" type="string"
      mandatory="true"/>
    <param name="PrimaryMGMNode" comment="Node id of Primary ndb_mgmd(MGM) node"
      type="unsigned" default="0" min="0" max="4294967039"/>
    <param name="ConfigGenerationNumber" comment="Configuration generation number"
      type="unsigned" default="0" min="0" max="4294967039"/>
  </section>
  <section name="NDBD">
    <param name="MaxNoOfSubscriptions"
      comment="Max no of subscriptions (default 0 == MaxNoOfTables)"
      type="unsigned" default="0" min="0" max="4294967039"/>
    <param name="MaxNoOfSubscribers"
      comment="Max no of subscribers (default 0 == 2 * MaxNoOfTables)"
      type="unsigned" default="0" min="0" max="4294967039"/>
  </section>
</configvariables>
```

```

...
</section>
...
</configvariables>

```



Note

Normally, the XML output produced by `ndb_config --configinfo --xml` is formatted using one line per element; we have added extra whitespace in the previous example, as well as the next one, for reasons of legibility. This should not make any difference to applications using this output, since most XML processors either ignore nonessential whitespace as a matter of course, or can be instructed to do so.

The XML output also indicates when changing a given parameter requires that data nodes be restarted using the `--initial [2178]` option. This is shown by the presence of an `initial="true"` attribute in the corresponding `<param>` element. In addition, the restart type (`system` or `node`) is also shown; if a given parameter requires a system restart, this is indicated by the presence of a `restart="system"` attribute in the corresponding `<param>` element. For example, changing the value set for the `Diskless [2058]` parameter requires a system initial restart, as shown here (with the `restart` and `initial` attributes highlighted for visibility):

```
<param name="Diskless" comment="Run wo/ disk" type="bool" default="false"
      restart="system" initial="true"/>
```

Currently, no `initial` attribute is included in the XML output for `<param>` elements corresponding to parameters which do not require initial restarts; in other words, `initial="false"` is the default, and the value `false` should be assumed if the attribute is not present. Similarly, the default restart type is `node` (that is, an online or “rolling” restart of the cluster), but the `restart` attribute is included only if the restart type is `system` (meaning that all cluster nodes must be shut down at the same time, then restarted).



Important

The `--xml` option can be used only with the `--configinfo` option. Using `--xml` without `--configinfo` fails with an error.

Unlike the options used with this program to obtain current configuration data, `--configinfo` and `--xml` use information obtained from the MySQL Cluster sources when `ndb_config` was compiled. For this reason, no connection to a running MySQL Cluster or access to a `config.ini` or `my.cnf` file is required for these two options.

Combining other `ndb_config` options (such as `--query [2198]` or `--type [2199]`) with `--configinfo` or `--xml` is not supported. Currently, if you attempt to do so, the usual result is that all other options besides `--configinfo` or `--xml` are simply ignored. *However, this behavior is not guaranteed and is subject to change at any time.* In addition, since `ndb_config`, when used with the `--configinfo` option, does not access the MySQL Cluster or read any files, trying to specify additional options such as `--ndb-connectstring` or `--config-file` with `--configinfo` serves no purpose.

Examples

1. To obtain the node ID and type of each node in the cluster:

```
shell> ./ndb_config --query=id,type --fields=':' --rows='\n'
```

```
1:ndbd
2:ndbd
3:ndbd
4:ndbd
5:ndb_mgmd
6:mysql
7:mysql
8:mysql
9:mysql
```

In this example, we used the `--fields [2200]` options to separate the ID and type of each node with a colon character (:), and the `--rows [2200]` options to place the values for each node on a new line in the output.

2. To produce a connectstring that can be used by data, SQL, and API nodes to connect to the management server:

```
shell> ./ndb_config --config-file=usr/local/mysql/cluster-data/config.ini \
--query=hostname,portnumber --fields=: --rows=, --type=ndb_mgmd
192.168.0.179:1186
```

3. This invocation of `ndb_config` checks only data nodes (using the `--type [2199]` option), and shows the values for each node's ID and host name, as well as the values set for its `DataMemory [2043]`, `IndexMemory [2044]`, and `DataDir [2042]` parameters:

```
shell> ./ndb_config --type=ndbd --query=id,host,datememory,indexmemory,datadir -f ' : ' -r '\n'
1 : 192.168.0.193 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
2 : 192.168.0.112 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
3 : 192.168.0.176 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
4 : 192.168.0.119 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
```

In this example, we used the short options `-f` and `-r` for setting the field delimiter and row separator, respectively.

4. To exclude results from any host except one in particular, use the `--host [2198]` option:

```
shell> ./ndb_config --host=192.168.0.176 -f : -r '\n' -q id,type
3:ndbd
5:ndb_mgmd
```

In this example, we also used the short form `-q` to determine the attributes to be queried.

Similarly, you can limit results to a node with a specific ID using the `--id [2199]` or `--nodeid [2199]` option.

17.4.7. ndb_cpcd — Automate Testing for NDB Development

A utility having this name was formerly part of an internal automated test framework used in testing and debugging MySQL Cluster. It was deprecated in MySQL Cluster NDB 7.0, and removed from MySQL Cluster distributions provided by Oracle beginning with MySQL Cluster NDB 7.2.1.

17.4.8. ndb_delete_all — Delete All Rows from an NDB Table

`ndb_delete_all` deletes all rows from the given NDB table. In some cases, this can be much faster than `DELETE` or even `TRUNCATE TABLE`.

Usage

```
ndb_delete_all -c connect_string tbl_name -d db_name
```

This deletes all rows from the table named *tbl_name* in the database named *db_name*. It is exactly equivalent to executing `TRUNCATE db_name.tbl_name` in MySQL.

The following table includes options that are specific to `ndb_delete_all`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_delete_all`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.15. `ndb_delete_all` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=dbname,</code> <code>-d</code>	Name of the database in which the table is found	All MySQL 5.5 based releases
<code>--transactional [2204],</code> <code>-t [2204]</code>	Perform the delete in a single transaction (may run out of operations)	All MySQL 5.5 based releases
<code>--tupscan</code>	Run tup scan	All MySQL 5.5 based releases
<code>--diskscan</code>	Run disk scan	All MySQL 5.5 based releases

- `--transactional, -t`

Use of this option causes the delete operation to be performed as a single transaction.



Warning

With very large tables, using this option may cause the number of operations available to the cluster to be exceeded.

17.4.9. `ndb_desc` — Describe NDB Tables

`ndb_desc` provides a detailed description of one or more NDB tables.

Usage

```
ndb_desc -c connect_string tbl_name -d db_name [-p]
```

Sample Output

MySQL table creation and population statements:

```
USE test;

CREATE TABLE fish (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  length_mm INT(11) NOT NULL,
  weight_gm INT(11) NOT NULL,

  PRIMARY KEY pk (id),
  UNIQUE KEY uk (name)
) ENGINE=NDB;

INSERT INTO fish VALUES
```

```
(', 'guppy', 35, 2), (', 'tuna', 2500, 150000),
(', 'shark', 3000, 110000), (', 'manta ray', 1500, 50000),
(', 'grouper', 900, 125000), (', 'puffer', 250, 2500);
```

Output from `ndb_desc`:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 2
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 311
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
length_mm Int NOT NULL AT=FIXED ST=MEMORY
weight_gm Int NOT NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk$unique(name) - UniqueHashIndex
uk(name) - OrderedIndex

-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory  ...
0          2          2             32768              ...
1          4          4             32768              ...

... Frag var sized memory  Extent_space  Free extent_space
... 32768                  0             0
... 32768                  0             0

NDBT_ProgramExit: 0 - OK
```

Information about multiple tables can be obtained in a single invocation of `ndb_desc` by using their names, separated by spaces. All of the tables must be in the same database.

The `Version` column in the output contains the table's schema object version. For information about interpreting this value, see [NDB Schema Object Versions](#).

The `Extent_space` and `Free extent_space` columns are applicable only to `NDB` tables having columns on disk; for tables having only in-memory columns, these columns always contain the value `0`.

To illustrate their use, we modify the previous example. First, we must create the necessary Disk Data objects, as shown here:

```
CREATE LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_1.log'
  INITIAL_SIZE 16M
  UNDO_BUFFER_SIZE 2M
  ENGINE NDB;

ALTER LOGFILE GROUP lg_1
```

```

ADD UNDOFILE 'undo_2.log'
INITIAL_SIZE 12M
ENGINE NDB;

CREATE TABLESPACE ts_1
ADD DATAFILE 'data_1.dat'
USE LOGFILE GROUP lg_1
INITIAL_SIZE 32M
ENGINE NDB;

ALTER TABLESPACE ts_1
ADD DATAFILE 'data_2.dat'
INITIAL_SIZE 48M
ENGINE NDB;

```

(For more information on the statements just shown and the objects created by them, see [Section 17.5.12.1, “MySQL Cluster Disk Data Objects”](#), as well as [Section 13.1.14, “CREATE LOGFILE GROUP Syntax”](#), and [Section 13.1.18, “CREATE TABLESPACE Syntax”](#).)

Now we can create and populate a version of the `fish` table that stores 2 of its columns on disk (deleting the previous version of the table first, if it already exists):

```

CREATE TABLE fish (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  length_mm INT(11) NOT NULL,
  weight_gm INT(11) NOT NULL,

  PRIMARY KEY pk (id),
  UNIQUE KEY uk (name)
) TABLESPACE ts_1 STORAGE DISK
ENGINE=NDB;

INSERT INTO fish VALUES
  ('','guppy', 35, 2), ('','tuna', 2500, 150000),
  ('','shark', 3000, 110000), ('','manta ray', 1500, 50000),
  ('','grouper', 900, 125000), ('','puffer', 250, 2500);

```

When run against this version of the table, `ndb_desc` displays the following output:

```

shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 3
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 321
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
length_mm Int NOT NULL AT=FIXED ST=DISK
weight_gm Int NOT NULL AT=FIXED ST=DISK

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

```

```

uk$unique(name) - UniqueHashIndex
uk(name) - OrderedIndex
-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory ...
0          2          2            32768             ...
1          4          4            32768             ...

... Frag var sized memory  Extent_space  Free extent_space
... 32768                 0             0
... 32768                 0             0

NDBT_ProgramExit: 0 - OK

```

This means that 1048576 bytes are allocated from the tablespace for this table on each partition, of which 1044440 bytes remain free for additional storage. In other words, $1048576 - 1044440 = 4136$ bytes per partition is currently being used to store the data from this table's disk-based columns. The number of bytes shown as `Free extent_space` is available for storing on-disk column data from the `fish` table only; for this reason, it is not visible when selecting from the `INFORMATION_SCHEMA.FILES` table.

The following table includes options that are specific to `ndb_desc`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_desc`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.16. `ndb_desc` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=dbname,</code> <code>-d</code>	Name of database containing table	All MySQL 5.5 based releases
<code>--unqualified,</code> <code>-u</code>	Use unqualified table names	All MySQL 5.5 based releases
<code>--extra-partition-info [2207],</code> <code>-p [2207]</code>	Display information about partitions	All MySQL 5.5 based releases
<code>--retries=#,</code> <code>-r</code>	Number of times to retry the connection (once per second)	All MySQL 5.5 based releases
<code>--blob-info [2207],</code> <code>-b [2207]</code>	Include partition information for BLOB tables in output. Requires that the <code>-p</code> option also be used	All MySQL 5.5 based releases
<code>--extra-node-info [2207],</code> <code>-n [2207]</code>	Include partition-to-data-node mappings in output. Requires that the <code>-p</code> option also be used	All MySQL 5.5 based releases

- `--extra-partition-info, -p`

Print additional information about the table's partitions.

- `--blob-info, -b`

Include information about subordinate `BLOB` and `TEXT` columns.

Use of this option also requires the use of the `--extra-partition-info [2207] (-p)` option.

- `--extra-node-info, -n`

Include information about the mappings between table partitions and the data nodes upon which they reside. This information can be useful for verifying distribution awareness mechanisms and supporting more efficient application access to the data stored in MySQL Cluster.

Use of this option also requires the use of the `--extra-partition-info` [2207] (`-p`) option.

17.4.10. ndb_drop_index — Drop Index from an NDB Table

`ndb_drop_index` drops the specified index from an NDB table. *It is recommended that you use this utility only as an example for writing NDB API applications*—see the Warning later in this section for details.

Usage

```
ndb_drop_index -c connect_string table_name index -d db_name
```

The statement shown above drops the index named `index` from the `table` in the `database`.

The following table includes options that are specific to `ndb_drop_index`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_drop_index`), see Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.17. ndb_drop_index Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=dbname,</code> <code>-d</code>	Name of the database in which the table is found	All MySQL 5.5 based releases



Warning

Operations performed on Cluster table indexes using the NDB API are not visible to MySQL and make the table unusable by a MySQL server. If you use this program to drop an index, then try to access the table from an SQL node, an error results, as shown here:

```
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

shell> ./mysql -u jon -p ctest1
Enter password: *****
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.5.29-ndb-7.2.10

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW TABLES;
+-----+
| Tables_in_ctest1 |
+-----+
| a                 |
| bt1               |
| bt2               |
| dogs              |
| employees         |
+-----+
```

```
| fish |
+-----+
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): Got error 4243 'Index not found' from NDBCLUSTER
```

In such a case, your *only* option for making the table available to MySQL again is to drop the table and re-create it. You can use either the SQL statement `DROP TABLE` or the `ndb_drop_table` utility (see [Section 17.4.11, “ndb_drop_table — Drop an NDB Table”](#)) to drop the table.

17.4.11. ndb_drop_table — Drop an NDB Table

`ndb_drop_table` drops the specified NDB table. (If you try to use this on a table created with a storage engine other than NDB, the attempt fails with the error `723: No such table exists`.) This operation is extremely fast; in some cases, it can be an order of magnitude faster than using a MySQL `DROP TABLE` statement on an NDB table.

Usage

```
ndb_drop_table -c connect_string tbl_name -d db_name
```

The following table includes options that are specific to `ndb_drop_table`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_drop_table`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.18. ndb_drop_table Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=dbname,</code> <code>-d</code>	Name of the database in which the table is found	All MySQL 5.5 based releases

17.4.12. ndb_error_reporter — NDB Error-Reporting Utility

`ndb_error_reporter` creates an archive from data node and management node log files that can be used to help diagnose bugs or other problems with a cluster. *It is highly recommended that you make use of this utility when filing reports of bugs in MySQL Cluster.*

The following table includes command options specific to the MySQL Cluster program `ndb_error_reporter`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_error_reporter`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.19. ndb_error_reporter Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--fs [2210]</code>	Include file system data in error report; can use a large amount of disk space	All MySQL 5.5 based releases

Usage

```
ndb_error_reporter path/to/config-file [username] [--fs]
```

This utility is intended for use on a management node host, and requires the path to the management host configuration file (`config.ini`). Optionally, you can supply the name of a user that is able to access

the cluster's data nodes using SSH, to copy the data node log files. `ndb_error_reporter` then includes all of these files in archive that is created in the same directory in which it is run. The archive is named `ndb_error_report_YYYYMMDDHHMMSS.tar.bz2`, where `YYYYMMDDHHMMSS` is a datetime string.

If the `--fs` is used, then the data node file systems are also copied to the management host and included in the archive that is produced by this script. As data node file systems can be extremely large even after being compressed, we ask that you please do *not* send archives created using this option to Oracle unless you are specifically requested to do so.

Command-Line Format	<code>--fs</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

17.4.13. ndb_index_stat — NDB Index Statistics Utility

`ndb_index_stat ...`

Usage

To obtain basic index statistics about a given NDB table, invoke `ndb_index_stat` as shown here, with the name of the table as the first argument and the name of the database containing this table specified immediately following it, using the `--database [2211] (-d)` option:

```
ndb_index_stat table -d database
```

In this example, we use `ndb_index_stat` to obtain such information about an NDB table named `mytable` in the `test` database:

```
shell> ndb_index_stat -d test mytable
table:mytable index:PRIMARY fragCount:4
sampleVersion:2 loadTime:1336751773 sampleCount:0 keyBytes:0
query cache: valid:1 sampleCount:0 totalBytes:0
times in ms: save: 12.380 sort: 0.001

NDBT_ProgramExit: 0 - OK
```

The `--verbose [2214]` option provides some additional output, as shown here:

```
shell> ndb_index_stat -d test mytable --verbose
random seed 1337010518
connected
loop 1 of 1
table:mytable index:PRIMARY fragCount:4
sampleVersion:2 loadTime:1336751773 sampleCount:0 keyBytes:0
read stats
query cache created
query cache: valid:1 sampleCount:0 totalBytes:0
times in ms: save: 20.766 sort: 0.001
disconnected

NDBT_ProgramExit: 0 - OK

shell>
```

Options

The following table includes options that are specific to the MySQL Cluster `ndb_index_stat` utility. Additional descriptions are listed following the table. For options common to most MySQL Cluster

programs (including `ndb_index_stat`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.20. `ndb_index_stat` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=name,</code> <code>-d</code>	Name of the database containing the table.	All MySQL 5.5 based releases
<code>--delete</code>	Delete index statistics for the given table, stopping any auto-update previously configured.	All MySQL 5.5 based releases
<code>--update</code>	Update index statistics for the given table, restarting any auto-update previously configured.	All MySQL 5.5 based releases
<code>--dump</code>	Print the query cache.	All MySQL 5.5 based releases
<code>--query=#</code>	Perform a number of random range queries on first key attr (must be int unsigned).	All MySQL 5.5 based releases
<code>--sys-drop</code>	Drop any statistics tables and events in NDB kernel (all statistics are lost)	All MySQL 5.5 based releases
<code>--sys-create</code>	Create all statistics tables and events in NDB kernel, if none of them already exist	All MySQL 5.5 based releases
<code>--sys-create-if-not-exist</code>	Create any statistics tables and events in NDB kernel that do not already exist.	All MySQL 5.5 based releases
<code>--sys-create-if-not-valid</code>	Create any statistics tables or events that do not already exist in the NDB kernel. after dropping any that are invalid.	All MySQL 5.5 based releases
<code>--sys-check</code>	Verify that NDB system index statistics and event tables exist.	All MySQL 5.5 based releases
<code>--sys-skip-tables</code>	Do not apply sys-* options to tables.	All MySQL 5.5 based releases
<code>--sys-skip-events</code>	Do not apply sys-* options to events.	All MySQL 5.5 based releases
<code>--verbose,</code> <code>-v</code>	Turn on verbose output	All MySQL 5.5 based releases
<code>--loops=#</code>	Set the number of times to perform a given command. Default is 0.	All MySQL 5.5 based releases

`ndb_index_stat` statistics options. The following options are used to generate index statistics. They work with a given table and database. They cannot be mixed with `ndb_index_stat` [system options](#).

- `--database=name, -d name`

Command-Line Format	<code>--database=name</code>	
	<code>-d</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>[none]</code>
	Range	<code>..</code>

The name of the database that contains the table being queried.

- `--delete`

Command-Line Format	<code>--delete</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Delete the index statistics for the given table, stopping any auto-update that was previously configured.

- `--update`

Command-Line Format	<code>--update</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Update the index statistics for the given table, and restart any auto-update that previously configured.

- `--dump`

Command-Line Format	<code>--dump</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Dump the contents of the query cache.

- `--query=#`

Command-Line Format	<code>--query=#</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. MAX_INT</code>

Perform random range queries on first key attribute (must be int unsigned).

ndb_index_stat system options. The following options are used to generate and update the statistics tables in the NDB kernel. They cannot be mixed with [ndb_index_stat statistics options](#).

- `--sys-drop`

Command-Line Format	<code>--sys-drop</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Drop all statistics tables and events in the NDB kernel. *This causes all statistics to be lost.*

- `--sys-create`

Command-Line Format	<code>--sys-create</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Create all statistics tables and events in the NDB kernel. This works only if none of them exist previously.

- `sys-create-if-not-exist`

Command-Line Format	<code>--sys-create-if-not-exist</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Create any NDB system statistics tables or events (or both) that do not already exist when the program is invoked.

- `--sys-create-if-not-valid`

Command-Line Format	<code>--sys-create-if-not-valid</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Create any NDB system statistics tables or events that do not already exist, after dropping any that are invalid.

- `--sys-check`

Command-Line Format	<code>--sys-check</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Verify that all required system statistics tables and events exist in the NDB kernel.

- `--sys-skip-tables`

Command-Line Format	<code>--sys-skip-tables</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Do not apply any `--sys-*` options to any statistics tables.

- `--sys-skip-events`

Command-Line Format	<code>--sys-skip-events</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Do not apply any `--sys-*` options to any events.

- `--verbose`

Command-Line Format	<code>--verbose</code>	
	<code>-v</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>true</code>
	Range	<code>..</code>

Turn on verbose output.

- `--loops=#`

Command-Line Format	<code>--loops=#</code>	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>0</code>
	Range	<code>0 .. MAX_INT</code>

Repeat commands this number of times (for use in testing).

17.4.14. `ndb_print_backup_file` — Print NDB Backup File Contents

`ndb_print_backup_file` obtains diagnostic information from a cluster backup file.

Usage

```
ndb_print_backup_file file_name
```

file_name is the name of a cluster backup file. This can be any of the files (`.Data`, `.ctl`, or `.log` file) found in a cluster backup directory. These files are found in the data node's backup directory under the subdirectory `BACKUP-#`, where `#` is the sequence number for the backup. For more information about cluster backup files and their contents, see [Section 17.5.3.1, “MySQL Cluster Backup Concepts”](#).

Like `ndb_print_schema_file` and `ndb_print_sys_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_backup_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

17.4.15. `ndb_print_schema_file` — Print NDB Schema File Contents

`ndb_print_schema_file` obtains diagnostic information from a cluster schema file.

Usage

```
ndb_print_schema_file file_name
```

file_name is the name of a cluster schema file. For more information about cluster schema files, see [MySQL Cluster Data Node FileSystemDir Files](#).

Like `ndb_print_backup_file` and `ndb_print_sys_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_schema_backup_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

17.4.16. `ndb_print_sys_file` — Print NDB System File Contents

`ndb_print_sys_file` obtains diagnostic information from a MySQL Cluster system file.

Usage

```
ndb_print_sys_file file_name
```

file_name is the name of a cluster system file (sysfile). Cluster system files are located in a data node's data directory (`DataDir` [\[2042\]](#)); the path under this directory to system files matches the pattern `ndb_#_fs/D#/DBDIH/P#.sysfile`. In each case, the `#` represents a number (not necessarily the same number). For more information, see [MySQL Cluster Data Node FileSystemDir Files](#).

Like `ndb_print_backup_file` and `ndb_print_schema_file` (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_backup_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

17.4.17. ndbd_redo_log_reader — Check and Print Content of Cluster Redo Log

Reads a redo log file, checking it for errors, printing its contents in a human-readable format, or both. `ndbd_redo_log_reader` is intended for use primarily by MySQL Cluster developers and Support personnel in debugging and diagnosing problems.

This utility remains under development, and its syntax and behavior are subject to change in future MySQL Cluster releases.

The C++ source files for `ndbd_redo_log_reader` can be found in the directory `/storage/ndb/src/kernel/blocks/dblqh/redoLogReader`.

The following table includes options that are specific to the MySQL Cluster program `ndbd_redo_log_reader`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndbd_redo_log_reader`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.21. ndbd_redo_log_reader Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>-noprint</code> [2216]	Do not print records	All MySQL 5.5 based releases
<code>-nocheck</code> [2217]	Do not check records for errors	All MySQL 5.5 based releases

Usage

```
ndbd_redo_log_reader file_name [options]
```

`file_name` is the name of a cluster redo log file. redo log files are located in the numbered directories under the data node's data directory (`DataDir` [2042]); the path under this directory to the redo log files matches the pattern `ndb_#_fs/D#/LCP/#/T#F#.Data`. In each case, the # represents a number (not necessarily the same number). For more information, see [MySQL Cluster Data Node FileSystemDir Files](#).

The name of the file to be read may be followed by one or more of the options listed here:

- | | | |
|----------------------------|-------------------------|----------------------|
| Command-Line Format | <code>-noprint</code> | |
| | Permitted Values | |
| | Type | <code>boolean</code> |
| | Default | <code>FALSE</code> |

`-noprint`: Do not print the contents of the log file.

• Command-Line Format	<code>-nocheck</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

`-nocheck`: Do not check the log file for errors.

Like `ndb_print_backup_file` and `ndb_print_schema_file` (and unlike most of the NDB utilities that are intended to be run on a management server host or to connect to a management server) `ndbd_redo_log_reader` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

17.4.18. ndb_restore — Restore a MySQL Cluster Backup

The cluster restoration program is implemented as a separate command-line utility `ndb_restore`, which can normally be found in the MySQL `bin` directory. This program reads the files created as a result of the backup and inserts the stored information into the database.

`ndb_restore` must be executed once for each of the backup files that were created by the `START BACKUP` command used to create the backup (see [Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”](#)). This is equal to the number of data nodes in the cluster at the time that the backup was created.



Note

Before using `ndb_restore`, it is recommended that the cluster be running in single user mode, unless you are restoring multiple data nodes in parallel. See [Section 17.5.8, “MySQL Cluster Single User Mode”](#), for more information.

The following table includes options that are specific to the MySQL Cluster native backup restoration program `ndb_restore`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_restore`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.22. ndb_restore Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--connect [2220], -c [2220]</code>	Same as connectstring	All MySQL 5.5 based releases
<code>--nodeid=# [2220], -n [2220]</code>	Back up files from node with this ID	All MySQL 5.5 based releases
<code>--backupid=# [2221], -b [2221]</code>	Restore from the backup with the given ID	All MySQL 5.5 based releases
<code>--restore_data [2222]</code>	Restore table data and logs into NDB Cluster using the NDB API	All MySQL 5.5 based releases
<code>--restore_meta [2222], -m [2222]</code>	Restore metadata to NDB Cluster using the NDB API	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--no-upgrade [2223],</code> <code>-u [2223]</code>	Do not upgrade array type for varsize attributes which do not already resize VAR data, and do not change column attributes	All MySQL 5.5 based releases
<code>--promote-attributes [2221],</code> <code>-A [2221]</code>	Allow attributes to be promoted when restoring data from backup	All MySQL 5.5 based releases
<code>--preserve-trailing-spaces [2221],</code> <code>-P [2221]</code>	Allow preservation of trailing spaces (including padding) when promoting fixed-width string types to variable-width types	All MySQL 5.5 based releases
<code>--no-restore-disk-objects [2225],</code> <code>-d [2225]</code>	Do not restore objects relating to Disk Data	All MySQL 5.5 based releases
<code>--restore_epoch [2222],</code> <code>-e [2222]</code>	Restore epoch info into the status table. Convenient on a MySQL Cluster replication slave for starting replication. The row in <code>mysql.ndb_apply_status</code> with id 0 will be updated/inserted.	All MySQL 5.5 based releases
<code>--skip-table-check [2220],</code> <code>-s [2220]</code>	Skip table structure check during restoring of data	All MySQL 5.5 based releases
<code>--parallelism=# [2225],</code> <code>-p [2225]</code>	Number of parallel transactions to use while restoring data	All MySQL 5.5 based releases
<code>--print [2224]</code>	Print metadata, data and log to stdout (equivalent to <code>--print_meta --print_data --print_log</code>)	All MySQL 5.5 based releases
<code>--print_meta [2224]</code>	Print metadata to stdout	All MySQL 5.5 based releases
<code>--print_data [2223]</code>	Print data to stdout	All MySQL 5.5 based releases
<code>--print_log [2224]</code>	Print to stdout	All MySQL 5.5 based releases
<code>--backup_path=path [2222]</code>	Path to backup files directory	All MySQL 5.5 based releases
<code>--dont_ignore_systab_0 [2224],</code> <code>-f [2224]</code>	Do not ignore system table during restore. Experimental only; not for production use	All MySQL 5.5 based releases
<code>--ndb-nodegroup-map=map [2225],</code> <code>-z [2225]</code>	Nodegroup map for NDBCLUSTER storage engine. Syntax: list of (source_nodegroup, destination_nodegroup)	All MySQL 5.5 based releases
<code>--fields-enclosed-by=char [2223]</code>	Fields are enclosed with the indicated character	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--fields-terminated-by=char [2223]</code>	Fields are terminated by the indicated character	All MySQL 5.5 based releases
<code>--fields-optionally-enclosed-by [2223]</code>	Fields are optionally enclosed with the indicated character	All MySQL 5.5 based releases
<code>--lines-terminated-by=char [2224]</code>	Lines are terminated by the indicated character	All MySQL 5.5 based releases
<code>--hex [2224]</code>	Print binary types in hexadecimal format	All MySQL 5.5 based releases
<code>--tab=path [2223], -T [2223]</code>	Creates a tab-separated .txt file for each table in the given path	All MySQL 5.5 based releases
<code>--append [2224]</code>	Append data to a tab-delimited file	All MySQL 5.5 based releases
<code>--progress-frequency=# [2225]</code>	Print status of restoration each given number of seconds	All MySQL 5.5 based releases
<code>--no-binlog [2225]</code>	If a mysqld is connected and using binary logging, do not log the restored data	All MySQL 5.5 based releases
<code>--verbose=# [2225]</code>	Level of verbosity in output	All MySQL 5.5 based releases
<code>--include-databases=db-list [2225]</code>	List of one or more databases to restore (excludes those not named)	All MySQL 5.5 based releases
<code>--exclude-databases=db-list [2226]</code>	List of one or more databases to exclude (includes those not named)	All MySQL 5.5 based releases
<code>--include-tables=table-list [2225]</code>	List of one or more tables to restore (excludes those in same database that are not named); each table reference must include the database name	All MySQL 5.5 based releases
<code>--exclude-tables=table-list [2226]</code>	List of one or more tables to exclude (includes those in same database that are not named); each table reference must include the database name	All MySQL 5.5 based releases
<code>--exclude-missing-columns [2227]</code>	Causes columns from the backup version of a table that are missing from the version of the table in the database to be ignored.	All MySQL 5.5 based releases
<code>--disable-indexes [2228]</code>	Causes indexes from a backup to be ignored; may decrease time needed to restore data.	All MySQL 5.5 based releases
<code>--rebuild-indexes [2228]</code>	Causes multi-threaded ordered index rebuilding of indexes found in the backup.	All MySQL 5.5 based releases
<code>--skip-broken-objects [2228]</code>	Causes missing blob tables in the backup file to be ignored.	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--skip-unknown-objects</code> [2228]	Causes schema objects not recognized by <code>ndb_restore</code> to be ignored when restoring a backup made from a newer MySQL Cluster version to an older version.	All MySQL 5.5 based releases
<code>--rewrite-database=olddb,newdb</code> [2228]	Restores to a database with a different name than the original	All MySQL 5.5 based releases
<code>--lossy-conversions</code> [2221], <code>-L</code> [2221]	Allow lossy conversions of column values (type demotions or changes in sign) when restoring data from backup	All MySQL 5.5 based releases
<code>--restore-privilege-tables</code> [2222]	Restore MySQL privilege tables that were previously moved to NDB.	All MySQL 5.5 based releases

Typical options for this utility are shown here:

```
ndb_restore [-c connectstring] -n node_id [-m] -b backup_id \
-r --backup_path=/path/to/backup/files
```

The `-c` option is used to specify a connectstring which tells `ndb_restore` where to locate the cluster management server. (See [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#), for information on connectstrings.) If this option is not used, then `ndb_restore` attempts to connect to a management server on `localhost:1186`. This utility acts as a cluster API node, and so requires a free connection “slot” to connect to the cluster management server. This means that there must be at least one `[api]` or `[mysqld]` section that can be used by it in the cluster `config.ini` file. It is a good idea to keep at least one empty `[api]` or `[mysqld]` section in `config.ini` that is not being used for a MySQL server or other application for this reason (see [Section 17.3.2.7, “Defining SQL and Other API Nodes in a MySQL Cluster”](#)).

You can verify that `ndb_restore` is connected to the cluster by using the `SHOW` command in the `ndb_mgm` management client. You can also accomplish this from a system shell, as shown here:

```
shell> ndb_mgm -e "SHOW"
```

`-n` is used to specify the node ID of the data node on which the backups were taken.

The first time you run the `ndb_restore` restoration program, you also need to restore the metadata. In other words, you must re-create the database tables—this can be done by running it with the `-m` option. Note that the cluster should have an empty database when starting to restore a backup. (In other words, you should start `ndbd` with `--initial` [2178] prior to performing the restore. You should also remove manually any Disk Data files present in the data node's `DataDir` [2042].)

It is possible to restore data without restoring table metadata. The default behavior when doing this is for `ndb_restore` to fail with an error if table data do not match the table schema; this can be overridden using the `--skip-table-check` or `-s` option.

Some of the restrictions on mismatches in column definitions when restoring data using `ndb_restore` are relaxed; when one of these types of mismatches is encountered, `ndb_restore` does not stop with an error as it did previously, but rather accepts the data and inserts it into the target table while issuing a warning to the user that this is being done. This behavior occurs whether or not either of the options `--skip-table-check` or `--promote-attributes` [2221] is in use. These differences in column definitions are of the following types:

- Different `COLUMN_FORMAT` settings (`FIXED`, `DYNAMIC`, `DEFAULT`)

- Different `STORAGE` settings (`MEMORY`, `DISK`)
- Different default values
- Different distribution key settings

`ndb_restore` supports limited *attribute promotion* in much the same way that it is supported by MySQL replication; that is, data backed up from a column of a given type can generally be restored to a column using a “larger, similar” type. For example, data from a `CHAR(20)` column can be restored to a column declared as `VARCHAR(20)`, `VARCHAR(30)`, or `CHAR(30)`; data from a `MEDIUMINT` column can be restored to a column of type `INT` or `BIGINT`. See [Section 16.4.1.8.2, “Replication of Columns Having Different Data Types”](#), for a table of type conversions currently supported by attribute promotion.

Attribute promotion by `ndb_restore` must be enabled explicitly, as follows:

1. Prepare the table to which the backup is to be restored. `ndb_restore` cannot be used to re-create the table with a different definition from the original; this means that you must either create the table manually, or alter the columns which you wish to promote using `ALTER TABLE` after restoring the table metadata but before restoring the data.
2. Invoke `ndb_restore` with the `--promote-attributes [2221]` option (short form `-A`) when restoring the table data. Attribute promotion does not occur if this option is not used; instead, the restore operation fails with an error.

`--lossy-conversions, -L`

Command-Line Format	<code>--lossy-conversions</code>	
	<code>-L</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

This option is intended to complement the `--promote-attributes [2221]` option. Using `--lossy-conversions` allows lossy conversions of column values (type demotions or changes in sign) when restoring data from backup. With some exceptions, the rules governing demotion are the same as for MySQL replication; see [Section 16.4.1.8.2, “Replication of Columns Having Different Data Types”](#), for information about specific type conversions currently supported by attribute demotion.

`ndb_restore` reports any truncation of data that it performs during lossy conversions once per attribute and column.

The `--preserve-trailing-spaces` option (short form `-R`) causes trailing spaces to be preserved when promoting a fixed-width character data type to its variable-width equivalent—that is, when promoting a `CHAR` column value to `VARCHAR` or a `BINARY` column value to `VARBINARY`. Otherwise, any trailing spaces are dropped from such column values when they are inserted into the new columns.



Note

Although you can promote `CHAR` columns to `VARCHAR` and `BINARY` columns to `VARBINARY`, you cannot promote `VARCHAR` columns to `CHAR` or `VARBINARY` columns to `BINARY`.

The `-b` option is used to specify the ID or sequence number of the backup, and is the same number shown by the management client in the `Backup backup_id completed` message displayed upon completion of a backup. (See [Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”](#).)

**Important**

When restoring cluster backups, you must be sure to restore all data nodes from backups having the same backup ID. Using files from different backups will at best result in restoring the cluster to an inconsistent state, and may fail altogether.

`--restore_epoch` (short form: `-e`) adds (or restores) epoch information to the cluster replication status table. This is useful for starting replication on a MySQL Cluster replication slave. When this option is used, the row in the `mysql.ndb_apply_status` having 0 in the `id` column is updated if it already exists; such a row is inserted if it does not already exist. (See [Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”](#).)

`--restore_data`

This option causes `ndb_restore` to output NDB table data and logs.

`--restore_meta`

This option causes `ndb_restore` to print NDB table metadata.

`--restore-privilege-tables`

`ndb_restore` does not by default restore distributed MySQL privilege tables (MySQL Cluster NDB 7.2.0 and later). This option causes `ndb_restore` to restore the privilege tables.

This works only if the privilege tables were converted to NDB before the backup was taken. For more information, see [Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”](#).

`--backup_path`

The path to the backup directory is required; this is supplied to `ndb_restore` using the `--backup_path` option, and must include the subdirectory corresponding to the ID backup of the backup to be restored. For example, if the data node's `DataDir [2042]` is `/var/lib/mysql-cluster`, then the backup directory is `/var/lib/mysql-cluster/BACKUP`, and the backup files for the backup with the ID 3 can be found in `/var/lib/mysql-cluster/BACKUP/BACKUP-3`. The path may be absolute or relative to the directory in which the `ndb_restore` executable is located, and may be optionally prefixed with `backup_path=`.

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID 12, created in a cluster with two database nodes having the node IDs 2 and 3, is to be restored to a cluster with four nodes. Then `ndb_restore` must be run twice—once for each database node in the cluster where the backup was taken. However, `ndb_restore` cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version. See [Section 17.2.7, “Upgrading and Downgrading MySQL Cluster NDB 7.2”](#), for more information.

**Important**

It is not possible to restore a backup made from a newer version of MySQL Cluster using an older version of `ndb_restore`. You can restore a backup made from a newer version of MySQL to an older cluster, but you must use a copy of `ndb_restore` from the newer MySQL Cluster version to do so.

For example, to restore a cluster backup taken from a cluster running MySQL Cluster NDB 7.2.5 to a cluster running MySQL Cluster NDB 7.1.21, you must use the `ndb_restore` that comes with the MySQL Cluster NDB 7.2.5 distribution.

For more rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. That is, when restoring to multiple nodes in parallel, you must have an

[`api`] or [`mysqld`] section in the cluster `config.ini` file available for each concurrent `ndb_restore` process. However, the data files must always be applied before the logs.

`--no-upgrade`

When using `ndb_restore` to restore a backup, `VARCHAR` columns created using the old fixed format are resized and recreated using the variable-width format now employed. This behavior can be overridden using the `--no-upgrade` [2223] option (short form: `-u`) when running `ndb_restore`.

`--print_data`

The `--print_data` option causes `ndb_restore` to direct its output to `stdout`.

`TEXT` and `BLOB` column values are always truncated to the first 256 bytes in the output; this cannot currently be overridden when using `--print_data`.

Several additional options are available for use with the `--print_data` option in generating data dumps, either to `stdout`, or to a file. These are similar to some of the options used with `mysqldump`, and are shown in the following list:

- `--tab, -T`

Command-Line Format	<code>--tab=path</code>
	<code>-T</code>

This option causes `--print_data` [2223] to create dump files, one per table, each named `tbl_name.txt`. It requires as its argument the path to the directory where the files should be saved; use `.` for the current directory.

- `--fields-enclosed-by=string`

Command-Line Format	<code>--fields-enclosed-by=char</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

Each column values are enclosed by the string passed to this option (regardless of data type; see next item).

- `--fields-optionally-enclosed-by=string`

Command-Line Format	<code>--fields-optionally-enclosed-by</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	

The string passed to this option is used to enclose column values containing character data (such as `CHAR`, `VARCHAR`, `BINARY`, `TEXT`, or `ENUM`).

- `--fields-terminated-by=string`

Command-Line Format	<code>--fields-terminated-by=char</code>	
	Permitted Values	

	Type	string
	Default	\t (tab)

The string passed to this option is used to separate column values. The default value is a tab character (\t).

- `--hex`

Command-Line Format	<code>--hex</code>
----------------------------	--------------------

If this option is used, all binary values are output in hexadecimal format.

- `--fields-terminated-by=string`

Command-Line Format	<code>--fields-terminated-by=char</code>	
	Permitted Values	
	Type	string
	Default	\t (tab)

This option specifies the string used to end each line of output. The default is a linefeed character (\n).

- `--append`

Command-Line Format	<code>--append</code>
----------------------------	-----------------------

When used with the `--tab` [2223] and `--print_data` [2223] options, this causes the data to be appended to any existing files having the same names.



Note

If a table has no explicit primary key, then the output generated when using the `--print_data` [2223] option includes the table's hidden primary key.

`--print_meta`

This option causes `ndb_restore` to print all metadata to `stdout`.

`--print_log`

The `--print_log` option causes `ndb_restore` to output its log to `stdout`.

`--print`

Causes `ndb_restore` to print all data, metadata, and logs to `stdout`. Equivalent to using the `--print_data` [2223], `--print_meta` [2224], and `--print_log` [2224] options together.



Note

Use of `--print` or any of the `--print_*` options is in effect performing a dry run. Including one or more of these options causes any output to be redirected to `stdout`; in such cases, `ndb_restore` makes no attempt to restore data or metadata to a MySQL Cluster.

`--dont_ignore_systab_0`

Normally, when restoring table data and metadata, `ndb_restore` ignores the copy of the `NDB` system table that is present in the backup. `--dont_ignore_systab_0` causes the system table to be restored. *This option is intended for experimental and development use only, and is not recommended in a production environment.*

`--ndb-nodegroup-map, -z`

This option can be used to restore a backup taken from one node group to a different node group. Its argument is a list of the form `source_node_group, target_node_group`.

`--no-binlog`

This option prevents any connected SQL nodes from writing data restored by `ndb_restore` to their binary logs.

`--no-restore-disk-objects, -d`

This option stops `ndb_restore` from restoring any MySQL Cluster Disk Data objects, such as tablespaces and log file groups; see [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#), for more information about these.

`--parallelism=#, -p`

Determines the maximum number of parallel transactions that `ndb_restore` tries to use. By default, this is 128; the minimum is 1, and the maximum is 1024.

`--progress-frequency=N`

Print a status report each `N` seconds while the backup is in progress. 0 (the default) causes no status reports to be printed. The maximum is 65535.

`--verbose=#`

Sets the level for the verbosity of the output. The minimum is 0; the maximum is 255. The default value is 1.

It is possible to restore only selected databases, or selected tables from a single database, using the syntax shown here:

```
ndb_restore other_options db_name,[db_name[,...] | tbl_name[,tbl_name][,...]]
```

In other words, you can specify either of the following to be restored:

- All tables from one or more databases
- One or more tables from a single database

`--include-databases=db_name[,db_name][,...]`

Command-Line Format	<code>--include-databases=db-list</code>	
	Permitted Values	
	Type	string
	Default	

`--include-tables=db_name.tbl_name[,db_name.tbl_name][,...]`

Command-Line Format	<code>--include-tables=table-list</code>	
	Permitted Values	

	Type	string
	Default	

Use the `--include-databases` [2225] option or the `--include-tables` option for restoring only specific databases or tables, respectively. `--include-databases` takes a comma-delimited list of databases to be restored. `--include-tables` takes a comma-delimited list of tables (in `database.table` format) to be restored.

When `--include-databases` or `--include-tables` is used, only those databases or tables named by the option are restored; all other databases and tables are excluded by `ndb_restore`, and are not restored.

The following table shows several invocations of `ndb_restore` using `--include-*` options (other options possibly required have been omitted for clarity), and the effects these have on restoring from a MySQL Cluster backup:

Option Used	Result
<code>--include-databases=db1</code>	Only tables in database <code>db1</code> are restored; all tables in all other databases are ignored
<code>--include-databases=db1,db2</code> (or <code>--include-databases=db1 --include-databases=db2</code>)	Only tables in databases <code>db1</code> and <code>db2</code> are restored; all tables in all other databases are ignored
<code>--include-tables=db1.t1</code>	Only table <code>t1</code> in database <code>db1</code> is restored; no other tables in <code>db1</code> or in any other database are restored
<code>--include-tables=db1.t2,db2.t1</code> (or <code>--include-tables=db1.t2 --include-tables=db2.t1</code>)	Only the table <code>t2</code> in database <code>db1</code> and the table <code>t1</code> in database <code>db2</code> are restored; no other tables in <code>db1</code> , <code>db2</code> , or any other database are restored

You can also use these two options together. For example, the following causes all tables in databases `db1` and `db2`, together with the tables `t1` and `t2` in database `db3`, to be restored (and no other databases or tables):

```
shell> ndb_restore [...] --include-databases=db1,db2 --include-tables=db3.t1,db3.t2
```

(Again we have omitted other, possibly required, options in the example just shown.)

```
--exclude-databases=db_name[,db_name][,...]
```

Command-Line Format	<code>--exclude-databases=db-list</code>	
	Permitted Values	
	Type	string
	Default	

```
--exclude-tables=db_name.tbl_name[,db_name.tbl_name][,...]
```

Command-Line Format	<code>--exclude-tables=table-list</code>	
	Permitted Values	
	Type	string
	Default	

It is possible to prevent one or more databases or tables from being restored using the `ndb_restore` options `--exclude-databases` and `--exclude-tables`. `--exclude-databases` takes a comma-

delimited list of one or more databases which should not be restored. `--exclude-tables` takes a comma-delimited list of one or more tables (using `database.table` format) which should not be restored.

When `--exclude-databases` or `--exclude-tables` is used, only those databases or tables named by the option are excluded; all other databases and tables are restored by `ndb_restore`.

This table shows several invocations of `ndb_restore` using `--exclude-*` options (other options possibly required have been omitted for clarity), and the effects these options have on restoring from a MySQL Cluster backup:

Option Used	Result
<code>--exclude-databases=db1</code>	All tables in all databases except <code>db1</code> are restored; no tables in <code>db1</code> are restored
<code>--exclude-databases=db1,db2</code> (or <code>--exclude-databases=db1 --exclude-databases=db2</code>)	All tables in all databases except <code>db1</code> and <code>db2</code> are restored; no tables in <code>db1</code> or <code>db2</code> are restored
<code>--exclude-tables=db1.t1</code>	All tables except <code>t1</code> in database <code>db1</code> are restored; all other tables in <code>db1</code> are restored; all tables in all other databases are restored
<code>--exclude-tables=db1.t2,db2.t1</code> (or <code>--exclude-tables=db1.t2 --exclude-tables=db2.t1</code>)	All tables in database <code>db1</code> except for <code>t2</code> and all tables in database <code>db2</code> except for table <code>t1</code> are restored; no other tables in <code>db1</code> or <code>db2</code> are restored; all tables in all other databases are restored

You can use these two options together. For example, the following causes all tables in all databases *except for* databases `db1` and `db2`, along with the tables `t1` and `t2` in database `db3`, *not* to be restored:

```
shell> ndb_restore [...] --exclude-databases=db1,db2 --exclude-tables=db3.t1,db3.t2
```

(Again, we have omitted other possibly necessary options in the interest of clarity and brevity from the example just shown.)

You can use `--include-*` and `--exclude-*` options together, subject to the following rules:

- The actions of all `--include-*` and `--exclude-*` options are cumulative.
- All `--include-*` and `--exclude-*` options are evaluated in the order passed to `ndb_restore`, from right to left.
- In the event of conflicting options, the first (rightmost) option takes precedence. In other words, the first option (going from right to left) that matches against a given database or table “wins”.

For example, the following set of options causes `ndb_restore` to restore all tables from database `db1` except `db1.t1`, while restoring no other tables from any other databases:

```
--include-databases=db1 --exclude-tables=db1.t1
```

However, reversing the order of the options just given simply causes all tables from database `db1` to be restored (including `db1.t1`, but no tables from any other database), because the `--include-databases` [2225] option, being farthest to the right, is the first match against database `db1` and thus takes precedence over any other option that matches `db1` or any tables in `db1`:

```
--exclude-tables=db1.t1 --include-databases=db1
```

```
--exclude-missing-columns
```

Command-Line Format	<code>--exclude-missing-columns</code>
----------------------------	--

It is also possible to restore only selected table columns using the `--exclude-missing-columns` option. When this option is used, `ndb_restore` ignores any columns missing from tables being restored as compared to the versions of those tables found in the backup. This option applies to all tables being restored. If you wish to apply this option only to selected tables or databases, you can use it in combination with one or more of the options described in the previous paragraph to do so, then restore data to the remaining tables using a complementary set of these options.

`--disable-indexes`

Command-Line Format	<code>--disable-indexes</code>
----------------------------	--------------------------------

Disable restoration of indexes during restoration of the data from a native NDB backup. Afterwards, you can restore indexes for all tables at once with multi-threaded building of indexes using `--rebuild-indexes` [2228], which should be faster than rebuilding indexes concurrently for very large tables.

`--rebuild-indexes`

Command-Line Format	<code>--rebuild-indexes</code>
----------------------------	--------------------------------

You can use this option with `ndb_restore` to cause multi-threaded rebuilding of the ordered indexes while restoring a native NDB backup.

`--skip-broken-objects`

Command-Line Format	<code>--skip-broken-objects</code>
----------------------------	------------------------------------

This option causes `ndb_restore` to ignore corrupt tables while reading a native NDB backup, and to continue restoring any remaining tables (that are not also corrupted). Currently, the `--skip-broken-objects` option works only in the case of missing blob parts tables.

`--skip-unknown-objects`

Command-Line Format	<code>--skip-unknown-objects</code>
----------------------------	-------------------------------------

This option causes `ndb_restore` to ignore any schema objects it does not recognize while reading a native NDB backup. This can be used for restoring a backup made from a cluster running MySQL Cluster NDB 7.2 to a cluster running MySQL Cluster NDB 7.1.

`--rewrite-database=old_dbname , new_dbname`

Command-Line Format	<code>--rewrite-database=olddb , newdb</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>none</code>

This option makes it possible to restore to a database having a different name from that used in the backup. For example, if a backup is made of a database named `products`, you can restore the data it contains to a database named `inventory`, use this option as shown here (omitting any other options that might be required):

```
shell> ndb_restore --rewrite-database=product , inventory
```

The option can be employed multiple times in a single invocation of `ndb_restore`. Thus it is possible to restore simultaneously from a database named `db1` to a database named `db2` and from

a database named `db3` to one named `db4` using `--rewrite-database=db1,db2 --rewrite-database=db3,db4`. Other `ndb_restore` options may be used between multiple occurrences of `--rewrite-database`.

In the event of conflicts between multiple `--rewrite-database` options, the last `--rewrite-database` option used, reading from left to right, is the one that takes effect. For example, if `--rewrite-database=db1,db2 --rewrite-database=db1,db3` is used, only `--rewrite-database=db1,db3` is honored, and `--rewrite-database=db1,db2` is ignored. It is also possible to restore from multiple databases to a single database, so that `--rewrite-database=db1,db3 --rewrite-database=db2,db3` restores all tables and data from databases `db1` and `db2` into database `db3`.



Important

When restoring from multiple backup databases into a single target database using `--rewrite-database`, no check is made for collisions between table or other object names, and the order in which rows are restored is not guaranteed. This means that it is possible in such cases for rows to be overwritten and updates to be lost.

Error reporting.

`ndb_restore` reports both temporary and permanent errors. In the case of temporary errors, it may be able to recover from them, and reports `Restore successful, but encountered temporary error, please look at configuration` in such cases.



Important

After using `ndb_restore` to initialize a MySQL Cluster for use in circular replication, binary logs on the SQL node acting as the replication slave are not automatically created, and you must cause them to be created manually. To cause the binary logs to be created, issue a `SHOW TABLES` statement on that SQL node before running `START SLAVE`. This is a known issue in MySQL Cluster.

17.4.19. ndb_select_all — Print Rows from an NDB Table

`ndb_select_all` prints all rows from an NDB table to `stdout`.

Usage

```
ndb_select_all -c connect_string tbl_name -d db_name [> file_name]
```

The following table includes options that are specific to the MySQL Cluster native backup restoration program `ndb_select_all`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_select_all`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.23. ndb_select_all Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=dbname [2230],</code> <code>-d [2230]</code>	Name of the database in which the table is found	All MySQL 5.5 based releases
<code>--parallelism=# [2230],</code> <code>-p [2230]</code>	Degree of parallelism	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--lock=# [2230],</code> <code>-l [2230]</code>	Lock type	All MySQL 5.5 based releases
<code>--order=index [2230],</code> <code>-o [2230]</code>	Sort resultset according to index whose name is supplied	All MySQL 5.5 based releases
<code>--descending [2231],</code> <code>-z [2231]</code>	Sort resultset in descending order (requires order flag)	All MySQL 5.5 based releases
<code>--header [2231],</code> <code>-h [2231]</code>	Print header (set to 0 FALSE to disable headers in output)	All MySQL 5.5 based releases
<code>--useHexFormat [2231],</code> <code>-x [2231]</code>	Output numbers in hexadecimal format	All MySQL 5.5 based releases
<code>--delimiter=char [2231],</code> <code>-D [2231]</code>	Set a column delimiter	All MySQL 5.5 based releases
<code>--disk [2231]</code>	Print disk references (useful only for Disk Data tables having nonindexed columns)	All MySQL 5.5 based releases
<code>--rowid [2231]</code>	Print rowid	All MySQL 5.5 based releases
<code>--gci [2231]</code>	Include GCI in output	All MySQL 5.5 based releases
<code>--tup [2231],</code> <code>-t [2231]</code>	Scan in tup order	All MySQL 5.5 based releases
<code>--nodata [2231]</code>	Do not print table column data	All MySQL 5.5 based releases

- `--database=dbname, -d dbname`

Name of the database in which the table is found. The default value is `TEST_DB`.

- `parallelism=#, -p #`

Specifies the degree of parallelism.

- `--lock=lock_type, -l lock_type`

Employs a lock when reading the table. Possible values for `lock_type` are:

- 0: Read lock
- 1: Read lock with hold
- 2: Exclusive read lock

There is no default value for this option.

- `--order=index_name, -o index_name`

Orders the output according to the index named *index_name*. Note that this is the name of an index, not of a column, and that the index must have been explicitly named when created.

- `--descending, -z`

Sorts the output in descending order. This option can be used only in conjunction with the `-o` (`--order` [2230]) option.

- `--header=FALSE`

Excludes column headers from the output.

- `--useHexFormat -x`

Causes all numeric values to be displayed in hexadecimal format. This does not affect the output of numerals contained in strings or datetime values.

- `--delimiter=character, -D character`

Causes the *character* to be used as a column delimiter. Only table data columns are separated by this delimiter.

The default delimiter is the tab character.

- `--disk`

Adds a disk reference column to the output. The column is nonempty only for Disk Data tables having nonindexed columns.

- `--rowid`

Adds a `ROWID` column providing information about the fragments in which rows are stored.

- `--gci`

Adds a column to the output showing the global checkpoint at which each row was last updated. See [Section 17.1, “MySQL Cluster Overview”](#), and [Section 17.5.6.2, “MySQL Cluster Log Events”](#), for more information about checkpoints.

- `--tupscan, -t`

Scan the table in the order of the tuples.

- `--nodata`

Causes any table data to be omitted.

Sample Output

Output from a MySQL `SELECT` statement:

```
mysql> SELECT * FROM ctest1.fish;
+----+-----+
| id | name  |
+----+-----+
| 3  | shark |
| 6  | puffer|
| 2  | tuna  |
| 4  | manta ray |
```

```
| 5 | grouper |
| 1 | guppy   |
+----+-----+
6 rows in set (0.04 sec)
```

Output from the equivalent invocation of `ndb_select_all`:

```
shell> ./ndb_select_all -c localhost fish -d ctest1
id      name
3       [shark]
6       [puffer]
2       [tuna]
4       [manta ray]
5       [grouper]
1       [guppy]
6 rows returned

NDBT_ProgramExit: 0 - OK
```

Note that all string values are enclosed by square brackets (“[...]”) in the output of `ndb_select_all`. For a further example, consider the table created and populated as shown here:

```
CREATE TABLE dogs (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(25) NOT NULL,
  breed VARCHAR(50) NOT NULL,
  PRIMARY KEY pk (id),
  KEY ix (name)
)
TABLESPACE ts STORAGE DISK
ENGINE=NDBCLUSTER;

INSERT INTO dogs VALUES
  ('', 'Lassie', 'collie'),
  ('', 'Scooby-Doo', 'Great Dane'),
  ('', 'Rin-Tin-Tin', 'Alsatian'),
  ('', 'Rosscoe', 'Mutt');
```

This demonstrates the use of several additional `ndb_select_all` options:

```
shell> ./ndb_select_all -d ctest1 dogs -o ix -z --gci --disk
GCI      id name          breed          DISK_REF
834461  2 [Scooby-Doo] [Great Dane] [ m_file_no: 0 m_page: 98 m_page_idx: 0 ]
834878  4 [Rosscoe]    [Mutt]        [ m_file_no: 0 m_page: 98 m_page_idx: 16 ]
834463  3 [Rin-Tin-Tin] [Alsatian]   [ m_file_no: 0 m_page: 34 m_page_idx: 0 ]
835657  1 [Lassie]     [Collie]     [ m_file_no: 0 m_page: 66 m_page_idx: 0 ]
4 rows returned

NDBT_ProgramExit: 0 - OK
```

17.4.20. ndb_select_count — Print Row Counts for NDB Tables

`ndb_select_count` prints the number of rows in one or more NDB tables. With a single table, the result is equivalent to that obtained by using the MySQL statement `SELECT COUNT(*) FROM tbl_name`.

Usage

```
ndb_select_count [-c connect_string] -ddb_name tbl_name[, tbl_name2[, ...]]
```

The following table includes options that are specific to the MySQL Cluster native backup restoration program `ndb_select_count`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_select_count`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.24. ndb_select_count Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=dbname,</code> <code>-d</code>	Name of the database in which the table is found	All MySQL 5.5 based releases
<code>--parallelism=#,</code> <code>-p</code>	Degree of parallelism	All MySQL 5.5 based releases
<code>--lock=#,</code> <code>-l</code>	Lock type	All MySQL 5.5 based releases

You can obtain row counts from multiple tables in the same database by listing the table names separated by spaces when invoking this command, as shown under **Sample Output**.

Sample Output

```
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs

NDBT_ProgramExit: 0 - OK
```

17.4.21. ndb_show_tables — Display List of NDB Tables

`ndb_show_tables` displays a list of all NDB database objects in the cluster. By default, this includes not only both user-created tables and NDB system tables, but NDB-specific indexes, internal triggers, and MySQL Cluster Disk Data objects as well.

The following table includes options that are specific to the MySQL Cluster native backup restoration program `ndb_show_tables`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_show_tables`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.25. ndb_show_tables Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=string [2234],</code> <code>-d [2234]</code>	Specifies the database in which the table is found	All MySQL 5.5 based releases
<code>--loops=# [2234],</code> <code>-l [2234]</code>	Number of times to repeat output	All MySQL 5.5 based releases
<code>--type=# [2234],</code> <code>-t [2234]</code>	Limit output to objects of this type	All MySQL 5.5 based releases
<code>--unqualified [2234],</code> <code>-u [2234]</code>	Do not qualify table names	All MySQL 5.5 based releases
<code>--parsable [2234],</code> <code>-p [2234]</code>	Return output suitable for MySQL LOAD DATA INFILE statement	All MySQL 5.5 based releases
<code>--show-temp-status [2234]</code>	Show table temporary flag	All MySQL 5.5 based releases

Usage

```
ndb_show_tables [-c connect_string]
```

- `--database, -d`

Specifies the name of the database in which the tables are found.

- `--loops, -l`

Specifies the number of times the utility should execute. This is 1 when this option is not specified, but if you do use the option, you must supply an integer argument for it.

- `--parsable, -p`

Using this option causes the output to be in a format suitable for use with `LOAD DATA INFILE`.

- `--show-temp-status`

If specified, this causes temporary tables to be displayed.

- `--type, -t`

Can be used to restrict the output to one type of object, specified by an integer type code as shown here:

- 1: System table
- 2: User-created table
- 3: Unique hash index

Any other value causes all `NDB` database objects to be listed (the default).

- `--unqualified, -u`

If specified, this causes unqualified object names to be displayed.



Note

Only user-created MySQL Cluster tables may be accessed from MySQL; system tables such as `SYSTAB_0` are not visible to `mysqld`. However, you can examine the contents of system tables using `NDB` API applications such as `ndb_select_all` (see [Section 17.4.19, “ndb_select_all — Print Rows from an NDB Table”](#)).

17.4.22. ndb_size.pl — NDBCLUSTER Size Requirement Estimator

This is a Perl script that can be used to estimate the amount of space that would be required by a MySQL database if it were converted to use the `NDBCLUSTER` storage engine. Unlike the other utilities discussed in this section, it does not require access to a MySQL Cluster (in fact, there is no reason for it to do so). However, it does need to access the MySQL server on which the database to be tested resides.

Requirements

- A running MySQL server. The server instance does not have to provide support for MySQL Cluster.
- A working installation of Perl.

- The `DBI` module, which can be obtained from CPAN if it is not already part of your Perl installation. (Many Linux and other operating system distributions provide their own packages for this library.)
- A MySQL user account having the necessary privileges. If you do not wish to use an existing account, then creating one using `GRANT USAGE ON db_name.*`—where `db_name` is the name of the database to be examined—is sufficient for this purpose.

`ndb_size.pl` can also be found in the MySQL sources in `storage/ndb/tools`.

The following table includes options that are specific to the MySQL Cluster program `ndb_size.pl`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_size.pl`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.26. `ndb_size.pl` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--database=dbname [2235]</code>	The database or databases to examine; accepts a comma-delimited list; the default is ALL (use all databases found on the server)	All MySQL 5.5 based releases
<code>--hostname[:port] [2235]</code>	Specify host and optional port as host[:port]	All MySQL 5.5 based releases
<code>--socket=file [2235]</code>	Specify a socket to connect to	All MySQL 5.5 based releases
<code>--user=string [2235]</code>	Specify a MySQL user name	All MySQL 5.5 based releases
<code>--password=string [2235]</code>	Specify a MySQL user password	All MySQL 5.5 based releases
<code>--format=string [2237]</code>	Set output format (text or HTML)	All MySQL 5.5 based releases
<code>--excludetables=tbl_list [2235]</code>	Skip any tables in a comma-separated list of tables	All MySQL 5.5 based releases
<code>--excludedbs=db_list [2235]</code>	Skip any databases in a comma-separated list of databases	All MySQL 5.5 based releases
<code>--savequeries=file [2237]</code>	Saves all queries to the database into the file specified	All MySQL 5.5 based releases
<code>--loadqueries=file [2237]</code>	Loads all queries from the file specified; does not connect to a database	All MySQL 5.5 based releases
<code>--real_table_name=table [2235]</code>	Designates a table to handle unique index size calculations	All MySQL 5.5 based releases

Usage

```
perl ndb_size.pl [--database={db_name|ALL}] [--hostname=host[:port]] [--socket=socket] \
  [--user=user] [--password=password] \
  [--help|-h] [--format={html|text}] \
  [--loadqueries=file_name] [--savequeries=file_name]
```

By default, this utility attempts to analyze all databases on the server. You can specify a single database using the `--database` option; the default behavior can be made explicit by using `ALL` for the name

of the database. You can also exclude one or more databases by using the `--excludedbs` option with a comma-separated list of the names of the databases to be skipped. Similarly, you can cause specific tables to be skipped by listing their names, separated by commas, following the optional `--excludetables` option. A host name can be specified using `--hostname`; the default is `localhost`. In MySQL Cluster NDB 7.2.6 and later, you can specify a port in addition to the host using `host:port` format for the value of `--hostname`. The default port number is 3306. If necessary, you can also specify a socket; the default is `/var/lib/mysql.sock`. A MySQL user name and password can be specified the corresponding options shown. It also possible to control the format of the output using the `--format` option; this can take either of the values `html` or `text`, with `text` being the default. An example of the text output is shown here:

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock
ndb_size.pl report for database: 'test' (1 tables)
-----
Connected to: DBI:mysql:host=localhost;mysql_socket=/tmp/mysql.sock

Including information for versions: 4.1, 5.0, 5.1

test.t1
-----

DataMemory for Columns (* means var sized DataMemory):
  Column Name      Type      Varsized  Key  4.1  5.0  5.1
  -----
  HIDDEN_NDB_PKEY  bigint    --      PRI  8    8    8
  c2                varchar(50) Y        52   52   4*
  c1                int(11)   --        4    4    4
  -----
  Fixed Size Columns DM/Row      64   64   12
  Varsize Columns DM/Row        0    0    4

DataMemory for Indexes:
  Index Name      Type      4.1      5.0      5.1
  -----
  PRIMARY         BTREE     16       16       16
  -----
  Total Index DM/Row      16       16       16

IndexMemory for Indexes:
  Index Name      4.1      5.0      5.1
  -----
  PRIMARY         33       16       16
  -----
  Indexes IM/Row      33       16       16

Summary (for THIS table):
  4.1      5.0      5.1
  -----
  Fixed Overhead DM/Row      12       12       16
  NULL Bytes/Row            4         4         4
  DataMemory/Row            96        96        48
  (Includes overhead, bitmap and indexes)

  Varsize Overhead DM/Row      0         0         8
  Varsize NULL Bytes/Row      0         0         4
  Avg Varsize DM/Row          0         0        16

  No. Rows                  0         0         0

  Rows/32kb DM Page        340        340       680
  Fixedsize DataMemory (KB) 0           0         0

  Rows/32kb Varsize DM Page  0           0       2040
  Varsize DataMemory (KB)  0           0         0

  Rows/8kb IM Page         248        512       512
  IndexMemory (KB)         0           0         0

Parameter Minimum Requirements
```

```
-----
* indicates greater than default
```

Parameter	Default	4.1	5.0	5.1
DataMemory (KB)	81920	0	0	0
NoOfOrderedIndexes	128	1	1	1
NoOfTables	128	1	1	1
IndexMemory (KB)	18432	0	0	0
NoOfUniqueHashIndexes	64	0	0	0
NoOfAttributes	1000	3	3	3
NoOfTriggers	768	5	5	5

For debugging purposes, the Perl arrays containing the queries run by this script can be read from the file specified using `can` can be saved to a file using `--savequeries`; a file containing such arrays to be read in during script execution can be specified using `--loadqueries`. Neither of these options has a default value.

To produce output in HTML format, use the `--format` option and redirect the output to a file, as shown here:

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock --format=html > ndb_size.html
```

(Without the redirection, the output is sent to `stdout`.)

The output from this script includes the following information:

- Minimum values for the `DataMemory` [2043], `IndexMemory` [2044], `MaxNoOfTables` [2054], `MaxNoOfAttributes` [2054], `MaxNoOfOrderedIndexes` [2055], `MaxNoOfUniqueHashIndexes` [2055], and `MaxNoOfTriggers` [2056] configuration parameters required to accommodate the tables analyzed.
- Memory requirements for all of the tables, attributes, ordered indexes, and unique hash indexes defined in the database.
- The `IndexMemory` [2044] and `DataMemory` [2043] required per table and table row.

17.4.23. `ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status

`ndb_waiter` repeatedly (each 100 milliseconds) prints out the status of all cluster data nodes until either the cluster reaches a given status or the `--timeout` [2238] limit is exceeded, then exits. By default, it waits for the cluster to achieve `STARTED` status, in which all nodes have started and connected to the cluster. This can be overridden using the `--no-contact` [2238] and `--not-started` [2238] options.

The node states reported by this utility are as follows:

- `NO_CONTACT`: The node cannot be contacted.
- `UNKNOWN`: The node can be contacted, but its status is not yet known. Usually, this means that the node has received a `START` or `RESTART` command from the management server, but has not yet acted on it.
- `NOT_STARTED`: The node has stopped, but remains in contact with the cluster. This is seen when restarting the node using the management client's `RESTART` command.
- `STARTING`: The node's `ndbd` process has started, but the node has not yet joined the cluster.
- `STARTED`: The node is operational, and has joined the cluster.
- `SHUTTING_DOWN`: The node is shutting down.
- `SINGLE_USER_MODE`: This is shown for all cluster data nodes when the cluster is in single user mode.

The following table includes options that are specific to the MySQL Cluster native backup restoration program `ndb_waiter`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_waiter`), see [Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”](#).

Table 17.27. `ndb_waiter` Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<code>--no-contact [2238], -n [2238]</code>	Wait for cluster to reach NO CONTACT state	All MySQL 5.5 based releases
<code>--not-started [2238]</code>	Wait for cluster to reach NOT STARTED state	All MySQL 5.5 based releases
<code>--single-user [2238]</code>	Wait for cluster to enter single user mode	All MySQL 5.5 based releases
<code>--timeout=# [2238], -t [2238]</code>	Wait this many seconds, then exit whether or not cluster has reached desired state; default is 2 minutes (120 seconds)	All MySQL 5.5 based releases
<code>--nowait-nodes=list [2238]</code>	List of nodes not to be waited for.	All MySQL 5.5 based releases
<code>--wait-nodes=list [2239], -w [2239]</code>	List of nodes to be waited for.	All MySQL 5.5 based releases

Usage

```
ndb_waiter [-c connect_string]
```

Additional Options

- `--no-contact, -n`

Instead of waiting for the `STARTED` state, `ndb_waiter` continues running until the cluster reaches `NO_CONTACT` status before exiting.

- `--not-started`

Instead of waiting for the `STARTED` state, `ndb_waiter` continues running until the cluster reaches `NOT_STARTED` status before exiting.

- `--timeout=seconds, -t seconds`

Time to wait. The program exits if the desired state is not achieved within this number of seconds. The default is 120 seconds (1200 reporting cycles).

- `--single-user`

The program waits for the cluster to enter single user mode.

- `--nowait-nodes=list`

When this option is used, `ndb_waiter` does not wait for the nodes whose IDs are listed. The list is comma-delimited; ranges can be indicated by dashes, as shown here:

```
shell> ndb_waiter --nowait-nodes=1,3,7-9
```



Important

Do *not* use this option together with the `--wait-nodes` [2239] option.

- `--wait-nodes=list, -w list`

When this option is used, `ndb_waiter` waits only for the nodes whose IDs are listed. The list is comma-delimited; ranges can be indicated by dashes, as shown here:

```
shell> ndb_waiter --wait-nodes=2,4-6,10
```



Important

Do *not* use this option together with the `--nowait-nodes` [2238] option.

Sample Output. Shown here is the output from `ndb_waiter` when run against a 4-node cluster in which two nodes have been shut down and then started again manually. Duplicate reports (indicated by "...") are omitted.

```
shell> ./ndb_waiter -c localhost

Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED
```

```

...
State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED

NDBT_ProgramExit: 0 - OK

```



Note

If no connectstring is specified, then `ndb_waiter` tries to connect to a management on `localhost`, and reports `Connecting to mgmsrv at (null)`.

17.4.24. Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs

All MySQL Cluster programs accept the options described in this section, with the following exceptions:

- `mysqld`
- `ndb_print_backup_file`
- `ndb_print_schema_file`
- `ndb_print_sys_file`

Users of earlier MySQL Cluster versions should note that some of these options have been changed to make them consistent with one another as well as with `mysqld`. You can use the `--help` option with any MySQL Cluster program—with the exception of `ndb_print_backup_file`, `ndb_print_schema_file`, and `ndb_print_sys_file`—to view a list of the options which the program supports.

The options in the following table are common to all MySQL Cluster executables (except those noted previously in this section).

Table 17.28. Common MySQL Cluster Program Options and Variables: MySQL 5.5

Format	Description	Added / Removed
<pre>--help [2241], --usage [2241], -? [2241]</pre>	Display help message and exit	All MySQL 5.5 based releases
<pre>--ndb- connectstring=connectstring [2241], --connect- string=connectstring [2241], -c [2241]</pre>	Set connectstring for connecting to <code>ndb_mgmd</code> . Syntax: <code>[nodeid=<id>;] [host=]<hostname>[:<port>]</code> . Overrides entries specified in <code>NDB_CONNECTSTRING</code> or <code>my.cnf</code> .	All MySQL 5.5 based releases

Format	Description	Added / Removed
<code>--ndb-mgmd-host=host[:port] [2242]</code>	Set the host (and port, if desired) for connecting to the management server	All MySQL 5.5 based releases
<code>--ndb-nodeid=# [2242]</code>	Set node id for this node	All MySQL 5.5 based releases
<code>--ndb-optimized-node-selection [2243]</code>	Select nodes for transactions in a more optimal way	All MySQL 5.5 based releases
<code>--character-sets-dir=path [2241]</code>	Directory where character sets are	All MySQL 5.5 based releases
<code>--debug=options [2242]</code>	Enable output from debug calls. Can be used only for versions compiled with debugging enabled	All MySQL 5.5 based releases
<code>--core-file [2242]</code>	Write core on errors (defaults to TRUE in debug builds)	All MySQL 5.5 based releases
<code>--version [2243], -V [2243]</code>	Output version information and exit	All MySQL 5.5 based releases

For options specific to individual MySQL Cluster programs, see [Section 17.4, “MySQL Cluster Programs”](#).

See [Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”](#), for `mysqld` options relating to MySQL Cluster.

- `--help`, `--usage`, `-?`

Command-Line Format	<code>--help</code>
	<code>--usage</code>
	<code>-?</code>

Prints a short list with descriptions of the available command options.

- `--character-sets-dir=name`

Command-Line Format	<code>--character-sets-dir=path</code>	
	Permitted Values	
	Type	<code>file name</code>
	Default	

Tells the program where to find character set information.

- `--ndb-connectstring=connect_string`, `--connect-string=connect_string`, `-c connect_string`

Command-Line Format	<code>--ndb-connectstring=connectstring</code>	
	<code>--connect-string=connectstring</code>	
	<code>-c</code>	
	Permitted Values	
	Type	<code>string</code>

	Default	<code>localhost:1186</code>
--	----------------	-----------------------------

This option takes a MySQL Cluster connectstring that specifies the management server for the application to connect to, as shown here:

```
shell> ndbd --ndb-connectstring="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

For more information, see [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#).

- `--core-file`

Command-Line Format	<code>--core-file</code>	
	Permitted Values	
	Type	<code>boolean</code>
	Default	<code>FALSE</code>

Write a core file if the program dies. The name and location of the core file are system-dependent. (For MySQL Cluster programs nodes running on Linux, the default location is the program's working directory—for a data node, this is the node's `DataDir` [2042].) For some systems, there may be restrictions or limitations; for example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation for detailed information.

If MySQL Cluster was built using the `--debug` option for `configure`, then `--core-file` is enabled by default. For regular builds, `--core-file` is disabled by default.

- `--debug[=options]`

Command-Line Format	<code>--debug=options</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>d:t:0,/tmp/ndb_restore.trace</code>

This option can be used only for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the `mysqld` process.

- `--ndb-mgmd-host=host[:port]`

Command-Line Format	<code>--ndb-mgmd-host=host[:port]</code>	
Option-File Format	<code>ndb-mgmd-host</code>	
	Permitted Values	
	Type	<code>string</code>
	Default	<code>localhost:1186</code>

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` [2241] option instead.

- `--ndb-nodeid=#`

Command-Line Format	<code>--ndb-nodeid=#</code>	
	Permitted Values	

	Type	numeric
	Default	0

Sets this node's MySQL Cluster node ID. *The range of permitted values depends on the node's type (data, management, or API) and the MySQL Cluster software version.* See [Section 17.1.6.2, "Limits and Differences of MySQL Cluster from Standard MySQL Limits"](#), for more information.

- `--ndb-optimized-node-selection`

Command-Line Format	<code>--ndb-optimized-node-selection</code>	
	Permitted Values	
	Type	boolean
	Default	TRUE

Optimize selection of nodes for transactions. Enabled by default.

- `--version, -V`

Command-Line Format	<code>--version</code>
	<code>-V</code>

Prints the MySQL Cluster version number of the executable. The version number is relevant because not all versions can be used together, and the MySQL Cluster startup process verifies that the versions of the binaries being used can co-exist in the same cluster. This is also important when performing an online (rolling) software upgrade or downgrade of MySQL Cluster.

See [Section 17.5.5, "Performing a Rolling Restart of a MySQL Cluster"](#)), for more information.

17.5. Management of MySQL Cluster

Managing a MySQL Cluster involves a number of tasks, the first of which is to configure and start MySQL Cluster. This is covered in [Section 17.3, "MySQL Cluster Configuration"](#), and [Section 17.4, "MySQL Cluster Programs"](#).

The next few sections cover the management of a running MySQL Cluster.

For information about security issues relating to management and deployment of a MySQL Cluster, see [Section 17.5.11, "MySQL Cluster Security Issues"](#).

There are essentially two methods of actively managing a running MySQL Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped, and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_node_id_cluster.log`; this is usually found in the management server's `DataDir` [2037] directory, but this location can be overridden using the `LogDestination` [2036] option. (Recall that `node_id` represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by `ndbd`. It is also possible to send cluster log entries to a Unix system log.

Some aspects of the cluster's operation can be also be monitored from an SQL node using the `SHOW ENGINE NDB STATUS` statement.

More detailed information about MySQL Cluster operations is available in real time through an SQL interface using the `ndbinfo` database. For more information, see [Section 17.5.10, "The ndbinfo MySQL Cluster Information Database"](#).

NDB statistics counters provide improved monitoring using the `mysql` client. These counters, implemented in the NDB kernel, relate to operations performed by or affecting `Ndb` objects, such as starting, closing, and aborting transactions; primary key and unique key operations; table, range, and pruned scans; blocked threads waiting for various operations to complete; and data and events sent and received by MySQL Cluster. The counters are incremented by the NDB kernel whenever NDB API calls are made or data is sent to or received by the data nodes.

`mysqld` exposes the NDB API statistics counters as system status variables, which can be identified from the prefix common to all of their names (`Ndb_api_`). The values of these variables can be read in the `mysql` client from the output of a `SHOW STATUS` statement, or by querying either the `SESSION_STATUS` table or the `GLOBAL_STATUS` table (in the `INFORMATION_SCHEMA` database). By comparing the values of the status variables before and after the execution of an SQL statement that acts on `NDB` tables, you can observe the actions taken on the NDB API level that correspond to this statement, which can be beneficial for monitoring and performance tuning of MySQL Cluster.

MySQL Enterprise Monitor can also be used to monitor MySQL Servers that are part of a MySQL Cluster deployment. MySQL Enterprise Monitor 2.3, added a MySQL Cluster advisor, including a set of graphs providing information on MySQL Cluster resources, and defining rules for alerts on key information from data nodes such as `DataMemory [2043]` usage. This information is made available to MySQL Enterprise Monitor 2.3 or later by any MySQL Server which is connected to the MySQL Cluster, using `ndbinfo`. The advisor could be run against a single MySQL Server in the Cluster, or against a pair in order to provide a higher level of availability for the monitoring service. For more information, see the [MySQL Enterprise Monitor 2.3 Manual](#).

MySQL Cluster Manager provides an advanced command-line interface that simplifies many otherwise complex MySQL Cluster management tasks, such as starting, stopping, or restarting a MySQL Cluster with a large number of nodes. The MySQL Cluster Manager client also supports commands for getting and setting the values of most node configuration parameters as well as `mysqld` server options and variables relating to MySQL Cluster. MySQL Cluster Manager 1.1 provides support for adding data nodes online. See the [MySQL Cluster Manager 1.1 User Manual](#), for more information.

17.5.1. Summary of MySQL Cluster Start Phases

This section provides a simplified outline of the steps involved when MySQL Cluster data nodes are started. More complete information can be found in [MySQL Cluster Start Phases](#), in the *NDB Internals Guide*.

These phases are the same as those reported in the output from the `node_id STATUS` command in the management client (see [Section 17.5.2, “Commands in the MySQL Cluster Management Client”](#)). These start phases are also reported in the `start_phase` column of the `ndbinfo.nodes` table.

Start types. There are several different startup types and modes, as shown in the following list:

- **Initial start.** The cluster starts with a clean file system on all data nodes. This occurs either when the cluster started for the very first time, or when all data nodes are restarted using the `--initial [2178]` option.



Note

Disk Data files are not removed when restarting a node using `--initial [2178]`.

- **System restart.** The cluster starts and reads data stored in the data nodes. This occurs when the cluster has been shut down after having been in use, when it is desired for the cluster to resume operations from the point where it left off.

- **Node restart.** This is the online restart of a cluster node while the cluster itself is running.
- **Initial node restart.** This is the same as a node restart, except that the node is reinitialized and started with a clean file system.

Setup and initialization (phase -1). Prior to startup, each data node (`ndbd` process) must be initialized. Initialization consists of the following steps:

1. Obtain a node ID
2. Fetch configuration data
3. Allocate ports to be used for inter-node communications
4. Allocate memory according to settings obtained from the configuration file

When a data node or SQL node first connects to the management node, it reserves a cluster node ID. To make sure that no other node allocates the same node ID, this ID is retained until the node has managed to connect to the cluster and at least one `ndbd` reports that this node is connected. This retention of the node ID is guarded by the connection between the node in question and `ndb_mgmd`.

After each data node has been initialized, the cluster startup process can proceed. The stages which the cluster goes through during this process are listed here:

- **Phase 0.** The `NDBFS` and `NDBCNTR` blocks start (see [NDB Kernel Blocks](#)). Data node file systems are cleared on those data nodes that were started with `--initial [2178]` option.
- **Phase 1.** In this stage, all remaining `NDB` kernel blocks are started. MySQL Cluster connections are set up, inter-block communications are established, and heartbeats are started. In the case of a node restart, API node connections are also checked.



Note

When one or more nodes hang in Phase 1 while the remaining node or nodes hang in Phase 2, this often indicates network problems. One possible cause of such issues is one or more cluster hosts having multiple network interfaces. Another common source of problems causing this condition is the blocking of TCP/IP ports needed for communications between cluster nodes. In the latter case, this is often due to a misconfigured firewall.

- **Phase 2.** The `NDBCNTR` kernel block checks the states of all existing nodes. The master node is chosen, and the cluster schema file is initialized.
- **Phase 3.** The `DBLQH` and `DBTC` kernel blocks set up communications between them. The startup type is determined; if this is a restart, the `DBDIH` block obtains permission to perform the restart.
- **Phase 4.** For an initial start or initial node restart, the redo log files are created. The number of these files is equal to `NoOfFragmentLogFiles [2051]`.

For a system restart:

- Read schema or schemas.
- Read data from the local checkpoint.
- Apply all redo information until the latest restorable global checkpoint has been reached.

For a node restart, find the tail of the redo log.

- **Phase 5.** Most of the database-related portion of a data node start is performed during this phase. For an initial start or system restart, a local checkpoint is executed, followed by a global checkpoint. Periodic checks of memory usage begin during this phase, and any required node takeovers are performed.
- **Phase 6.** In this phase, node groups are defined and set up.
- **Phase 7.** The arbitrator node is selected and begins to function. The next backup ID is set, as is the backup disk write speed. Nodes reaching this start phase are marked as `Started`. It is now possible for API nodes (including SQL nodes) to connect to the cluster.
- **Phase 8.** If this is a system restart, all indexes are rebuilt (by `DBDIH`).
- **Phase 9.** The node internal startup variables are reset.
- **Phase 100 (OBSOLETE).** Formerly, it was at this point during a node restart or initial node restart that API nodes could connect to the node and begin to receive events. Currently, this phase is empty.
- **Phase 101.** At this point in a node restart or initial node restart, event delivery is handed over to the node joining the cluster. The newly-joined node takes over responsibility for delivering its primary data to subscribers. This phase is also referred to as *SUMA handover phase*.

After this process is completed for an initial start or system restart, transaction handling is enabled. For a node restart or initial node restart, completion of the startup process means that the node may now act as a transaction coordinator.

17.5.2. Commands in the MySQL Cluster Management Client

In addition to the central configuration file, a cluster may also be controlled through a command-line interface available through the management client `ndb_mgm`. This is the primary administrative interface to a running cluster.

Commands for the event logs are given in [Section 17.5.6, “Event Reports Generated in MySQL Cluster”](#); commands for creating backups and restoring from them are provided in [Section 17.5.3, “Online Backup of MySQL Cluster”](#).

The management client has the following basic commands. In the listing that follows, `node_id` denotes either a database node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

- `HELP`

Displays information on all available commands.

- `SHOW`

Displays information on the cluster's status. Possible node status values include `UNKNOWN`, `NO_CONTACT`, `NOT_STARTED`, `STARTING`, `STARTED`, `SHUTTING_DOWN`, and `RESTARTING`. The output from this command also indicates when the cluster is in single user mode (status `SINGLE_USER_MODE`).

- `node_id START`

Brings online the data node identified by `node_id` (or all data nodes).

`ALL START` works on all data nodes only, and does not affect management nodes.



Important

To use this command to bring a data node online, the data node must have been started using `ndbd --nostart [2180]` or `ndbd -n`.

- `node_id STOP [-a] [-f]`

Stops the data or management node identified by `node_id`. Note that `ALL STOP` works to stop all data nodes only, and does not affect management nodes.

A node affected by this command disconnects from the cluster, and its associated `ndbd` or `ndb_mgmd` process terminates.

The `-a` option causes the node to be stopped immediately, without waiting for the completion of any pending transactions.

Normally, `STOP` fails if the result would cause an incomplete cluster. The `-f` option forces the node to shut down without checking for this. If this option is used and the result is an incomplete cluster, the cluster immediately shuts down.



Warning

Use of the `-a` option also disables the safety check otherwise performed when `STOP` is invoked to insure that stopping the node does not cause an incomplete cluster. In other words, you should exercise extreme care when using the `-a` option with the `STOP` command, due to the fact that this option makes it possible for the cluster to undergo a forced shutdown because it no longer has a complete copy of all data stored in `NDB`.

- `node_id RESTART [-n] [-i] [-a] [-f]`

Restarts the data node identified by `node_id` (or all data nodes).

Using the `-i` option with `RESTART` causes the data node to perform an initial restart; that is, the node's file system is deleted and recreated. The effect is the same as that obtained from stopping the data node process and then starting it again using `ndbd --initial [2178]` from the system shell. Note that backup files and Disk Data files are not removed when this option is used.

Using the `-n` option causes the data node process to be restarted, but the data node is not actually brought online until the appropriate `START` command is issued. The effect of this option is the same as that obtained from stopping the data node and then starting it again using `ndbd --nostart [2180]` or `ndbd -n` from the system shell.

Using the `-a` causes all current transactions relying on this node to be aborted. No GCP check is done when the node rejoins the cluster.

Normally, `RESTART` fails if taking the node offline would result in an incomplete cluster. The `-f` option forces the node to restart without checking for this. If this option is used and the result is an incomplete cluster, the entire cluster is restarted.

- `node_id STATUS`

Displays status information for the data node identified by `node_id` (or for all data nodes).

The output from this command also indicates when the cluster is in single user mode.

- `node_id REPORT report-type`

Displays a report of type `report-type` for the data node identified by `node_id`, or for all data nodes using `ALL`.

Currently, there are two accepted values for `report-type`:

- `BackupStatus` provides a status report on a cluster backup in progress
- `MemoryUsage` displays how much data memory and index memory is being used by each data node as shown in this example:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(177 32K pages of total 3200)
Node 1: Index usage is 0%(108 8K pages of total 12832)
Node 2: Data usage is 5%(177 32K pages of total 3200)
Node 2: Index usage is 0%(108 8K pages of total 12832)
```

This information is also available from the `ndbinfo.memoryusage` table.

`report-type` is case-insensitive and “fuzzy”; for `MemoryUsage`, you can use `MEMORY` (as shown in the prior example), `memory`, or even simply `MEM` (or `mem`). You can abbreviate `BackupStatus` in a similar fashion.

Prior to MySQL Cluster NDB 7.2.10, `ALL REPORT BackupStatus` did not work correctly with multithreaded data nodes. (Bug #15908907)

- `ENTER SINGLE USER MODE node_id`

Enters single user mode, whereby only the MySQL server identified by the node ID `node_id` is permitted to access the database.

Currently, it is not possible for data nodes to join a MySQL Cluster while it is running in single user mode. (Bug #20395)

- `EXIT SINGLE USER MODE`

Exits single user mode, enabling all SQL nodes (that is, all running `mysqld` processes) to access the database.



Note

It is possible to use `EXIT SINGLE USER MODE` even when not in single user mode, although the command has no effect in this case.

- `QUIT, EXIT`

Terminates the management client.

This command does not affect any nodes connected to the cluster.

- `SHUTDOWN`

Shuts down all cluster data nodes and management nodes. To exit the management client after this has been done, use `EXIT` or `QUIT`.

This command does *not* shut down any SQL nodes or API nodes that are connected to the cluster.

- `CREATE NODEGROUP nodeid[, nodeid, ...]`

Creates a new MySQL Cluster node group and causes data nodes to join it.

This command is used after adding new data nodes online to a MySQL Cluster, and causes them to join a new node group and thus to begin participating fully in the cluster. The command takes as its sole

parameter a comma-separated list of node IDs—these are the IDs of the nodes just added and started that are to join the new node group. The number of nodes must be the same as the number of nodes in each node group that is already part of the cluster (each MySQL Cluster node group must have the same number of nodes). In other words, if the MySQL Cluster has 2 node groups of 2 data nodes each, then the new node group must also have 2 data nodes.

The node group ID of the new node group created by this command is determined automatically, and always the next highest unused node group ID in the cluster; it is not possible to set it manually.

For more information, see [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#).

- `DROP NODEGROUP nodegroup_id`

Drops the MySQL Cluster node group with the given *nodegroup_id*.

This command can be used to drop a node group from a MySQL Cluster. `DROP NODEGROUP` takes as its sole argument the node group ID of the node group to be dropped.

`DROP NODEGROUP` acts only to remove the data nodes in the effected node group from that node group. It does not stop data nodes, assign them to a different node group, or remove them from the cluster's configuration. A data node that does not belong to a node group is indicated in the output of the management client `SHOW` command with `no nodegroup` in place of the node group ID, like this (indicated using bold text):

```
id=3      @10.100.2.67  (5.5.29-ndb-7.2.10, no nodegroup)
```

Prior to MySQL Cluster NDB 7.0.4, the `SHOW` output was not updated correctly following `DROP NODEGROUP`. (Bug #43413)

`DROP NODEGROUP` works only when all data nodes in the node group to be dropped are completely empty of any table data and table definitions. Since there is currently no way using `ndb_mgm` or the `mysql` client to remove all data from a specific data node or node group, this means that the command succeeds only in the two following cases:

1. After issuing `CREATE NODEGROUP` in the `ndb_mgm` client, but before issuing any `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statements in the `mysql` client.
2. After dropping all `NDBCLUSTER` tables using `DROP TABLE`.

`TRUNCATE TABLE` does not work for this purpose because this removes only the table data; the data nodes continue to store an `NDBCLUSTER` table's definition until a `DROP TABLE` statement is issued that causes the table metadata to be dropped.

For more information about `DROP NODEGROUP`, see [Section 17.5.13, “Adding MySQL Cluster Data Nodes Online”](#).

17.5.3. Online Backup of MySQL Cluster

The next few sections describe how to prepare for and then to create a MySQL Cluster backup using the functionality for this purpose found in the `ndb_mgm` management client. To distinguish this type of backup from a backup made using `mysqldump`, we sometimes refer to it as a “native” MySQL Cluster backup. (For information about the creation of backups with `mysqldump`, see [Section 4.5.4, “mysqldump — A Database Backup Program”](#).) Restoration of MySQL Cluster backups is done using the `ndb_restore` utility provided with the MySQL Cluster distribution; for information about `ndb_restore` and its use in restoring MySQL Cluster backups, see [Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”](#).

17.5.3.1. MySQL Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata.** The names and definitions of all database tables
- **Table records.** The data actually stored in the database tables at the time that the backup was made
- **Transaction log.** A sequential record telling how and when data was stored in the database

Each of these parts is saved on all nodes participating in the backup. During backup, each node saves these three parts into three files on disk:

- `BACKUP-backup_id.node_id.ctl`

A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- `BACKUP-backup_id-0.node_id.data`

A data file containing the table records, which are saved on a per-fragment basis. That is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states the tables to which the records belong. Following the list of records there is a footer containing a checksum for all records.

- `BACKUP-backup_id.node_id.log`

A log file containing records of committed transactions. Only transactions on tables stored in the backup are stored in the log. Nodes involved in the backup save different records because different nodes host different database fragments.

In the listing above, `backup_id` stands for the backup identifier and `node_id` is the unique identifier for the node creating the file.

17.5.3.2. Using The MySQL Cluster Management Client to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See [Section 17.5.3.3, "Configuration for MySQL Cluster Backups"](#).)

The `START BACKUP` command is used to create a backup:

```
START BACKUP [backup_id] [wait_option] [snapshot_option]

wait_option:
WAIT {STARTED | COMPLETED} | NOWAIT

snapshot_option:
SNAPSHOTSTART | SNAPSHOTEND
```

Successive backups are automatically identified sequentially, so the `backup_id`, an integer greater than or equal to 1, is optional; if it is omitted, the next available value is used. If an existing `backup_id` value is used, the backup fails with the error `Backup failed: file already exists`. If used, the `backup_id` must follow `START BACKUP` immediately, before any other options are used.

The `wait_option` can be used to determine when control is returned to the management client after a `START BACKUP` command is issued, as shown in the following list:

- If `NOWAIT` is specified, the management client displays a prompt immediately, as seen here:

```
ndb_mgm> START BACKUP NOWAIT
```

```
ndb_mgm>
```

In this case, the management client can be used even while it prints progress information from the backup process.

- With `WAIT STARTED` the management client waits until the backup has started before returning control to the user, as shown here:

```
ndb_mgm> START BACKUP WAIT STARTED
Waiting for started, this may take several minutes
Node 2: Backup 3 started from node 1
ndb_mgm>
```

- `WAIT COMPLETED` causes the management client to wait until the backup process is complete before returning control to the user.

`WAIT COMPLETED` is the default.

A `snapshot_option` can be used to determine whether the backup matches the state of the cluster when `START BACKUP` was issued, or when it was completed. `SNAPSHOTSTART` causes the backup to match the state of the cluster when the backup began; `SNAPSHOTEND` causes the backup to reflect the state of the cluster when the backup was finished. `SNAPSHOTEND` is the default, and matches the behavior found in previous MySQL Cluster releases.



Note

If you use the `SNAPSHOTSTART` option with `START BACKUP`, and the `CompressedBackup [2059]` parameter is enabled, only the data and control files are compressed—the log file is not compressed.

If both a `wait_option` and a `snapshot_option` are used, they may be specified in either order. For example, all of the following commands are valid, assuming that there is no existing backup having 4 as its ID:

```
START BACKUP WAIT STARTED SNAPSHOTSTART
START BACKUP SNAPSHOTSTART WAIT STARTED
START BACKUP 4 WAIT COMPLETED SNAPSHOTSTART
START BACKUP SNAPSHOTEND WAIT COMPLETED
START BACKUP 4 NOWAIT SNAPSHOTSTART
```

The procedure for creating a backup consists of the following steps:

1. Start the management client (`ndb_mgm`), if it not running already.
2. Execute the `START BACKUP` command. This produces several lines of output indicating the progress of the backup, as shown here:

```
ndb_mgm> START BACKUP
Waiting for completed, this may take several minutes
Node 2: Backup 1 started from node 1
Node 2: Backup 1 started from node 1 completed
StartGCP: 177 StopGCP: 180
#Records: 7362 #LogRecords: 0
Data: 453648 bytes Log: 0 bytes
ndb_mgm>
```

3. When the backup has started the management client displays this message:

```
Backup backup_id started from node node_id
```

`backup_id` is the unique identifier for this particular backup. This identifier is saved in the cluster log, if it has not been configured otherwise. `node_id` is the identifier of the management server

that is coordinating the backup with the data nodes. At this point in the backup process the cluster has received and processed the backup request. It does not mean that the backup has finished. An example of this statement is shown here:

```
Node 2: Backup 1 started from node 1
```

4. The management client indicates with a message like this one that the backup has started:

```
Backup backup_id started from node node_id completed
```

As is the case for the notification that the backup has started, *backup_id* is the unique identifier for this particular backup, and *node_id* is the node ID of the management server that is coordinating the backup with the data nodes. This output is accompanied by additional information including relevant global checkpoints, the number of records backed up, and the size of the data, as shown here:

```
Node 2: Backup 1 started from node 1 completed
StartGCP: 177 StopGCP: 180
#Records: 7362 #LogRecords: 0
Data: 453648 bytes Log: 0 bytes
```

It is also possible to perform a backup from the system shell by invoking `ndb_mgm` with the `-e` or `--execute` [2195] option, as shown in this example:

```
shell> ndb_mgm -e "START BACKUP 6 WAIT COMPLETED SNAPSHOTSTART"
```

When using `START BACKUP` in this way, you must specify the backup ID.

Cluster backups are created by default in the `BACKUP` subdirectory of the `DataDir` [2042] on each data node. This can be overridden for one or more data nodes individually, or for all cluster data nodes in the `config.ini` file using the `BackupDataDir` [2042] configuration parameter. The backup files created for a backup with a given *backup_id* are stored in a subdirectory named `BACKUP-backup_id` in the backup directory.

To abort a backup that is already in progress:

1. Start the management client.
2. Execute this command:

```
ndb_mgm> ABORT BACKUP backup_id
```

The number *backup_id* is the identifier of the backup that was included in the response of the management client when the backup was started (in the message `Backup backup_id started from node management_node_id`).

3. The management client will acknowledge the abort request with `Abort of backup backup_id ordered.`



Note

At this point, the management client has not yet received a response from the cluster data nodes to this request, and the backup has not yet actually been aborted.

4. After the backup has been aborted, the management client will report this fact in a manner similar to what is shown here:

```
Node 1: Backup 3 started from 5 has been aborted.
```

```

Error: 1321 - Backup aborted by user request: Permanent error: User defined error
Node 3: Backup 3 started from 5 has been aborted.
Error: 1323 - 1323: Permanent error: Internal error
Node 2: Backup 3 started from 5 has been aborted.
Error: 1323 - 1323: Permanent error: Internal error
Node 4: Backup 3 started from 5 has been aborted.
Error: 1323 - 1323: Permanent error: Internal error
    
```

In this example, we have shown sample output for a cluster with 4 data nodes, where the sequence number of the backup to be aborted is 3, and the management node to which the cluster management client is connected has the node ID 5. The first node to complete its part in aborting the backup reports that the reason for the abort was due to a request by the user. (The remaining nodes report that the backup was aborted due to an unspecified internal error.)



Note

There is no guarantee that the cluster nodes respond to an `ABORT BACKUP` command in any particular order.

The `Backup backup_id started from node management_node_id has been aborted` messages mean that the backup has been terminated and that all files relating to this backup have been removed from the cluster file system.

It is also possible to abort a backup in progress from a system shell using this command:

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```



Note

If there is no backup having the ID `backup_id` running when an `ABORT BACKUP` is issued, the management client makes no response, nor is it indicated in the cluster log that an invalid abort command was sent.

17.5.3.3. Configuration for MySQL Cluster Backups

Five configuration parameters are essential for backup:

- `BackupDataBufferSize` [2074]

The amount of memory used to buffer data before it is written to disk.

- `BackupLogBufferSize` [2074]

The amount of memory used to buffer log records before these are written to disk.

- `BackupMemory` [2075]

The total memory allocated in a data node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.

- `BackupWriteSize` [2075]

The default size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

- `BackupMaxWriteSize` [2076]

The maximum size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

More detailed information about these parameters can be found in [Backup Parameters](#).

17.5.3.4. MySQL Cluster Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or disk space. You should check that there is enough memory allocated for the backup.



Important

If you have set `BackupDataBufferSize` [2074] and `BackupLogBufferSize` [2074] and their sum is greater than 4MB, then you must also set `BackupMemory` [2075] as well.

You should also make sure that there is sufficient space on the hard drive partition of the backup target.

`NDB` does not support repeatable reads, which can cause problems with the restoration process. Although the backup process is “hot”, restoring a MySQL Cluster from backup is not a 100% “hot” process. This is due to the fact that, for the duration of the restore process, running transactions get nonrepeatable reads from the restored data. This means that the state of the data is inconsistent while the restore is in progress.

17.5.4. MySQL Server Usage for MySQL Cluster

`mysqld` is the traditional MySQL server process. To be used with MySQL Cluster, `mysqld` needs to be built with support for the `NDBCLUSTER` storage engine, as it is in the precompiled binaries available from <http://dev.mysql.com/downloads/>. If you build MySQL from source, you must invoke `configure` with one of the options to enable `NDBCLUSTER` storage engine support:

- `--with-plugins=ndbcluster`
- `--with-plugins=max`
- `--with-plugins=max-no-innodb`

(`--with-ndbcluster` also works to enable `NDBCLUSTER` support, but is deprecated and so produces a `configure` warning as of MySQL 5.1.9.)

For information about other MySQL server options and variables relevant to MySQL Cluster in addition to those discussed in this section, see [Section 17.3.4, “MySQL Server Options and Variables for MySQL Cluster”](#).

If the `mysqld` binary has been built with Cluster support, the `NDBCLUSTER` storage engine is still disabled by default. You can use either of two possible options to enable this engine:

- Use `--ndbcluster` [2142] as a startup option on the command line when starting `mysqld`.
- Insert a line containing `ndbcluster` in the `[mysqld]` section of your `my.cnf` file.

An easy way to verify that your server is running with the `NDBCLUSTER` storage engine enabled is to issue the `SHOW ENGINES` statement in the MySQL Monitor (`mysql`). You should see the value `YES` as the `Support` value in the row for `NDBCLUSTER`. If you see `NO` in this row or if there is no such row displayed in the output, you are not running an `NDB`-enabled version of MySQL. If you see `DISABLED` in this row, you need to enable it in either one of the two ways just described.

To read cluster configuration data, the MySQL server requires at a minimum three pieces of information:

- The MySQL server's own cluster node ID

- The host name or IP address for the management server (MGM node)
- The number of the TCP/IP port on which it can connect to the management server

Node IDs can be allocated dynamically, so it is not strictly necessary to specify them explicitly.

The `mysqld` parameter `ndb-connectstring` is used to specify the connectstring either on the command line when starting `mysqld` or in `my.cnf`. The connectstring contains the host name or IP address where the management server can be found, as well as the TCP/IP port it uses.

In the following example, `ndb_mgmd.mysql.com` is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
shell> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See [Section 17.3.2.3, “The MySQL Cluster Connectstring”](#), for more information on connectstrings.

Given this information, the MySQL server will be a full participant in the cluster. (We often refer to a `mysqld` process running in this manner as an SQL node.) It will be fully aware of all cluster data nodes as well as their status, and will establish connections to all data nodes. In this case, it is able to use any data node as a transaction coordinator and to read and update node data.

You can see in the `mysql` client whether a MySQL server is connected to the cluster using `SHOW PROCESSLIST`. If the MySQL server is connected to the cluster, and you have the `PROCESS [696]` privilege, then the first row of the output is as shown here:

```
mysql> SHOW PROCESSLIST \G
***** 1. row *****
  Id: 1
  User: system user
  Host:
  db:
Command: Daemon
  Time: 1
  State: Waiting for event from ndbcluster
  Info: NULL
```



Important

To participate in a MySQL Cluster, the `mysqld` process must be started with *both* the options `--ndbcluster [2142]` and `--ndb-connectstring [2141]` (or their equivalents in `my.cnf`). If `mysqld` is started with only the `--ndbcluster [2142]` option, or if it is unable to contact the cluster, it is not possible to work with `NDB` tables, *nor is it possible to create any new tables regardless of storage engine*. The latter restriction is a safety measure intended to prevent the creation of tables having the same names as `NDB` tables while the SQL node is not connected to the cluster. If you wish to create tables using a different storage engine while the `mysqld` process is not participating in a MySQL Cluster, you must restart the server *without* the `--ndbcluster [2142]` option.

17.5.5. Performing a Rolling Restart of a MySQL Cluster

This section discusses how to perform a *rolling restart* of a MySQL Cluster installation, so called because it involves stopping and starting (or restarting) each node in turn, so that the cluster itself remains operational. This is often done as part of a *rolling upgrade* or *rolling downgrade*, where high availability of the cluster is mandatory and no downtime of the cluster as a whole is permissible. Where we refer to upgrades, the information provided here also generally applies to downgrades as well.

There are a number of reasons why a rolling restart might be desirable. These are described in the next few paragraphs.

Configuration change.

To make a change in the cluster's configuration, such as adding an SQL node to the cluster, or setting a configuration parameter to a new value.

MySQL Cluster software upgrade or downgrade. To upgrade the cluster to a newer version of the MySQL Cluster software (or to downgrade it to an older version). This is usually referred to as a “rolling upgrade” (or “rolling downgrade”, when reverting to an older version of MySQL Cluster).

Change on node host. To make changes in the hardware or operating system on which one or more MySQL Cluster node processes are running.

System reset (cluster reset).

To reset the cluster because it has reached an undesirable state. In such cases it is often desirable to reload the data and metadata of one or more data nodes. This can be done in any of three ways:

- Start each data node process (`ndbd` or possibly `ndbmtid`) with the `--initial [2178]` option, which forces the data node to clear its file system and to reload all MySQL Cluster data and metadata from the other data nodes.
- Create a backup using the `ndb_mgm` client `BACKUP` command prior to performing the restart. Following the upgrade, restore the node or nodes using `ndb_restore`.

See [Section 17.5.3, “Online Backup of MySQL Cluster”](#), and [Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”](#), for more information.

- Use `mysqldump` to create a backup prior to the upgrade; afterward, restore the dump using `LOAD DATA INFILE`.

Resource Recovery.

To free memory previously allocated to a table by successive `INSERT` and `DELETE` operations, for re-use by other MySQL Cluster tables.

The process for performing a rolling restart may be generalized as follows:

1. Stop all cluster management nodes (`ndb_mgmd` processes), reconfigure them, then restart them. (See [Rolling restarts with multiple management servers.](#))
2. Stop, reconfigure, then restart each cluster data node (`ndbd` process) in turn.
3. Stop, reconfigure, then restart each cluster SQL node (`mysqld` process) in turn.

The specifics for implementing a given rolling upgrade depend upon the changes being made. A more detailed view of the process is presented here:

RESTART TYPE:					
Cluster Configuration Change		Cluster Software Upgrade or Downgrade	Change on Node Host	Cluster Reset	
A. Management node (ndb_mgmd) processes...					
<ol style="list-style-type: none"> 1. Stop all ndb_mgmd processes 2. Make changes in global configuration file(s) 3. Start all ndb_mgmd processes 		<ol style="list-style-type: none"> 1. Stop all ndb_mgmd processes 2. Replace each ndb_mgmd binary with new version 3. Start ndb_mgmd processes 	<ol style="list-style-type: none"> 1. Stop all ndb_mgmd processes 2. Make desired changes in hardware, operating system, or both 3. Start all ndb_mgmd processes 	(OR)	
				<ol style="list-style-type: none"> 1. Stop all ndb_mgmd processes 2. Start all ndb_mgmd processes 	Restart all ndb_mgmd processes (optional)
B. For each data node (ndbd) process...					
(OR)				(OR)	
<ol style="list-style-type: none"> 1. Stop ndbd 2. Start ndbd 	Restart ndbd	<ol style="list-style-type: none"> 1. Stop ndbd 2. Replace ndbd binary with new version 3. Start ndbd 	<ol style="list-style-type: none"> 1. Stop ndbd 2. Make desired changes in hardware, operating system, or both 3. Start ndbd 	<ol style="list-style-type: none"> 1. Stop ndbd 2. Start ndbd 	Restart ndbd
C. For each SQL node (mysqld) process...					
(OR)				(OR)	
<ol style="list-style-type: none"> 1. Stop mysqld 2. Start mysqld 	Restart mysqld	<ol style="list-style-type: none"> 1. Stop mysqld 2. Replace mysqld binary with new version 3. Start mysqld 	<ol style="list-style-type: none"> 1. Stop mysqld 2. Make desired changes in hardware, operating system, or both 3. Start mysqld 	<ol style="list-style-type: none"> 1. Stop mysqld 2. Start mysqld 	Restart mysqld

In the previous diagram, the **Stop** and **Start** steps indicate that the process must be stopped completely using a shell command (such as `kill` on most Unix systems) or the management client `STOP` command, then started again from a system shell by invoking the `ndbd` or `ndb_mgmd` executable as appropriate. On Windows, you can also use the system `NET START` and `NET STOP` commands or the Windows Service Manager to start and stop nodes which have been installed as Windows services (see [Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”](#)).

Restart indicates that the process may be restarted using the `ndb_mgm` management client `RESTART` command (see [Section 17.5.2, “Commands in the MySQL Cluster Management Client”](#)).

MySQL Cluster supports a flexible order for upgrading nodes. When upgrading a MySQL Cluster, you may upgrade API nodes (including SQL nodes) before upgrading the management nodes, data nodes, or both. In other words, you are permitted to upgrade the API and SQL nodes in any order. This is subject to the following provisions:

- This functionality is intended for use as part of an online upgrade only. A mix of node binaries from different MySQL Cluster releases is neither intended nor supported for continuous, long-term use in a production setting.
- All management nodes must be upgraded before any data nodes are upgraded. This remains true regardless of the order in which you upgrade the cluster's API and SQL nodes.
- Features specific to the “new” version must not be used until all management nodes and data nodes have been upgraded.

This also applies to any MySQL Server version change that may apply, in addition to the NDB engine version change, so do not forget to take this into account when planning the upgrade. (This is true for online upgrades of MySQL Cluster in general.)

See also Bug #48528 and Bug #49163.

Rolling restarts with multiple management servers. When performing a rolling restart of a MySQL Cluster with multiple management nodes, you should keep in mind that `ndb_mgmd` checks to see if any other management node is running, and, if so, tries to use that node's configuration data. To keep this from occurring, and to force `ndb_mgmd` to reread its configuration file, perform the following steps:

1. Stop all MySQL Cluster `ndb_mgmd` processes.
2. Update all `config.ini` files.
3. Start a single `ndb_mgmd` with `--reload [2191]`, `--initial [2190]`, or both options as desired.
4. Start any remaining `ndb_mgmd` processes *without using either of the `--reload [2191]` or `--initial [2190]` options.*
5. Complete the rolling restarts of the data nodes and API nodes as normal.

When performing a rolling restart to update the cluster's configuration, you can use the `config_generation` column of the `ndbinfo.nodes` table to keep track of which data nodes have been successfully restarted with the new configuration. See [Section 17.5.10.13, “The `ndbinfo nodes` Table”](#).

17.5.6. Event Reports Generated in MySQL Cluster

In this section, we discuss the types of event logs provided by MySQL Cluster, and the types of events that are logged.

MySQL Cluster provides two types of event log:

- The *cluster log*, which includes events generated by all cluster nodes. The cluster log is the log recommended for most uses because it provides logging information for an entire cluster in a single location.

By default, the cluster log is saved to a file named `ndb_node_id_cluster.log`, (where `node_id` is the node ID of the management server) in the same directory where the `ndb_mgm` binary resides.

Cluster logging information can also be sent to `stdout` or a `syslog` facility in addition to or instead of being saved to a file, as determined by the values set for the `DataDir` [2037] and `LogDestination` [2036] configuration parameters. See [Section 17.3.2.5, “Defining a MySQL Cluster Management Server”](#), for more information about these parameters.

- *Node logs* are local to each node.

Output generated by node event logging is written to the file `ndb_node_id_out.log` (where `node_id` is the node's node ID) in the node's `DataDir` [2042]. Node event logs are generated for both management nodes and data nodes.

Node logs are intended to be used only during application development, or for debugging application code.

Both types of event logs can be set to log different subsets of events.

Each reportable event can be distinguished according to three different criteria:

- *Category*: This can be any one of the following values: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, or `INFO`.
- *Priority*: This is represented by one of the numbers from 1 to 15 inclusive, where 1 indicates “most important” and 15 “least important.”
- *Severity Level*: This can be any one of the following values: `ALERT`, `CRITICAL`, `ERROR`, `WARNING`, `INFO`, or `DEBUG`.

Both the cluster log and the node log can be filtered on these properties.

The format used in the cluster log is as shown here:

```
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 1: Data usage is 2%(60 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 1: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 1: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 2: Data usage is 2%(76 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 2: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 2: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 3: Data usage is 2%(58 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 3: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 3: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 4: Data usage is 2%(74 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 4: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 4: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 4: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 1: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 1: Node 9: API 5.5.29-ndb-7.2.10
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 2: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 2: Node 9: API 5.5.29-ndb-7.2.10
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 3: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 3: Node 9: API 5.5.29-ndb-7.2.10
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 4: Node 9: API 5.5.29-ndb-7.2.10
2007-01-26 19:59:22 [MgmSrvr] ALERT     -- Node 2: Node 7 Disconnected
2007-01-26 19:59:22 [MgmSrvr] ALERT     -- Node 2: Node 7 Disconnected
```

Each line in the cluster log contains the following information:

- A timestamp in `YYYY-MM-DD HH:MM:SS` format.
- The type of node which is performing the logging. In the cluster log, this is always `[MgmSrvr]`.
- The severity of the event.

- The ID of the node reporting the event.
- A description of the event. The most common types of events to appear in the log are connections and disconnections between different nodes in the cluster, and when checkpoints occur. In some cases, the description may contain status information.

17.5.6.1. MySQL Cluster Logging Management Commands

The following management commands are related to the cluster log:

- `CLUSTERLOG ON`

Turns the cluster log on.

- `CLUSTERLOG OFF`

Turns the cluster log off.

- `CLUSTERLOG INFO`

Provides information about cluster log settings.

- `node_id CLUSTERLOG category=threshold`

Logs *category* events with priority less than or equal to *threshold* in the cluster log.

- `CLUSTERLOG FILTER severity_level`

Toggles cluster logging of events of the specified *severity_level*.

The following table describes the default setting (for all data nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, it is reported in the cluster log.

Note that events are reported per data node, and that the threshold can be set to different values on different nodes.

Category	Default threshold (All data nodes)
<code>STARTUP</code>	7
<code>SHUTDOWN</code>	7
<code>STATISTICS</code>	7
<code>CHECKPOINT</code>	7
<code>NODERESTART</code>	7
<code>CONNECTION</code>	7
<code>ERROR</code>	15
<code>INFO</code>	7

The `STATISTICS` category can provide a great deal of useful data. See [Section 17.5.6.3, “Using `CLUSTERLOG STATISTICS` in the MySQL Cluster Management Client”](#), for more information.

Thresholds are used to filter events within each category. For example, a `STARTUP` event with a priority of 3 is not logged unless the threshold for `STARTUP` is set to 3 or higher. Only events with priority 3 or lower are sent if the threshold is 3.

The following table shows the event severity levels.

**Note**

These correspond to Unix `syslog` levels, except for `LOG_EMERG` and `LOG_NOTICE`, which are not used or mapped.

1	<code>ALERT</code>	A condition that should be corrected immediately, such as a corrupted system database
2	<code>CRITICAL</code>	Critical conditions, such as device errors or insufficient resources
3	<code>ERROR</code>	Conditions that should be corrected, such as configuration errors
4	<code>WARNING</code>	Conditions that are not errors, but that might require special handling
5	<code>INFO</code>	Informational messages
6	<code>DEBUG</code>	Debugging messages used for <code>NDBCLUSTER</code> development

Event severity levels can be turned on or off (using `CLUSTERLOG FILTER`—see above). If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

**Important**

Cluster log levels are set on a per `ndb_mgmd`, per subscriber basis. This means that, in a MySQL Cluster with multiple management servers, using a `CLUSTERLOG` command in an instance of `ndb_mgm` connected to one management server affects only logs generated by that management server but not by any of the others. This also means that, should one of the management servers be restarted, only logs generated by that management server are affected by the resetting of log levels caused by the restart.

17.5.6.2. MySQL Cluster Log Events

An event report reported in the event logs has the following format:

```
datetime [string] severity -- message
```

For example:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

This section discusses all reportable events, ordered by category and severity level within each category.

In the event descriptions, GCP and LCP mean “Global Checkpoint” and “Local Checkpoint”, respectively.

CONNECTION Events

These events are associated with connections between Cluster nodes.

Event	Priority	Severity Level	Description
data nodes connected	8	<code>INFO</code>	Data nodes connected
data nodes disconnected	8	<code>INFO</code>	Data nodes disconnected
Communication closed	8	<code>INFO</code>	SQL node or data node connection closed
Communication opened	8	<code>INFO</code>	SQL node or data node connection opened

CHECKPOINT Events

The logging messages shown here are associated with checkpoints.

Event	Priority	Severity Level	Description
LCP stopped in calc keep GCI	0	ALERT	LCP stopped
Local checkpoint fragment completed	11	INFO	LCP on a fragment has been completed
Global checkpoint completed	10	INFO	GCP finished
Global checkpoint started	9	INFO	Start of GCP: REDO log is written to disk
Local checkpoint completed	8	INFO	LCP completed normally
Local checkpoint started	7	INFO	Start of LCP: data written to disk

STARTUP Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

Event	Priority	Severity Level	Description
Internal start signal received STTORY	15	INFO	Blocks received after completion of restart
New REDO log started	10	INFO	GCI keep <i>x</i> , newest restorable GCI <i>y</i>
New log started	10	INFO	Log part <i>x</i> , start MB <i>y</i> , stop MB <i>z</i>
Node has been refused for inclusion in the cluster	8	INFO	Node cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem
data node neighbors	8	INFO	Shows neighboring data nodes
data node start phase <i>x</i> completed	4	INFO	A data node start phase has been completed
Node has been successfully included into the cluster	3	INFO	Displays the node, managing node, and dynamic ID
data node start phases initiated	1	INFO	NDB Cluster nodes starting
data node all start phases completed	1	INFO	NDB Cluster nodes started
data node shutdown initiated	1	INFO	Shutdown of data node has commenced
data node shutdown aborted	1	INFO	Unable to shut down data node normally

NODERESTART Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

Event	Priority	Severity Level	Description
Node failure phase completed	8	ALERT	Reports completion of node failure phases
Node has failed, node state was <i>x</i>	8	ALERT	Reports that a node has failed
Report arbitrator results	2	ALERT	There are eight different possible results for arbitration attempts:

Event Reports Generated in MySQL Cluster

Event	Priority	Severity Level	Description
			<ul style="list-style-type: none"> • Arbitration check failed—less than 1/2 nodes left • Arbitration check succeeded—node group majority • Arbitration check failed—missing node group • Network partitioning—arbitration required • Arbitration succeeded—affirmative response from node <i>x</i> • Arbitration failed - negative response from node <i>x</i> • Network partitioning - no arbitrator available • Network partitioning - no arbitrator configured
Completed copying a fragment	10	INFO	
Completed copying of dictionary information	8	INFO	
Completed copying distribution information	8	INFO	
Starting to copy fragments	8	INFO	
Completed copying all fragments	8	INFO	
GCP takeover started	7	INFO	
GCP takeover completed	7	INFO	
LCP takeover started	7	INFO	
LCP takeover completed (state = <i>x</i>)	7	INFO	
Report whether an arbitrator is found or not	6	INFO	<p>There are seven different possible outcomes when seeking an arbitrator:</p> <ul style="list-style-type: none"> • Management server restarts arbitration thread [state=<i>x</i>] • Prepare arbitrator node <i>x</i> [ticket=<i>y</i>] • Receive arbitrator node <i>x</i> [ticket=<i>y</i>] • Started arbitrator node <i>x</i> [ticket=<i>y</i>] • Lost arbitrator node <i>x</i> - process failure [state=<i>y</i>] • Lost arbitrator node <i>x</i> - process exit [state=<i>y</i>] • Lost arbitrator node <i>x</i> <error msg> [state=<i>y</i>]

STATISTICS Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

Event	Priority	Severity Level	Description
Report job scheduling statistics	9	INFO	Mean internal job scheduling statistics
Sent number of bytes	9	INFO	Mean number of bytes sent to node <i>X</i>
Received # of bytes	9	INFO	Mean number of bytes received from node <i>X</i>
Report transaction statistics	8	INFO	Numbers of: transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts
Report operations	8	INFO	Number of operations
Report table create	7	INFO	
Memory usage	5	INFO	Data and index memory usage (80%, 90%, and 100%)

ERROR Events

These events relate to Cluster errors and warnings. The presence of one or more of these generally indicates that a major malfunction or failure has occurred.

Event	Priority	Severity	Description
Dead due to missed heartbeat	8	ALERT	Node <i>X</i> declared "dead" due to missed heartbeat
Transporter errors	2	ERROR	
Transporter warnings	8	WARNING	
Missed heartbeats	8	WARNING	Node <i>X</i> missed heartbeat # <i>Y</i>
General warning events	2	WARNING	

INFO Events

These events provide general information about the state of the cluster and activities associated with Cluster maintenance, such as logging and heartbeat transmission.

Event	Priority	Severity	Description
Sent heartbeat	12	INFO	Heartbeat sent to node <i>X</i>
Create log bytes	11	INFO	Log part, log file, MB
General information events	2	INFO	

17.5.6.3. Using CLUSTERLOG STATISTICS in the MySQL Cluster Management Client

The NDB management client's `CLUSTERLOG STATISTICS` command can provide a number of useful statistics in its output. Counters providing information about the state of the cluster are updated at 5-second reporting intervals by the transaction coordinator (TC) and the local query handler (LQH), and written to the cluster log.

Transaction coordinator statistics. Each transaction has one transaction coordinator, which is chosen by one of the following methods:

- In a round-robin fashion
- By communication proximity
- (*Beginning with MySQL Cluster NDB 6.3.4:*) By supplying a data placement hint when the transaction is started

**Note**

You can determine which TC selection method is used for transactions started from a given SQL node using the `ndb_optimized_node_selection [2154]` system variable.

All operations within the same transaction use the same transaction coordinator, which reports the following statistics:

- **Trans count.** This is the number transactions started in the last interval using this TC as the transaction coordinator. Any of these transactions may have committed, have been aborted, or remain uncommitted at the end of the reporting interval.

**Note**

Transactions do not migrate between TCs.

- **Commit count.** This is the number of transactions using this TC as the transaction coordinator that were committed in the last reporting interval. Because some transactions committed in this reporting interval may have started in a previous reporting interval, it is possible for `Commit count` to be greater than `Trans count`.
- **Read count.** This is the number of primary key read operations using this TC as the transaction coordinator that were started in the last reporting interval, including simple reads. This count also includes reads performed as part of unique index operations. A unique index read operation generates 2 primary key read operations—1 for the hidden unique index table, and 1 for the table on which the read takes place.
- **Simple read count.** This is the number of simple read operations using this TC as the transaction coordinator that were started in the last reporting interval. This is a subset of `Read count`. Because the value of `Simple read count` is incremented at a different point in time from `Read count`, it can lag behind `Read count` slightly, so it is conceivable that `Simple read count` is not equal to `Read count` for a given reporting interval, even if all reads made during that time were in fact simple reads.
- **Write count.** This is the number of primary key write operations using this TC as the transaction coordinator that were started in the last reporting interval. This includes all inserts, updates, writes and deletes, as well as writes performed as part of unique index operations.

**Note**

A unique index update operation can generate multiple PK read and write operations on the index table and on the base table.

- **AttrInfoCount.** This is the number of 32-bit data words received in the last reporting interval for primary key operations using this TC as the transaction coordinator. For reads, this is proportional to the number of columns requested. For inserts and updates, this is proportional to the number of columns written, and the size of their data. For delete operations, this is usually zero.

Unique index operations generate multiple PK operations and so increase this count. However, data words sent to describe the PK operation itself, and the key information sent, are *not* counted here.

Attribute information sent to describe columns to read for scans, or to describe ScanFilters, is also not counted in `AttrInfoCount`.

- **Concurrent Operations.** This is the number of primary key or scan operations using this TC as the transaction coordinator that were started during the last reporting interval but that were not completed. Operations increment this counter when they are started and decrement it when they are completed; this occurs after the transaction commits. Dirty reads and writes—as well as failed operations—decrement this counter.

The maximum value that `Concurrent Operations` can have is the maximum number of operations that a TC block can support; currently, this is $(2 * \text{MaxNoOfConcurrentOperations}) + 16 + \text{MaxNoOfConcurrentTransactions}$. (For more information about these configuration parameters, see the *Transaction Parameters* section of [Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”](#).)

- **Abort count.** This is the number of transactions using this TC as the transaction coordinator that were aborted during the last reporting interval. Because some transactions that were aborted in the last reporting interval may have started in a previous reporting interval, `Abort count` can sometimes be greater than `Trans count`.
- **Scans.** This is the number of table scans using this TC as the transaction coordinator that were started during the last reporting interval. This does not include range scans (that is, ordered index scans).
- **Range scans.** This is the number of ordered index scans using this TC as the transaction coordinator that were started in the last reporting interval.

Local query handler statistics (`Operations`). There is 1 cluster event per local query handler block (that is, 1 per data node process). Operations are recorded in the LQH where the data they are operating on resides.



Note

A single transaction may operate on data stored in multiple LQH blocks.

The `Operations` statistic provides the number of local operations performed by this LQH block in the last reporting interval, and includes all types of read and write operations (insert, update, write, and delete operations). This also includes operations used to replicate writes. For example, in a 2-replica cluster, the write to the primary replica is recorded in the primary LQH, and the write to the backup will be recorded in the backup LQH. Unique key operations may result in multiple local operations; however, this does *not* include local operations generated as a result of a table scan or ordered index scan, which are not counted.

Process scheduler statistics. In addition to the statistics reported by the transaction coordinator and local query handler, each `ndbd` process has a scheduler which also provides useful metrics relating to the performance of a MySQL Cluster. This scheduler runs in an infinite loop; during each loop the scheduler performs the following tasks:

1. Read any incoming messages from sockets into a job buffer.
2. Check whether there are any timed messages to be executed; if so, put these into the job buffer as well.
3. Execute (in a loop) any messages in the job buffer.
4. Send any distributed messages that were generated by executing the messages in the job buffer.
5. Wait for any new incoming messages.

Process scheduler statistics include the following:

- **Mean Loop Counter.** This is the number of loops executed in the third step from the preceding list. This statistic increases in size as the utilization of the TCP/IP buffer improves. You can use this to monitor changes in performance as you add new data node processes.
- **Mean send size and Mean receive size.** These statistics enable you to gauge the efficiency of, respectively writes and reads between nodes. The values are given in bytes. Higher values mean a lower cost per byte sent or received; the maximum value is 64K.

To cause all cluster log statistics to be logged, you can use the following command in the **NDB** management client:

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```



Note

Setting the threshold for **STATISTICS** to 15 causes the cluster log to become very verbose, and to grow quite rapidly in size, in direct proportion to the number of cluster nodes and the amount of activity in the MySQL Cluster.

For more information about MySQL Cluster management client commands relating to logging and reporting, see [Section 17.5.6.1, “MySQL Cluster Logging Management Commands”](#).

17.5.7. MySQL Cluster Log Messages

This section contains information about the messages written to the cluster log in response to different cluster log events. It provides additional, more specific information on **NDB** transporter errors.

17.5.7.1. MySQL Cluster: Messages in the Cluster Log

The following table lists the most common **NDB** cluster log messages. For information about the cluster log, log events, and event types, see [Section 17.5.6, “Event Reports Generated in MySQL Cluster”](#). These log messages also correspond to log event types in the MGM API; see [The Ndb_logevent_type Type](#), for related information of interest to Cluster API developers.

<p>Log Message. Node <i>mgm_node_id</i>: Node <i>data_node_id</i> Connected</p> <p>Description. The data node having node ID <i>node_id</i> has connected to the management server (node <i>mgm_node_id</i>).</p>	<p>Event Name. Connected</p> <p>Event Type. Connection</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>mgm_node_id</i>: Node <i>data_node_id</i> Disconnected</p> <p>Description. The data node having node ID <i>data_node_id</i> has disconnected from the management server (node <i>mgm_node_id</i>).</p>	<p>Event Name. Disconnected</p> <p>Event Type. Connection</p> <p>Priority. 8</p> <p>Severity. ALERT</p>
<p>Log Message. Node <i>data_node_id</i>: Communication to Node <i>api_node_id</i> closed</p> <p>Description. The API node or SQL node having node ID <i>api_node_id</i> is no longer communicating with data node <i>data_node_id</i>.</p>	<p>Event Name. CommunicationClosed</p> <p>Event Type. Connection</p> <p>Priority. 8</p> <p>Severity. INFO</p>

MySQL Cluster Log Messages

<p>Log Message. Node <i>data_node_id</i>: Communication to Node <i>api_node_id</i> opened</p> <p>Description. The API node or SQL node having node ID <i>api_node_id</i> is now communicating with data node <i>data_node_id</i>.</p>	<p>Event Name. CommunicationOpened</p> <p>Event Type. Connection</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>mgm_node_id</i>: Node <i>api_node_id</i>: API version</p> <p>Description. The API node having node ID <i>api_node_id</i> has connected to management node <i>mgm_node_id</i> using NDB API version <i>version</i> (generally the same as the MySQL version number).</p>	<p>Event Name. ConnectedApiVersion</p> <p>Event Type. Connection</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Global checkpoint <i>gci</i> started</p> <p>Description. A global checkpoint with the ID <i>gci</i> has been started; node <i>node_id</i> is the master responsible for this global checkpoint.</p>	<p>Event Name. GlobalCheckpointStarted</p> <p>Event Type. Checkpoint</p> <p>Priority. 9</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Global checkpoint <i>gci</i> completed</p> <p>Description. The global checkpoint having the ID <i>gci</i> has been completed; node <i>node_id</i> was the master responsible for this global checkpoint.</p>	<p>Event Name. GlobalCheckpointCompleted</p> <p>Event Type. Checkpoint</p> <p>Priority. 10</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Local checkpoint <i>lcp</i> started. Keep GCI = <i>current_gci</i> oldest restorable GCI = <i>old_gci</i></p> <p>Description. The local checkpoint having sequence ID <i>lcp</i> has been started on node <i>node_id</i>. The most recent GCI that can be used has the index <i>current_gci</i>, and the oldest GCI from which the cluster can be restored has the index <i>old_gci</i>.</p>	<p>Event Name. LocalCheckpointStarted</p> <p>Event Type. Checkpoint</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Local checkpoint <i>lcp</i> completed</p> <p>Description. The local checkpoint having sequence ID <i>lcp</i> on node <i>node_id</i> has been completed.</p>	<p>Event Name. LocalCheckpointCompleted</p> <p>Event Type. Checkpoint</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Local Checkpoint stopped in CALCULATED_KEEP_GCI</p> <p>Description. The node was unable to determine the most recent usable GCI.</p>	<p>Event Name. LCPStoppedInCalcKeepGci</p> <p>Event Type. Checkpoint</p> <p>Priority. 0</p>

MySQL Cluster Log Messages

	<p>Severity. ALERT</p>
<p>Log Message. Node <i>node_id</i>: Table ID = <i>table_id</i>, fragment ID = <i>fragment_id</i> has completed LCP on Node <i>node_id</i> maxGciStarted: <i>started_gci</i> maxGciCompleted: <i>completed_gci</i></p> <p>Description. A table fragment has been checkpointed to disk on node <i>node_id</i>. The GCI in progress has the index <i>started_gci</i>, and the most recent GCI to have been completed has the index <i>completed_gci</i>.</p>	<p>Event Name. LCPFragmentCompleted</p> <p>Event Type. Checkpoint</p> <p>Priority. 11</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: ACC Blocked <i>num_1</i> and TUP Blocked <i>num_2</i> times last second</p> <p>Description. Undo logging is blocked because the log buffer is close to overflowing.</p>	<p>Event Name. UndoLogBlocked</p> <p>Event Type. Checkpoint</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Start initiated <i>version</i></p> <p>Description. Data node <i>node_id</i>, running NDB version <i>version</i>, is beginning its startup process.</p>	<p>Event Name. NDBStartStarted</p> <p>Event Type. StartUp</p> <p>Priority. 1</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Started <i>version</i></p> <p>Description. Data node <i>node_id</i>, running NDB version <i>version</i>, has started successfully.</p>	<p>Event Name. NDBStartCompleted</p> <p>Event Type. StartUp</p> <p>Priority. 1</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: STTORY received after restart finished</p> <p>Description. The node has received a signal indicating that a cluster restart has completed.</p>	<p>Event Name. STTORYRecieved</p> <p>Event Type. StartUp</p> <p>Priority. 15</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Start phase <i>phase</i> completed (<i>type</i>)</p> <p>Description. The node has completed start phase <i>phase</i> of a <i>type</i> start. For a listing of start phases, see Section 17.5.1, "Summary of MySQL Cluster Start Phases". (<i>type</i> is one of <i>initial</i>, <i>system</i>, <i>node</i>, <i>initial node</i>, or <Unknown>.)</p>	<p>Event Name. StartPhaseCompleted</p> <p>Event Type. StartUp</p> <p>Priority. 4</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: CM_REGCONF president = <i>president_id</i>, own Node = <i>own_id</i>, our dynamic id = <i>dynamic_id</i></p> <p>Description. Node <i>president_id</i> has been selected as "president". <i>own_id</i> and <i>dynamic_id</i> should always be the same as the ID (<i>node_id</i>) of the reporting node.</p>	<p>Event Name. CM_REGCONF</p> <p>Event Type. StartUp</p> <p>Priority. 3</p> <p>Severity. INFO</p>

MySQL Cluster Log Messages

<p>Log Message. Node <i>node_id</i>: CM_REGREF from Node <i>president_id</i> to our Node <i>node_id</i>. Cause = <i>cause</i></p> <p>Description. The reporting node (ID <i>node_id</i>) was unable to accept node <i>president_id</i> as president. The <i>cause</i> of the problem is given as one of <i>Busy</i>, <i>Election with wait = false</i>, <i>Not president</i>, <i>Election without selecting new candidate</i>, or <i>No such cause</i>.</p>	<p>Event Name. CM_REGREF</p> <p>Event Type. StartUp</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: We are Node <i>own_id</i> with dynamic ID <i>dynamic_id</i>, our left neighbor is Node <i>id_1</i>, our right is Node <i>id_2</i></p> <p>Description. The node has discovered its neighboring nodes in the cluster (node <i>id_1</i> and node <i>id_2</i>). <i>node_id</i>, <i>own_id</i>, and <i>dynamic_id</i> should always be the same; if they are not, this indicates a serious misconfiguration of the cluster nodes.</p>	<p>Event Name. FIND_NEIGHBOURS</p> <p>Event Type. StartUp</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: <i>type</i> shutdown initiated</p> <p>Description. The node has received a shutdown signal. The <i>type</i> of shutdown is either <i>Cluster</i> or <i>Node</i>.</p>	<p>Event Name. NDBStopStarted</p> <p>Event Type. StartUp</p> <p>Priority. 1</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Node shutdown completed [, <i>action</i>] [Initiated by signal <i>signal</i>.]</p> <p>Description. The node has been shut down. This report may include an <i>action</i>, which if present is one of <i>restarting</i>, <i>no start</i>, or <i>initial</i>. The report may also include a reference to an NDB Protocol <i>signal</i>; for possible signals, refer to Operations and Signals.</p>	<p>Event Name. NDBStopCompleted</p> <p>Event Type. StartUp</p> <p>Priority. 1</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Forced node shutdown completed [, <i>action</i>]. [Occured during startphase <i>start_phase</i>.] [Initiated by <i>signal</i>.] [Caused by error <i>error_code</i>: '<i>error_message(error_classification)</i>.<i>error_status</i>'. [(extra info <i>extra_code</i>)]]</p> <p>Description. The node has been forcibly shut down. The <i>action</i> (one of <i>restarting</i>, <i>no start</i>, or <i>initial</i>) subsequently being taken, if any, is also reported. If the shutdown occurred while the node was starting, the report includes the <i>start_phase</i> during which the node failed. If this was a result of a <i>signal</i> sent to the node, this information is also provided (see Operations and Signals, for more information). If the error causing the failure is known, this is also included; for more information about NDB error messages and classifications, see MySQL Cluster API Errors.</p>	<p>Event Name. NDBStopForced</p> <p>Event Type. StartUp</p> <p>Priority. 1</p> <p>Severity. ALERT</p>
<p>Log Message. Node <i>node_id</i>: Node shutdown aborted</p>	<p>Event Name. NDBStopAborted</p>

MySQL Cluster Log Messages

<p>Description. The node shutdown process was aborted by the user.</p>	<p>Event Type. StartUp</p> <p>Priority. 1</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: StartLog: [GCI Keep: <i>keep_pos</i> LastCompleted: <i>last_pos</i> NewestRestorable: <i>restore_pos</i>]</p> <p>Description. This reports global checkpoints referenced during a node start. The redo log prior to <i>keep_pos</i> is dropped. <i>last_pos</i> is the last global checkpoint in which data node the participated; <i>restore_pos</i> is the global checkpoint which is actually used to restore all data nodes.</p>	<p>Event Name. StartREDOLog</p> <p>Event Type. StartUp</p> <p>Priority. 4</p> <p>Severity. INFO</p>
<p>Log Message. <i>startup_message</i> [Listed separately; see below.]</p> <p>Description. There are a number of possible startup messages that can be logged under different circumstances.</p>	<p>Event Name. StartReport</p> <p>Event Type. StartUp</p> <p>Priority. 4</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Node restart completed copy of dictionary information</p> <p>Description. Copying of data dictionary information to the restarted node has been completed.</p>	<p>Event Name. NR_CopyDict</p> <p>Event Type. NodeRestart</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Node restart completed copy of distribution information</p> <p>Description. Copying of data distribution information to the restarted node has been completed.</p>	<p>Event Name. NR_CopyDistr</p> <p>Event Type. NodeRestart</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Node restart starting to copy the fragments to Node <i>node_id</i></p> <p>Description. Copy of fragments to starting data node <i>node_id</i> has begun</p>	<p>Event Name. NR_CopyFragStarted</p> <p>Event Type. NodeRestart</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Table ID = <i>table_id</i>, fragment ID = <i>fragment_id</i> have been copied to Node <i>node_id</i></p> <p>Description. Fragment <i>fragment_id</i> from table <i>table_id</i> has been copied to data node <i>node_id</i></p>	<p>Event Name. NR_CopyFragDone</p> <p>Event Type. NodeRestart</p> <p>Priority. 10</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Node restart completed copying the fragments to Node <i>node_id</i></p>	<p>Event Name. NR_CopyFragCompleted</p> <p>Event Type. NodeRestart</p>

<p>Description. Copying of all table fragments to restarting data node <i>node_id</i> has been completed</p>	<p>Priority. 8 Severity. INFO</p>
<p>Log Message. Any of the following:</p> <ol style="list-style-type: none"> 1. Node <i>node_id</i>: Node <i>node1_id</i> completed failure of Node <i>node2_id</i> 2. All nodes completed failure of Node <i>node_id</i> 3. Node failure of <i>node_id</i>block completed <p>Description. One of the following (each corresponding to the same-numbered message listed above):</p> <ol style="list-style-type: none"> 1. Data node <i>node1_id</i> has detected the failure of data node <i>node2_id</i> 2. All (remaining) data nodes have detected the failure of data node <i>node_id</i> 3. The failure of data node <i>node_id</i> has been detected in the <i>block</i>NDB kernel block, where block is 1 of DBTC, DBDICT, DBDIH, or DBLQH; for more information, see NDB Kernel Blocks 	<p>Event Name. NodeFailCompleted Event Type. NodeRestart Priority. 8 Severity. ALERT</p>
<p>Log Message. Node <i>mgm_node_id</i>: Node <i>data_node_id</i> has failed. The Node state at failure was <i>state_code</i></p> <p>Description. A data node has failed. Its state at the time of failure is described by an arbitration state code <i>state_code</i>: possible state code values can be found in the file <code>include/kernel/signaldata/ArbitSignalData.hpp</code>.</p>	<p>Event Name. NODE_FAILREP Event Type. NodeRestart Priority. 8 Severity. ALERT</p>
<p>Log Message. President restarts arbitration thread [<i>state=state_code</i>] or Prepare arbitrator node <i>node_id</i> [<i>ticket=ticket_id</i>] or Receive arbitrator node <i>node_id</i> [<i>ticket=ticket_id</i>] or Started arbitrator node <i>node_id</i> [<i>ticket=ticket_id</i>] or Lost arbitrator node <i>node_id</i> - process failure [<i>state=state_code</i>] or Lost arbitrator node <i>node_id</i> - process exit [<i>state=state_code</i>] or Lost arbitrator node <i>node_id</i> - <i>error_message</i> [<i>state=state_code</i>]</p> <p>Description. This is a report on the current state and progress of arbitration in the cluster. <i>node_id</i> is the node ID of the management node or SQL node selected as the arbitrator. <i>state_code</i> is an arbitration state code, as found in <code>include/kernel/signaldata/ArbitSignalData.hpp</code>. When an error has occurred, an <i>error_message</i>, also defined in <code>ArbitSignalData.hpp</code>, is provided. <i>ticket_id</i> is a unique identifier handed out by the arbitrator when it is selected to all the nodes that participated in its selection; this is</p>	<p>Event Name. ArbitState Event Type. NodeRestart Priority. 6 Severity. INFO</p>

<p>used to ensure that each node requesting arbitration was one of the nodes that took part in the selection process.</p>	
<p>Log Message. Arbitration check lost - less than 1/2 nodes left or Arbitration check won - all node groups and more than 1/2 nodes left or Arbitration check won - node group majority or Arbitration check lost - missing node group or Network partitioning - arbitration required or Arbitration won - positive reply from node <i>node_id</i> or Arbitration lost - negative reply from node <i>node_id</i> or Network partitioning - no arbitrator available or Network partitioning - no arbitrator configured or Arbitration failure - <i>error_message</i> [state=<i>state_code</i>]</p> <p>Description. This message reports on the result of arbitration. In the event of arbitration failure, an <i>error_message</i> and an arbitration <i>state_code</i> are provided; definitions for both of these are found in <code>include/kernel/signaldata/ArbitSignalData.hpp</code>.</p>	<p>Event Name. ArbitResult</p> <p>Event Type. NodeRestart</p> <p>Priority. 2</p> <p>Severity. ALERT</p>
<p>Log Message. Node <i>node_id</i>: GCP Take over started</p> <p>Description. This node is attempting to assume responsibility for the next global checkpoint (that is, it is becoming the master node)</p>	<p>Event Name. GCP_TakeoverStarted</p> <p>Event Type. NodeRestart</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: GCP Take over completed</p> <p>Description. This node has become the master, and has assumed responsibility for the next global checkpoint</p>	<p>Event Name. GCP_TakeoverCompleted</p> <p>Event Type. NodeRestart</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: LCP Take over started</p> <p>Description. This node is attempting to assume responsibility for the next set of local checkpoints (that is, it is becoming the master node)</p>	<p>Event Name. LCP_TakeoverStarted</p> <p>Event Type. NodeRestart</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: LCP Take over completed</p> <p>Description. This node has become the master, and has assumed responsibility for the next set of local checkpoints</p>	<p>Event Name. LCP_TakeoverCompleted</p> <p>Event Type. NodeRestart</p> <p>Priority. 7</p> <p>Severity. INFO</p>

<p>Log Message. Node <i>node_id</i>: Trans. Count = <i>transactions</i>, Commit Count = <i>commits</i>, Read Count = <i>reads</i>, Simple Read Count = <i>simple_reads</i>, Write Count = <i>writes</i>, AttrInfo Count = <i>AttrInfo_objects</i>, Concurrent Operations = <i>concurrent_operations</i>, Abort Count = <i>aborts</i>, Scans = <i>scans</i>, Range scans = <i>range_scans</i></p> <p>Description. This report of transaction activity is given approximately once every 10 seconds</p>	<p>Event Name. TransReportCounters</p> <p>Event Type. Statistic</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Operations=<i>operations</i></p> <p>Description. Number of operations performed by this node, provided approximately once every 10 seconds</p>	<p>Event Name. OperationReportCounters</p> <p>Event Type. Statistic</p> <p>Priority. 8</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Table with ID = <i>table_id</i> created</p> <p>Description. A table having the table ID shown has been created</p>	<p>Event Name. TableCreated</p> <p>Event Type. Statistic</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Mean loop Counter in doJob last 8192 times = <i>count</i></p> <p>Description.</p>	<p>Event Name. JobStatistic</p> <p>Event Type. Statistic</p> <p>Priority. 9</p> <p>Severity. INFO</p>
<p>Log Message. Mean send size to Node = <i>node_id</i> last 4096 sends = <i>bytes</i> bytes</p> <p>Description. This node is sending an average of <i>bytes</i> bytes per send to node <i>node_id</i></p>	<p>Event Name. SendBytesStatistic</p> <p>Event Type. Statistic</p> <p>Priority. 9</p> <p>Severity. INFO</p>
<p>Log Message. Mean receive size to Node = <i>node_id</i> last 4096 sends = <i>bytes</i> bytes</p> <p>Description. This node is receiving an average of <i>bytes</i> of data each time it receives data from node <i>node_id</i></p>	<p>Event Name. ReceiveBytesStatistic</p> <p>Event Type. Statistic</p> <p>Priority. 9</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Data usage is <i>data_memory_percentage%</i> (<i>data_pages_used</i> 32K pages of total <i>data_pages_total</i>)/Node <i>node_id</i>: Index usage is <i>index_memory_percentage%</i> (<i>index_pages_used</i> 8K pages of total <i>index_pages_total</i>)</p>	<p>Event Name. MemoryUsage</p> <p>Event Type. Statistic</p> <p>Priority. 5</p>

MySQL Cluster Log Messages

<p>Description. This report is generated when a <code>DUMP 1000</code> command is issued in the cluster management client; for more information, see <code>DUMP 1000</code>, in MySQL Cluster Internals</p>	<p>Severity. INFO</p>
<p>Log Message. Node <code>node1_id</code>: Transporter to node <code>node2_id</code> reported error <code>error_code</code>: <code>error_message</code></p> <p>Description. A transporter error occurred while communicating with node <code>node2_id</code>; for a listing of transporter error codes and messages, see NDB Transporter Errors, in MySQL Cluster Internals</p>	<p>Event Name. TransporterError</p> <p>Event Type. Error</p> <p>Priority. 2</p> <p>Severity. ERROR</p>
<p>Log Message. Node <code>node1_id</code>: Transporter to node <code>node2_id</code> reported error <code>error_code</code>: <code>error_message</code></p> <p>Description. A warning of a potential transporter problem while communicating with node <code>node2_id</code>; for a listing of transporter error codes and messages, see NDB Transporter Errors, for more information</p>	<p>Event Name. TransporterWarning</p> <p>Event Type. Error</p> <p>Priority. 8</p> <p>Severity. WARNING</p>
<p>Log Message. Node <code>node1_id</code>: Node <code>node2_id</code> missed heartbeat <code>heartbeat_id</code></p> <p>Description. This node missed a heartbeat from node <code>node2_id</code></p>	<p>Event Name. MissedHeartbeat</p> <p>Event Type. Error</p> <p>Priority. 8</p> <p>Severity. WARNING</p>
<p>Log Message. Node <code>node1_id</code>: Node <code>node2_id</code> declared dead due to missed heartbeat</p> <p>Description. This node has missed at least 3 heartbeats from node <code>node2_id</code>, and so has declared that node “dead”</p>	<p>Event Name. DeadDueToHeartbeat</p> <p>Event Type. Error</p> <p>Priority. 8</p> <p>Severity. ALERT</p>
<p>Log Message. Node <code>node1_id</code>: Node Sent Heartbeat to node = <code>node2_id</code></p> <p>Description. This node has sent a heartbeat to node <code>node2_id</code></p>	<p>Event Name. SentHeartbeat</p> <p>Event Type. Info</p> <p>Priority. 12</p> <p>Severity. INFO</p>
<p>Log Message. Node <code>node_id</code>: Event buffer status: <code>used=bytes_used (percent_used%) alloc=bytes_allocated (percent_available%) max=bytes_available apply_gci=latest_restorable_GCI latest_gci=latest_GCI</code></p> <p>Description. This report is seen during heavy event buffer usage, for example, when many updates are being applied in a relatively short period of time; the report shows the number of bytes and the percentage of event buffer memory used, the bytes allocated and percentage still available, and the latest and latest restorable global checkpoints</p>	<p>Event Name. EventBufferStatus</p> <p>Event Type. Info</p> <p>Priority. 7</p> <p>Severity. INFO</p>

<p>Log Message. Node <i>node_id</i>: Entering single user mode, Node <i>node_id</i>: Entered single user mode Node <i>API_node_id</i> has exclusive access, Node <i>node_id</i>: Entering single user mode</p> <p>Description. These reports are written to the cluster log when entering and exiting single user mode; <i>API_node_id</i> is the node ID of the API or SQL having exclusive access to the cluster (for more information, see Section 17.5.8, “MySQL Cluster Single User Mode”); the message <code>Unknown single user report API_node_id</code> indicates an error has taken place and should never be seen in normal operation</p>	<p>Event Name. SingleUser</p> <p>Event Type. Info</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Backup <i>backup_id</i> started from node <i>mgm_node_id</i></p> <p>Description. A backup has been started using the management node having <i>mgm_node_id</i>; this message is also displayed in the cluster management client when the <code>START BACKUP</code> command is issued; for more information, see Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”</p>	<p>Event Name. BackupStarted</p> <p>Event Type. Backup</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Backup <i>backup_id</i> started from node <i>mgm_node_id</i> completed. StartGCP: <i>start_gcp</i> StopGCP: <i>stop_gcp</i> #Records: <i>records</i> #LogRecords: <i>log_records</i> Data: <i>data_bytes</i> bytes Log: <i>log_bytes</i> bytes</p> <p>Description. The backup having the ID <i>backup_id</i> has been completed; for more information, see Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”</p>	<p>Event Name. BackupCompleted</p> <p>Event Type. Backup</p> <p>Priority. 7</p> <p>Severity. INFO</p>
<p>Log Message. Node <i>node_id</i>: Backup request from <i>mgm_node_id</i> failed to start. Error: <i>error_code</i></p> <p>Description. The backup failed to start; for error codes, see MGM API Errors</p>	<p>Event Name. BackupFailedToStart</p> <p>Event Type. Backup</p> <p>Priority. 7</p> <p>Severity. ALERT</p>
<p>Log Message. Node <i>node_id</i>: Backup <i>backup_id</i> started from <i>mgm_node_id</i> has been aborted. Error: <i>error_code</i></p> <p>Description. The backup was terminated after starting, possibly due to user intervention</p>	<p>Event Name. BackupAborted</p> <p>Event Type. Backup</p> <p>Priority. 7</p> <p>Severity. ALERT</p>

17.5.7.2. MySQL Cluster: NDB Transporter Errors

This section lists error codes, names, and messages that are written to the cluster log in the event of transporter errors.

MySQL Cluster Log Messages

Error Code	Error Name	Error Text
0x00	TE_NO_ERROR	No error
0x01	TE_ERROR_CLOSING_SOCKET	Error found during closing of socket
0x02	TE_ERROR_IN_SELECT_BEFORE_ACCEPT	Error found before accept. The transporter will retry
0x03	TE_INVALID_MESSAGE_LENGTH	Error found in message (invalid message length)
0x04	TE_INVALID_CHECKSUM	Error found in message (checksum)
0x05	TE_COULD_NOT_CREATE_SOCKET	Error found while creating socket (can't create socket)
0x06	TE_COULD_NOT_BIND_SOCKET	Error found while binding server socket
0x07	TE_LISTEN_FAILED	Error found while listening to server socket
0x08	TE_ACCEPT_RETURN_ERROR	Error found during accept (accept return error)
0x0b	TE_SHM_DISCONNECT	The remote node has disconnected
0x0c	TE_SHM_IPC_STAT	Unable to check shm segment
0x0d	TE_SHM_UNABLE_TO_CREATE_SEGMENT	Unable to create shm segment
0x0e	TE_SHM_UNABLE_TO_ATTACH_SEGMENT	Unable to attach shm segment
0x0f	TE_SHM_UNABLE_TO_REMOVE_SEGMENT	Unable to remove shm segment
0x10	TE_TOO_SMALL_SIGID	Sig ID too small
0x11	TE_TOO_LARGE_SIGID	Sig ID too large
0x12	TE_WAIT_STACK_FULL	Wait stack was full
0x13	TE_RECEIVE_BUFFER_FULL	Receive buffer was full
0x14	TE_SIGNAL_LOST_SEND_BUFFER_FULL	Send buffer was full, and trying to force send fails
0x15	TE_SIGNAL_LOST	Send failed for unknown reason (signal lost)

Error Code	Error Name	Error Text
0x16	TE_SEND_BUFFER_FULL	The send buffer was full, but sleeping for a while solved
0x0017	TE_SCI_LINK_ERROR	There is no link from this node to the switch
0x18	TE_SCI_UNABLE_TO_START_SEQUENCE	Could not start a sequence, because system resources are exumed or no sequence has been created
0x19	TE_SCI_UNABLE_TO_REMOVE_SEQUENCE	Could not remove a sequence
0x1a	TE_SCI_UNABLE_TO_CREATE_SEQUENCE	Could not create a sequence, because system resources are exempted. Must reboot
0x1b	TE_SCI_UNRECOVERABLE_DATA_TFX_ERROR	Tried to send data on redundant link but failed
0x1c	TE_SCI_CANNOT_INIT_LOCALSEGMENT	Cannot initialize local segment
0x1d	TE_SCI_CANNOT_MAP_REMOTESEGMENT	Cannot map remote segment
0x1e	TE_SCI_UNABLE_TO_UNMAP_SEGMENT	Cannot free the resources used by this segment (step 1)
0x1f	TE_SCI_UNABLE_TO_REMOVE_SEGMENT	Cannot free the resources used by this segment (step 2)
0x20	TE_SCI_UNABLE_TO_DISCONNECT_SEGMENT	Cannot disconnect from a remote segment
0x21	TE_SHM_IPC_PERMANENT	Shm ipc Permanent error
0x22	TE_SCI_UNABLE_TO_CLOSE_CHANNEL	Unable to close the sci channel and the resources allocated

17.5.8. MySQL Cluster Single User Mode

Single user mode enables the database administrator to restrict access to the database system to a single API node, such as a MySQL server (SQL node) or an instance of `ndb_restore`. When entering single user mode, connections to all other API nodes are closed gracefully and all running transactions are aborted. No new transactions are permitted to start.

Once the cluster has entered single user mode, only the designated API node is granted access to the database.

You can use the `ALL STATUS` command in the `ndb_mgm` client to see when the cluster has entered single user mode. You can also check the `status` column of the `ndbinfo.nodes` table (see [Section 17.5.10.13, “The `ndbinfo nodes` Table”](#), for more information).

Example:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

After this command has executed and the cluster has entered single user mode, the API node whose node ID is 5 becomes the cluster's only permitted user.

The node specified in the preceding command must be an API node; attempting to specify any other type of node will be rejected.



Note

When the preceding command is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command `EXIT SINGLE USER MODE` changes the state of the cluster's data nodes from single user mode to normal mode. API nodes—such as MySQL Servers—waiting for a connection (that is, waiting for the cluster to become ready and available), are again permitted to connect. The API node denoted as the single-user node continues to run (if still connected) during and after the state change.

Example:

```
ndb_mgm> EXIT SINGLE USER MODE
```

There are two recommended ways to handle a node failure when running in single user mode:

- Method 1:
 1. Finish all single user mode transactions
 2. Issue the `EXIT SINGLE USER MODE` command
 3. Restart the cluster's data nodes
- Method 2:

Restart database nodes prior to entering single user mode.

17.5.9. Quick Reference: MySQL Cluster SQL Statements

This section discusses several SQL statements that can prove useful in managing and monitoring a MySQL server that is connected to a MySQL Cluster, and in some cases provide information about the cluster itself.

- `SHOW ENGINE NDB STATUS`, `SHOW ENGINE NDBCLUSTER STATUS`

The output of this statement contains information about the server's connection to the cluster, creation and usage of MySQL Cluster objects, and binary logging for MySQL Cluster replication.

See [Section 13.7.5.16, “`SHOW ENGINE Syntax`”](#), for a usage example and more detailed information.

- `SHOW ENGINES`

This statement can be used to determine whether or not clustering support is enabled in the MySQL server, and if so, whether it is active.

See [Section 13.7.5.17, “SHOW ENGINES Syntax”](#), for more detailed information.



Note

In MySQL 5.1 and later, this statement does not support a [LIKE \[1111\]](#) clause. However, you can use [LIKE \[1111\]](#) to filter queries against the [INFORMATION_SCHEMA.ENGINES](#), as discussed in the next item.

- `SELECT * FROM INFORMATION_SCHEMA.ENGINES [WHERE ENGINE LIKE 'NDB%']`

This is the equivalent of `SHOW ENGINES`, but uses the `ENGINES` table of the `INFORMATION_SCHEMA` database. Unlike the case with the `SHOW ENGINES` statement, it is possible to filter the results using a [LIKE \[1111\]](#) clause, and to select specific columns to obtain information that may be of use in scripts. For example, the following query shows whether the server was built with `NDB` support and, if so, whether it is enabled:

```
mysql> SELECT SUPPORT FROM INFORMATION_SCHEMA.ENGINES
-> WHERE ENGINE LIKE 'NDB%';
+-----+
| support |
+-----+
| ENABLED |
+-----+
```

See [Section 20.6, “The INFORMATION_SCHEMA ENGINES Table”](#), for more information.

- `SHOW VARIABLES LIKE 'NDB%'`

This statement provides a list of most server system variables relating to the `NDB` storage engine, and their values, as shown here:

```
mysql> SHOW VARIABLES LIKE 'NDB%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| ndb_autoincrement_prefetch_sz | 32 |
| ndb_cache_check_time | 0 |
| ndb_extra_logging | 0 |
| ndb_force_send | ON |
| ndb_index_stat_cache_entries | 32 |
| ndb_index_stat_enable | OFF |
| ndb_index_stat_update_freq | 20 |
| ndb_report_thresh_binlog_epoch_slip | 3 |
| ndb_report_thresh_binlog_mem_usage | 10 |
| ndb_use_copying_alter_table | OFF |
| ndb_use_exact_count | ON |
| ndb_use_transactions | ON |
+-----+-----+
```

See [Section 5.1.4, “Server System Variables”](#), for more information.

- `SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES WHERE VARIABLE_NAME LIKE 'NDB%'`

This statement is the equivalent of the `SHOW` command described in the previous item, and provides almost identical output, as shown here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
```

```
-> WHERE VARIABLE_NAME LIKE 'NDB%';
```

VARIABLE_NAME	VARIABLE_VALUE
NDB_AUTOINCREMENT_PREFETCH_SZ	32
NDB_CACHE_CHECK_TIME	0
NDB_EXTRA_LOGGING	0
NDB_FORCE_SEND	ON
NDB_INDEX_STAT_CACHE_ENTRIES	32
NDB_INDEX_STAT_ENABLE	OFF
NDB_INDEX_STAT_UPDATE_FREQ	20
NDB_REPORT_THRESH_BINLOG_EPOCH_SLIP	3
NDB_REPORT_THRESH_BINLOG_MEM_USAGE	10
NDB_USE_COPYING_ALTER_TABLE	OFF
NDB_USE_EXACT_COUNT	ON
NDB_USE_TRANSACTIONS	ON

Unlike the case with the `SHOW` command, it is possible to select individual columns. For example:

```
mysql> SELECT VARIABLE_VALUE
-> FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
-> WHERE VARIABLE_NAME = 'ndb_force_send';
```

VARIABLE_VALUE
ON

See [Section 20.9, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”](#), and [Section 5.1.4, “Server System Variables”](#), for more information.

- `SHOW STATUS LIKE 'NDB%'`

This statement shows at a glance whether or not the MySQL server is acting as a cluster SQL node, and if so, it provides the MySQL server's cluster node ID, the host name and port for the cluster management server to which it is connected, and the number of data nodes in the cluster, as shown here:

```
mysql> SHOW STATUS LIKE 'NDB%';
```

Variable_name	Value
Ndb_cluster_node_id	10
Ndb_config_from_host	192.168.0.103
Ndb_config_from_port	1186
Ndb_number_of_data_nodes	4

If the MySQL server was built with clustering support, but it is not connected to a cluster, all rows in the output of this statement contain a zero or an empty string:

```
mysql> SHOW STATUS LIKE 'NDB%';
```

Variable_name	Value
Ndb_cluster_node_id	0
Ndb_config_from_host	
Ndb_config_from_port	0
Ndb_number_of_data_nodes	0

See also [Section 13.7.5.36, “SHOW STATUS Syntax”](#).

- `SELECT * FROM INFORMATION_SCHEMA.GLOBAL_STATUS WHERE VARIABLE_NAME LIKE 'NDB %';`

This statement provides similar output to the `SHOW` command discussed in the previous item. However, unlike the case with `SHOW STATUS`, it is possible using the `SELECT` to extract values in SQL for use in scripts for monitoring and automation purposes.

See [Section 20.8, “The `INFORMATION_SCHEMA.GLOBAL_STATUS` and `SESSION_STATUS` Tables”](#), for more information.

You can also query the tables in the `ndbinfo` information database for real-time data about many MySQL Cluster operations. See [Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”](#).

17.5.10. The `ndbinfo` MySQL Cluster Information Database

`ndbinfo` is a database storing containing information specific to MySQL Cluster.

This database contains a number of tables, each providing a different sort of data about MySQL Cluster node status, resource usage, and operations. You can find more detailed information about each of these tables in the next several sections.

`ndbinfo` is included with MySQL Cluster support in the MySQL Server; no special compilation or configuration steps are required; the tables are created by the MySQL Server when it connects to the cluster. You can verify that `ndbinfo` support is active in a given MySQL Server instance using `SHOW PLUGINS`; if `ndbinfo` support is enabled, you should see a row containing `ndbinfo` in the `Name` column and `ACTIVE` in the `Status` column, as shown here (emphasized text):

```
mysql> SHOW PLUGINS;
+-----+-----+-----+-----+-----+-----+
| Name                               | Status | Type                | Library | License |
+-----+-----+-----+-----+-----+-----+
| binlog                             | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| mysql_native_password              | ACTIVE | AUTHENTICATION     | NULL   | GPL     |
| mysql_old_password                 | ACTIVE | AUTHENTICATION     | NULL   | GPL     |
| CSV                                | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| MEMORY                             | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| MRG_MYISAM                         | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| MyISAM                             | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| PERFORMANCE_SCHEMA                | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| BLACKHOLE                          | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| ARCHIVE                            | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| ndbcluster                         | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| ndbinfo                           | ACTIVE | STORAGE ENGINE   | NULL   | GPL     |
| ndb_transid_mysql_connection_map   | ACTIVE | INFORMATION SCHEMA | NULL   | GPL     |
| InnoDB                             | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
| INNODB_TRX                        | ACTIVE | INFORMATION SCHEMA | NULL   | GPL     |
| INNODB_LOCKS                      | ACTIVE | INFORMATION SCHEMA | NULL   | GPL     |
| INNODB_LOCK_WAITS                 | ACTIVE | INFORMATION SCHEMA | NULL   | GPL     |
| INNODB_CMP                        | ACTIVE | INFORMATION SCHEMA | NULL   | GPL     |
| INNODB_CMP_RESET                  | ACTIVE | INFORMATION SCHEMA | NULL   | GPL     |
| INNODB_CMPMEM                    | ACTIVE | INFORMATION SCHEMA | NULL   | GPL     |
| INNODB_CMPMEM_RESET               | ACTIVE | INFORMATION SCHEMA | NULL   | GPL     |
| partition                         | ACTIVE | STORAGE ENGINE     | NULL   | GPL     |
+-----+-----+-----+-----+-----+-----+
22 rows in set (0.00 sec)
```

You can also do this by checking the output of `SHOW ENGINES` for a line including `ndbinfo` in the `Engine` column and `YES` in the `Support` column, as shown here (emphasized text):

```
mysql> SHOW ENGINES\G
***** 1. row *****
      Engine: ndbcluster
      Support: YES
```

```

    Comment: Clustered, fault-tolerant tables
Transactions: YES
    XA: NO
    Savepoints: NO
***** 2. row *****
    Engine: MRG_MYISAM
    Support: YES
    Comment: Collection of identical MyISAM tables
Transactions: NO
    XA: NO
    Savepoints: NO
***** 3. row *****
    Engine: ndbinfo
    Support: YES
    Comment: MySQL Cluster system information storage engine
Transactions: NO
    XA: NO
    Savepoints: NO
***** 4. row *****
    Engine: CSV
    Support: YES
    Comment: CSV storage engine
Transactions: NO
    XA: NO
    Savepoints: NO
***** 5. row *****
    Engine: MEMORY
    Support: YES
    Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
    XA: NO
    Savepoints: NO
***** 6. row *****
    Engine: FEDERATED
    Support: NO
    Comment: Federated MySQL storage engine
Transactions: NULL
    XA: NULL
    Savepoints: NULL
***** 7. row *****
    Engine: ARCHIVE
    Support: YES
    Comment: Archive storage engine
Transactions: NO
    XA: NO
    Savepoints: NO
***** 8. row *****
    Engine: InnoDB
    Support: YES
    Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
    XA: YES
    Savepoints: YES
***** 9. row *****
    Engine: MyISAM
    Support: DEFAULT
    Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
    XA: NO
    Savepoints: NO
***** 10. row *****
    Engine: BLACKHOLE
    Support: YES
    Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
    XA: NO
    Savepoints: NO

```

```
10 rows in set (0.00 sec)
```

If `ndbinfo` support is enabled, then you can access `ndbinfo` using SQL statements in `mysql` or another MySQL client. For example, you can see `ndbinfo` listed in the output of `SHOW DATABASES`, as shown here:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| ndbinfo |
| test |
+-----+
4 rows in set (0.00 sec)
```

If the `mysqld` process was not started with the `--ndbcluster` [2142] option, `ndbinfo` is not available and is not displayed by `SHOW DATABASES`. If `mysqld` was formerly connected to a MySQL Cluster but the cluster becomes unavailable (due to events such as cluster shutdown, loss of network connectivity, and so forth), `ndbinfo` and its tables remain visible, but an attempt to access any tables (other than `blocks` or `config_params`) fails with `Got error 157 'Connection to NDB failed' from NDBINFO`.

With the exception of the `blocks` and `config_params` tables, what we refer to as `ndbinfo` “tables” are actually views generated from internal `NDB` tables not normally visible to the MySQL Server.

All `ndbinfo` tables are read-only, and are generated on demand when queried. Because many of them are generated in parallel by the data nodes while other are specific to a given SQL node, they are not guaranteed to provide a consistent snapshot.

In addition, pushing down of joins is not supported on `ndbinfo` tables; so joining large `ndbinfo` tables can require transfer of a large amount of data to the requesting API node, even when the query makes use of a `WHERE` clause.

`ndbinfo` tables are not included in the query cache. (Bug #59831)

You can select the `ndbinfo` database with a `USE` statement, and then issue a `SHOW TABLES` statement to obtain a list of tables, just as for any other database, like this:

```
mysql> USE ndbinfo;
Database changed

mysql> SHOW TABLES;
+-----+
| Tables_in_ndbinfo |
+-----+
| blocks |
| cluster_operations |
| cluster_transactions |
| config_params |
| counters |
| diskpagebuffer |
| logbuffers |
| logspaces |
| memoryusage |
| nodes |
| resources |
| server_operations |
| server_transactions |
| threadblocks |
| threadstat |
| transporters |
+-----+
16 rows in set (0.04 sec)
```

The `cluster_operations`, `cluster_transactions`, `server_operations`, `server_transactions`, `threadblocks`, and `threadstat` tables were added in MySQL Cluster NDB 7.2.2.

You can execute `SELECT` statements against these tables, just as you would normally expect:

```
mysql> SELECT * FROM memoryusage;
+-----+-----+-----+-----+
| node_id | DATA_MEMORY | used | max |
+-----+-----+-----+-----+
|      1 | DATA_MEMORY | 3230 | 6408 |
|      2 | DATA_MEMORY | 3230 | 6408 |
|      1 | INDEX_MEMORY | 16   | 12832 |
|      2 | INDEX_MEMORY | 16   | 12832 |
+-----+-----+-----+-----+
4 rows in set (0.37 sec)
```

More complex queries, such as the two following `SELECT` statements using the `memoryusage` table, are possible:

```
mysql> SELECT SUM(used) as 'Data Memory Used, All Nodes'
> FROM memoryusage
> WHERE DATA_MEMORY = 'DATA_MEMORY';
+-----+-----+
| Data Memory Used, All Nodes |
+-----+-----+
|                          6460 |
+-----+-----+
1 row in set (0.37 sec)

mysql> SELECT SUM(max) as 'Total IndexMemory Available'
> FROM memoryusage
> WHERE DATA_MEMORY = 'INDEX_MEMORY';
+-----+-----+
| Total IndexMemory Available |
+-----+-----+
|                          25664 |
+-----+-----+
1 row in set (0.33 sec)
```

`ndbinfo` table and column names are case sensitive (as is the name of the `ndbinfo` database itself). These identifiers are in lowercase. Trying to use the wrong lettercase results in an error, as shown in this example:

```
mysql> SELECT * FROM nodes;
+-----+-----+-----+-----+
| node_id | uptime | status | start_phase |
+-----+-----+-----+-----+
|      1 | 13602 | STARTED |          0 |
|      2 | 16    | STARTED |          0 |
+-----+-----+-----+-----+
2 rows in set (0.04 sec)

mysql> SELECT * FROM Nodes;
ERROR 1146 (42S02): Table 'ndbinfo.Nodes' doesn't exist
```

`mysqldump` ignores the `ndbinfo` database entirely, and excludes it from any output. This is true even when using the `--databases` [314] or `--all-databases` [313] option.

MySQL Cluster also maintains tables in the `INFORMATION_SCHEMA` information database, including the `FILES` table which contains information about files used for MySQL Cluster Disk Data storage. For more information, see [Section 20.29, “INFORMATION_SCHEMA Tables for MySQL Cluster”](#).

17.5.10.1. The `ndbinfo` `arbiterator_validity_detail` Table

The `arbitrator_validity_detail` table shows the view that each data node in the cluster of the arbitrator. It is a subset of the `membership` table.

The following table provides information about the columns in the `arbitrator_validity_detail` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	This node's node ID
<code>arbitrator</code>	integer	Node ID of arbitrator
<code>arb_ticket</code>	string	Internal identifier used to track arbitration
<code>arb_connected</code>	Yes or No	Whether this node is connected to the arbitrator
<code>arb_state</code>	Enumeration (see text)	Arbitration state

The node ID is the same as that reported by `ndb_mgm -e "SHOW"`.

All nodes should show the same `arbitrator` and `arb_ticket` values as well as the same `arb_state` value. Possible `arb_state` values are `ARBIT_NULL`, `ARBIT_INIT`, `ARBIT_FIND`, `ARBIT_PREP1`, `ARBIT_PREP2`, `ARBIT_START`, `ARBIT_RUN`, `ARBIT_CHOOSE`, `ARBIT_CRASH`, and `UNKNOWN`.

`arb_connected` shows whether the current node is connected to the `arbitrator`.

Like the `membership` and `arbitrator_validity_summary` tables, this table was added in MySQL Cluster NDB 7.2.10.

17.5.10.2. The `ndbinfo` `arbitrator_validity_summary` Table

The `arbitrator_validity_summary` table provides a composite view of the arbitrator with regard to the cluster's data nodes.

The following table provides information about the columns in the `arbitrator_validity_detail` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>arbitrator</code>	integer	Node ID of arbitrator
<code>arb_ticket</code>	string	Internal identifier used to track arbitration
<code>arb_connected</code>	Yes or No	Whether this arbitrator is connected to the cluster
<code>consensus_count</code>	integer	Number of data nodes that see this node as arbitrator

In normal operations, this table should have only 1 row for any appreciable length of time. If it has more than 1 row for longer than a few moments, then either not all nodes are connected to the arbitrator, or all nodes are connected, but do not agree on the same arbitrator.

The `arbitrator` column shows the arbitrator's node ID.

`arb_ticket` is the internal identifier used by this arbitrator.

`arb_connected` shows whether this node is connected to the cluster as an arbitrator.

Like the `membership` and `arbitrator_validity_detail` tables, this table was added in MySQL Cluster NDB 7.2.10.

17.5.10.3. The `ndbinfo` `blocks` Table

The `blocks` table is a static table which simply contains the names and internal IDs of all NDB kernel blocks (see [NDB Kernel Blocks](#)). It is for use by the other `ndbinfo` tables (most of which are actually views) in mapping block numbers to block names for producing human-readable output.

The following table provides information about the columns in the `blocks` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>block_number</code>	integer	Block number
<code>block_name</code>	string	Block name

To obtain a list of all block names, simply execute `SELECT block_name FROM ndbinfo.blocks`. Although this is a static table, its content can vary between different MySQL Cluster releases.

17.5.10.4. The `ndbinfo` `cluster_operations` Table

The `cluster_operations` table provides a per-operation (stateful primary key op) view of all activity in the MySQL Cluster from the point of view of the local data management (LQH) blocks (see [The DBLQH Block](#)).

The following table provides information about the columns in the `cluster_operations` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	Node ID of reporting LQH block
<code>block_instance</code>	integer	LQH block instance
<code>transid</code>	integer	Transaction ID
<code>operation_type</code>	string	Operation type (see text for possible values)
<code>state</code>	string	Operation state (see text for possible values)
<code>tableid</code>	integer	Table ID
<code>fragmentid</code>	integer	Fragment ID
<code>client_node_id</code>	integer	Client node ID
<code>client_block_ref</code>	integer	Client block reference
<code>tc_node_id</code>	integer	Transaction coordinator node ID
<code>tc_block_no</code>	integer	Transaction coordinator block number
<code>tc_block_instance</code>	integer	Transaction coordinator block instance

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `operation_type` column can take any one of the values `READ`, `READ-SH`, `READ-EX`, `INSERT`, `UPDATE`, `DELETE`, `WRITE`, `UNLOCK`, `REFRESH`, `SCAN`, `SCAN-SH`, `SCAN-EX`, or `<unknown>`.

The `state` column can have any one of the values `ABORT_QUEUED`, `ABORT_STOPPED`, `COMMITTED`, `COMMIT_QUEUED`, `COMMIT_STOPPED`, `COPY_CLOSE_STOPPED`,

`COPY_FIRST_STOPPED`, `COPY_STOPPED`, `COPY_TUPKEY`, `IDLE`, `LOG_ABORT_QUEUED`, `LOG_COMMIT_QUEUED`, `LOG_COMMIT_QUEUED_WAIT_SIGNAL`, `LOG_COMMIT_WRITTEN`, `LOG_COMMIT_WRITTEN_WAIT_SIGNAL`, `LOG_QUEUED`, `PREPARED`, `PREPARED_RECEIVED_COMMIT`, `SCAN_CHECK_STOPPED`, `SCAN_CLOSE_STOPPED`, `SCAN_FIRST_STOPPED`, `SCAN_RELEASE_STOPPED`, `SCAN_STATE_USED`, `SCAN_STOPPED`, `SCAN_TUPKEY`, `STOPPED`, `TC_NOT_CONNECTED`, `WAIT_ACC`, `WAIT_ACC_ABORT`, `WAIT_AI_AFTER_ABORT`, `WAIT_ATTR`, `WAIT_SCAN_AI`, `WAIT_TUP`, `WAIT_TUPKEYINFO`, `WAIT_TUP_COMMIT`, or `WAIT_TUP_TO_ABORT`. (If the MySQL Server is running with `ndbinfo_show_hidden` [2159] enabled, you can view this list of states by selecting from the `ndb $dbldqh_tcconnect_state` table, which is normally hidden.)

You can obtain the name of an NDB table from its table ID by checking the output of `ndb_show_tables`.

The `fragid` is the same as the partition number seen in the output of `ndb_desc --extra-partition-info` [2207] (short form `-p`).

In `client_node_id` and `client_block_ref`, `client` refers to a MySQL Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

This table was added in MySQL Cluster NDB 7.2.2.

17.5.10.5. The `ndbinfo cluster_transactions` Table

The `cluster_transactions` table shows information about all ongoing transactions in a MySQL Cluster.

The following table provides information about the columns in the `cluster_transactions` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	Node ID of transaction coordinator
<code>block_instance</code>	integer	TC block instance
<code>transid</code>	integer	Transaction ID
<code>state</code>	string	Operation state (see text for possible values)
<code>count_operations</code>	integer	Number of stateful primary key operations in transaction (includes reads with locks, as well as DML operations)
<code>outstanding_operations</code>	integer	Operations still being executed in local data management blocks
<code>inactive_seconds</code>	integer	Time spent waiting for API
<code>client_node_id</code>	integer	Client node ID
<code>client_block_ref</code>	integer	Client block reference

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `state` column can have any one of the values `CS_ABORTING`, `CS_COMMITTING`, `CS_COMMIT_SENT`, `CS_COMPLETE_SENT`, `CS_COMPLETING`, `CS_CONNECTED`, `CS_DISCONNECTED`, `CS_FAIL_ABORTED`, `CS_FAIL_ABORTING`, `CS_FAIL_COMMITTED`, `CS_FAIL_COMMITTING`, `CS_FAIL_COMPLETED`, `CS_FAIL_PREPARED`, `CS_PREPARE_TO_COMMIT`, `CS_RECEIVING`, `CS_REC_COMMITTING`, `CS_RESTART`, `CS_SEND_FIRE_TRIG_REQ`, `CS_STARTED`, `CS_START_COMMITTING`, `CS_START_SCAN`, `CS_WAIT_ABORT_CONF`, `CS_WAIT_COMMIT_CONF`, `CS_WAIT_COMPLETE_CONF`,

`CS_WAIT_FIRE_TRIG_REQ`. (If the MySQL Server is running with `ndbinfo_show_hidden` [2159] enabled, you can view this list of states by selecting from the `ndb$dbtbc_apiconnect_state` table, which is normally hidden.)

In `client_node_id` and `client_block_ref`, `client` refers to a MySQL Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

This table was added in MySQL Cluster NDB 7.2.2.

17.5.10.6. The `ndbinfo config_params` Table

The `config_params` table is a static table which provides the names and internal ID numbers of all MySQL Cluster configuration parameters.

The following table provides information about the columns in the `config_params` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>param_number</code>	integer	The parameter's internal ID number
<code>param_name</code>	string	The name of the parameter

Although this is a static table, its content can vary between MySQL Cluster installations, since supported parameters can vary due to differences between software releases, cluster hardware configurations, and other factors.

17.5.10.7. The `ndbinfo counters` Table

The `counters` table provides running totals of events such as reads and writes for specific kernel blocks and data nodes. Counts are kept from the most recent node start or restart; a node start or restart resets all counters on that node. Not all kernel blocks have all types of counters.

The following table provides information about the columns in the `counters` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	The data node ID
<code>block_name</code>	string	Name of the associated NDB kernel block (see NDB Kernel Blocks).
<code>block_instance</code>	integer	Block instance
<code>counter_id</code>	integer	The counter's internal ID number; normally an integer between 1 and 10, inclusive.
<code>counter_name</code>	string	The name of the counter. See text for names of individual counters and the NDB kernel block with which each counter is associated.
<code>val</code>	integer	The counter's value

Each counter is associated with a particular NDB kernel block. Prior to MySQL Cluster NDB 7.2.0, this was limited to either the `DBLQH` kernel block or the `DBTC` kernel block.

The `OPERATIONS` counter is associated with the `DBLQH` (local query handler) kernel block (see [The DBLQH Block](#)).

The `ATTRINFO`, `TRANSACTIONS`, `COMMITTS`, `READS`, `SIMPLE_READS`, `WRITES`, `ABORTS`, `TABLE_SCANS`, and `RANGE_SCANS` counters are associated with the DBTC (transaction co-ordinator) kernel block (see [The DBTC Block](#)).

MySQL Cluster NDB 7.2.0, as part of its implementation of distributed pushed-down joins, adds the `LOCAL_TABLE_SCANS_SENT`, `READS_RECEIVED`, `PRUNED_RANGE_SCANS_RECEIVED`, `RANGE_SCANS_RECEIVED`, `LOCAL_READS_SENT`, `CONST_PRUNED_RANGE_SCANS_RECEIVED`, `LOCAL_RANGE_SCANS_SENT`, `REMOTE_READS_SENT`, `REMOTE_RANGE_SCANS_SENT`, `READS_NOT_FOUND`, `SCAN_BATCHES_RETURNED`, `TABLE_SCANS_RECEIVED`, and `SCAN_ROWS_RETURNED` counters. These counters are associated with the `DBSPJ` (select push-down join) kernel block (see [The DBSPJ Block](#)).

A number of counters increasing the visibility of transporter overload and send buffer sizing when troubleshooting such issues were added in MySQL Cluster NDB 7.2.10. (Bug #15935206) For each LQH instance, there is one instance of each counter in the following list:

- `LQHKEY_OVERLOAD`: Number of primary key requests rejected at the LQH block instance due to transporter overload
- `LQHKEY_OVERLOAD_TC`: Count of instances of `LQHKEY_OVERLOAD` where the TC node transporter was overloaded
- `LQHKEY_OVERLOAD_READER`: Count of instances of `LQHKEY_OVERLOAD` where the API reader (reads only) node was overloaded.
- `LQHKEY_OVERLOAD_NODE_PEER`: Count of instances of `LQHKEY_OVERLOAD` where the next backup data node (writes only) was overloaded
- `LQHKEY_OVERLOAD_SUBSCRIBER`: Count of instances of `LQHKEY_OVERLOAD` where a event subscriber (writes only) was overloaded.
- `LQHSCAN_SLOWDOWNS`: Count of instances where a fragment scan batch size was reduced due to scanning API transporter overload.

17.5.10.8. The `ndbinfo` `diskpagebuffer` Table

The `diskpagebuffer` table provides statistics about disk page buffer usage by MySQL Cluster Disk Data tables.

The following table provides information about the columns in the `diskpagebuffer` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	The data node ID
<code>block_instance</code>	integer	Block instance
<code>pages_written</code>	integer	Number of pages written to disk.
<code>pages_written_lcp</code>	integer	Number of pages written by local checkpoints.
<code>pages_read</code>	integer	Number of pages read from disk
<code>log_waits</code>	integer	Number of page writes waiting for log to be written to disk
<code>page_requests_direct_return</code>	integer	Number of requests for pages that were available in buffer

Column Name	Type	Remarks
<code>page_requests_wait_queue</code>	integer	Number of requests that had to wait for pages to become available in buffer
<code>page_requests_wait_io</code>	integer	Number of requests that had to be read from pages on disk (pages were unavailable in buffer)

You can use this table with MySQL Cluster Disk Data tables to determine whether `DiskPageBufferMemory` [2083] is sufficiently large to allow data to be read from the buffer rather than from disk; minimizing disk seeks can help improve performance of such tables.

You can determine the proportion of reads from `DiskPageBufferMemory` [2083] to the total number of reads using a query such as this one, which obtains this ratio as a percentage:

```
SELECT
  node_id,
  100 * page_requests_direct_return /
    (page_requests_direct_return + page_requests_wait_io)
    AS hit_ratio
FROM ndbinfo.diskpagebuffer;
```

The result from this query should be similar to what is shown here, with one row for each data node in the cluster (in this example, the cluster has 4 data nodes):

```
+-----+-----+
| node_id | hit_ratio |
+-----+-----+
|      5 |  97.6744 |
|      6 |  97.6879 |
|      7 |  98.1776 |
|      8 |  98.1343 |
+-----+-----+
4 rows in set (0.00 sec)
```

`hit_ratio` values approaching 100% indicate that only a very small number of reads are being made from disk rather than from the buffer, which means that Disk Data read performance is approaching an optimum level. If any of these values are less than 95%, this is a strong indicator that the setting for `DiskPageBufferMemory` [2083] needs to be increased in the `config.ini` file.



Note

A change in `DiskPageBufferMemory` [2083] requires a rolling restart of all of the cluster's data nodes before it takes effect.

17.5.10.9. The `ndbinfo logbuffers` Table

The `logbuffer` table provides information on MySQL Cluster log buffer usage.

The following table provides information about the columns in the `logbuffers` table. For each column, the table shows the name, data type, and a brief description.

Column Name	Type	Remarks
<code>node_id</code>	integer	The ID of this data node.
<code>log_type</code>	string	Type of log; one of: <code>REDO</code> or <code>DD-UNDO</code> .
<code>log_id</code>	integer	The log ID.
<code>log_part</code>	integer	The log part number.
<code>total</code>	integer	Total space available for this log.

Column Name	Type	Remarks
<code>used</code>	integer	Space used by this log.

17.5.10.10. The `ndbinfo logspaces` Table

This table provides information about MySQL Cluster log space usage.

The following table provides information about the columns in the `logspaces` table. For each column, the table shows the name, data type, and a brief description.

Column Name	Type	Remarks
<code>node_id</code>	integer	The ID of this data node.
<code>log_type</code>	string	Type of log; one of: <code>REDO</code> or <code>DD-UNDO</code> .
<code>log_id</code>	integer	The log ID.
<code>log_part</code>	integer	The log part number.
<code>total</code>	integer	Total space available for this log.
<code>used</code>	integer	Space used by this log.

17.5.10.11. The `ndbinfo membership` Table

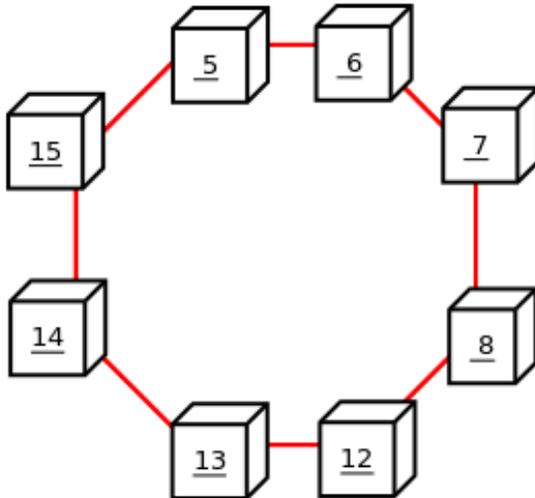
The `membership` table describes the view that each data node has of all the others in the cluster, including node group membership, president node, arbitrator, arbitrator successor, arbitrator connection states, and other information.

The following table provides information about the columns in the `membership` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	This node's node ID
<code>group_id</code>	integer	Node group to which this node belongs
<code>left_node</code>	integer	Node ID of the previous node
<code>right_node</code>	integer	Node ID of the next node
<code>president</code>	integer	President's node ID
<code>successor</code>	integer	Node ID of successor to president
<code>succession_order</code>	integer	Order in which this node succeeds to presidency
<code>Conf_HB_order</code>	integer	-
<code>arbitrator</code>	integer	Node ID of arbitrator
<code>arb_ticket</code>	string	Internal identifier used to track arbitration
<code>arb_state</code>	Enumeration (see text)	Arbitration state
<code>arb_connected</code>	Yes or No	Whether this node is connected to the arbitrator
<code>connected_rank1_arbs</code>	List of node IDs	Connected arbitrators of rank 1
<code>connected_rank2_arbs</code>	List of node IDs	Connected arbitrators of rank 1

The node ID and node group ID are the same as reported by `ndb_mgm -e "SHOW"`.

`left_node` and `right_node` are defined in terms of a model that connects all data nodes in a circle, in order of their node IDs, similar to the ordering of the numbers on a clock dial, as shown here:



In this example, we have 8 data nodes, numbered 5, 6, 7, 8, 12, 13, 14, and 15, ordered clockwise in a circle. We determine “left” and “right” from the interior of the circle. The node to the left of node 5 is node 15, and the node to the right of node 5 is node 6. You can see all these relationships by running the following query and observing the output:

```
mysql> SELECT node_id,left_node,right_node
-> FROM ndbinfo.membership;
```

node_id	left_node	right_node
5	15	6
6	5	7
7	6	8
8	7	12
12	8	13
13	12	14
14	13	15
15	14	5

8 rows in set (0.00 sec)

The designations “left” and “right” are used in the event log in the same way.

The `president` node is the node viewed by the current node as responsible for setting an arbitrator (see [MySQL Cluster Start Phases](#)). If the president fails or becomes disconnected, the current node expects the node whose ID is shown in the `successor` column to become the new president. The `succession_order` column shows the place in the succession queue that the current node views itself as having.

In a normal MySQL Cluster, all data nodes should see the same node as `president`, and the same node (other than the president) as its `successor`. In addition, the current president should see itself as `1` in the order of succession, the `successor` node should see itself as `2`, and so on.

All nodes should show the same `arb_ticket` values as well as the same `arb_state` values. Possible `arb_state` values are `ARBIT_NULL`, `ARBIT_INIT`, `ARBIT_FIND`, `ARBIT_PREP1`, `ARBIT_PREP2`, `ARBIT_START`, `ARBIT_RUN`, `ARBIT_CHOOSE`, `ARBIT_CRASH`, and `UNKNOWN`.

`arb_connected` shows whether this node is connected to the node shown as this node's `arbitorator`.

The `connected_rank1_arbs` and `connected_rank2_arbs` columns each display a list of 0 or more arbitrators having an `ArbitrationRank` [2037] equal to 1, or to 2, respectively.



Note

Both management nodes and API nodes are eligible to become arbitrators.

Like the `arbitrator_validity_detail` and `arbitrator_validity_summary` tables, this table was added in MySQL Cluster NDB 7.2.10.

17.5.10.12. The `ndbinfo memoryusage` Table

Querying this table provides information similar to that provided by the `ALL REPORT MemoryUsage` command in the `ndb_mgm` client, or logged by `ALL DUMP 1000`.

The following table provides information about the columns in the `memoryusage` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	The node ID of this data node.
<code>memory_type</code>	string	One of <code>DATA_MEMORY</code> or <code>INDEX_MEMORY</code> .
<code>used</code>	integer	Number of bytes currently used for data memory or index memory by this data node.
<code>used_pages</code>	integer	Number of pages currently used for data memory or index memory by this data node; see text.
<code>total</code>	integer	Total number of bytes of data memory or index memory available for this data node; see text.
<code>total_pages</code>	integer	Total number of memory pages available for data memory or index memory on this data node; see text.

The `total` column represents the total amount of memory in bytes available for the given resource (data memory or index memory) on a particular data node. This number should be approximately equal to the setting of the corresponding configuration parameter in the `config.ini` file.

Suppose that the cluster has 2 data nodes having node IDs 1 and 2, and the `config.ini` file contains the following:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
```

The following query shows approximately the same values:

```
mysql> SELECT node_id, memory_type, total
> FROM ndbinfo.memoryusage;
+-----+-----+-----+
| node_id | memory_type | total |
+-----+-----+-----+
| 1 | Data memory | 104857600 |
| 1 | Index memory | 105119744 |
| 2 | Data memory | 104857600 |
| 2 | Index memory | 105119744 |
+-----+-----+-----+
4 rows in set (0.30 sec)
```

In this case, the `total` column values for index memory are slightly higher than the value set of `IndexMemory` [2044] due to internal rounding.

For the `used_pages` and `total_pages` columns, resources are measured in pages, which are 32K in size for `DataMemory` [2043] and 8K for `IndexMemory` [2044].

17.5.10.13. The `ndbinfo nodes` Table

This table contains information on the status of data nodes. For each data node that is running in the cluster, a corresponding row in this table provides the node's node ID, status, and uptime. For nodes that are starting, it also shows the current start phase.

The following table provides information about the columns in the `nodes` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	The data node's unique node ID in the cluster.
<code>uptime</code>	integer	Time since the node was last started, in seconds.
<code>status</code>	string	Current status of the data node; see text for possible values.
<code>start_phase</code>	integer	If the data node is starting, the current start phase.
<code>config_generation</code>	integer	The version of the cluster configuration file in use on this data node.

The `uptime` column shows the time in seconds that this node has been running since it was last started or restarted. This is a `BIGINT` value. This figure includes the time actually needed to start the node; in other words, this counter starts running the moment that `ndbd` or `ndbmtid` is first invoked; thus, even for a node that has not yet finished starting, `uptime` may show a non-zero value.

The `status` column shows the node's current status. This is one of: `NOTHING`, `CMVMI`, `STARTING`, `STARTED`, `SINGLEUSER`, `STOPPING_1`, `STOPPING_2`, `STOPPING_3`, or `STOPPING_4`. When the status is `STARTING`, you can see the current start phase in the `start_phase` column (see later in this section). `SINGLEUSER` is displayed in the `status` column for all data nodes when the cluster is in single user mode (see Section 17.5.8, “MySQL Cluster Single User Mode”). Seeing one of the `STOPPING` states does not necessarily mean that the node is shutting down but can mean rather that it is entering a new state; for example, if you put the cluster in single user mode, you can sometimes see data nodes report their state briefly as `STOPPING_2` before the status changes to `SINGLEUSER`.

The `start_phase` column uses the same range of values as those used in the output of the `ndb_mgm` client `node_id STATUS` command (see Section 17.5.2, “Commands in the MySQL Cluster Management Client”). If the node is not currently starting, then this column shows `0`. For a listing of MySQL Cluster start phases with descriptions, see Section 17.5.1, “Summary of MySQL Cluster Start Phases”.

The `config_generation` column shows which version of the cluster configuration is in effect on each data node. This can be useful when performing a rolling restart of the cluster in order to make changes in configuration parameters. For example, from the output of the following `SELECT` statement, you can see that node 3 is not yet using the latest version of the cluster configuration (`6`) although nodes 1, 2, and 4 are doing so:

```
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
```

```

+-----+-----+-----+-----+-----+
| node_id | uptime | status | start_phase | config_generation |
+-----+-----+-----+-----+-----+
| 1 | 10462 | STARTED | 0 | 6 |
| 2 | 10460 | STARTED | 0 | 6 |
| 3 | 10457 | STARTED | 0 | 5 |
| 4 | 10455 | STARTED | 0 | 6 |
+-----+-----+-----+-----+-----+
2 rows in set (0.04 sec)

```

Therefore, for the case just shown, you should restart node 3 to complete the rolling restart of the cluster.

Nodes that are stopped are not accounted for in this table. Suppose that you have a MySQL Cluster with 4 data nodes (node IDs 1, 2, 3 and 4), and all nodes are running normally, then this table contains 4 rows, 1 for each data node:

```

mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+-----+-----+-----+-----+-----+
| node_id | uptime | status | start_phase | config_generation |
+-----+-----+-----+-----+-----+
| 1 | 11776 | STARTED | 0 | 6 |
| 2 | 11774 | STARTED | 0 | 6 |
| 3 | 11771 | STARTED | 0 | 6 |
| 4 | 11769 | STARTED | 0 | 6 |
+-----+-----+-----+-----+-----+
4 rows in set (0.04 sec)

```

If you shut down one of the nodes, only the nodes that are still running are represented in the output of this `SELECT` statement, as shown here:

```

ndb_mgm> 2 STOP
Node 2: Node shutdown initiated
Node 2: Node shutdown completed.
Node 2 has shutdown.

mysql> SELECT * FROM nodes;
+-----+-----+-----+-----+-----+
| node_id | uptime | status | start_phase | config_generation |
+-----+-----+-----+-----+-----+
| 1 | 11807 | STARTED | 0 | 6 |
| 3 | 11802 | STARTED | 0 | 6 |
| 4 | 11800 | STARTED | 0 | 6 |
+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)

```

17.5.10.14. The `ndbinfo resources` Table

This table provides information about data node resource availability and usage.

These resources are sometimes known as *super-pools*.

The following table provides information about the columns in the `resources` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	The unique node ID of this data node.
<code>resource_name</code>	string	Name of the resource; see text.
<code>reserved</code>	integer	The amount reserved for this resource.
<code>used</code>	integer	The amount actually used by this resource.

Column Name	Type	Remarks
<code>max</code>	integer	The maximum amount of this resource used, since the node was last started.

The `resource_name` can be one of `RESERVED`, `DISK_OPERATIONS`, `DISK_RECORDS`, `DATA_MEMORY`, `JOBBUFFER`, `FILE_BUFFERS`, or `TRANSPORTER_BUFFERS`.

17.5.10.15. The `ndbinfo server_operations` Table

The `server_operations` table contains entries for all ongoing NDB operations that the current SQL node (MySQL Server) is currently involved in. It effectively is a subset of the the `cluster_operations` table, in which operations for other SQL and API nodes are not shown.

The following table provides information about the columns in the `server_operations` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>mysql_connection_id</code>	integer	MySQL Server connection ID
<code>node_id</code>	integer	Node ID
<code>block_instance</code>	integer	Block instance
<code>transid</code>	integer	Transaction ID
<code>operation_type</code>	string	Operation type (see text for possible values)
<code>state</code>	string	Operation state (see text for possible values)
<code>tableid</code>	integer	Table ID
<code>fragmentid</code>	integer	Fragment ID
<code>client_node_id</code>	integer	Client node ID
<code>client_block_ref</code>	integer	Client block reference
<code>tc_node_id</code>	integer	Transaction coordinator node ID
<code>tc_block_no</code>	integer	Transaction coordinator block number
<code>tc_block_instance</code>	integer	Transaction coordinator block instance

The `mysql_connection_id` is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`. It is obtained from the `INFORMATION_SCHEMA` table `NDB_TRANSID_MYSQL_CONNECTION_MAP`.

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `operation_type` column can take any one of the values `READ`, `READ-SH`, `READ-EX`, `INSERT`, `UPDATE`, `DELETE`, `WRITE`, `UNLOCK`, `REFRESH`, `SCAN`, `SCAN-SH`, `SCAN-EX`, or `<unknown>`.

The `state` column can have any one of the values `ABORT_QUEUED`, `ABORT_STOPPED`, `COMMITTED`, `COMMIT_QUEUED`, `COMMIT_STOPPED`, `COPY_CLOSE_STOPPED`, `COPY_FIRST_STOPPED`, `COPY_STOPPED`, `COPY_TUPKEY`, `IDLE`, `LOG_ABORT_QUEUED`, `LOG_COMMIT_QUEUED`, `LOG_COMMIT_QUEUED_WAIT_SIGNAL`, `LOG_COMMIT_WRITTEN`, `LOG_COMMIT_WRITTEN_WAIT_SIGNAL`, `LOG_QUEUED`, `PREPARED`, `PREPARED_RECEIVED_COMMIT`, `SCAN_CHECK_STOPPED`, `SCAN_CLOSE_STOPPED`, `SCAN_FIRST_STOPPED`, `SCAN_RELEASE_STOPPED`, `SCAN_STATE_USED`, `SCAN_STOPPED`, `SCAN_TUPKEY`, `STOPPED`, `TC_NOT_CONNECTED`, `WAIT_ACC`, `WAIT_ACC_ABORT`, `WAIT_AI_AFTER_ABORT`, `WAIT_ATTR`, `WAIT_SCAN_AI`, `WAIT_TUP`,

`WAIT_TUPKEYINFO`, `WAIT_TUP_COMMIT`, or `WAIT_TUP_TO_ABORT`. (If the MySQL Server is running with `ndbinfo_show_hidden` [2159] enabled, you can view this list of states by selecting from the `ndb $dblqh_tcconnect_state` table, which is normally hidden.)

You can obtain the name of an NDB table from its table ID by checking the output of `ndb_show_tables`.

The `fragid` is the same as the partition number seen in the output of `ndb_desc --extra-partition-info` [2207] (short form `-p`).

In `client_node_id` and `client_block_ref`, `client` refers to a MySQL Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

This table was added in MySQL Cluster NDB 7.2.2.

17.5.10.16. The `ndbinfo server_transactions` Table

The `server_transactions` table is subset of the `cluster_transactions` table, but includes only those transactions in which the current SQL node (MySQL Server) is a participant, while including the relevant connection IDs.

The following table provides information about the columns in the `server_transactions` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>mysql_connection_id</code>	integer	MySQL Server connection ID
<code>node_id</code>	integer	Transaction coordinator node ID
<code>block_instance</code>	integer	Transaction coordinator block instance
<code>transid</code>	integer	Transaction ID
<code>state</code>	string	Operation state (see text for possible values)
<code>count_operations</code>	integer	Number of stateful operations in the transaction
<code>outstanding_operations</code>	integer	Operations still being executed by local data management layer (LQH blocks)
<code>inactive_seconds</code>	integer	Time spent waiting for API
<code>client_node_id</code>	integer	Client node ID
<code>client_block_ref</code>	integer	Client block reference

The `mysql_connection_id` is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`. It is obtained from the `INFORMATION_SCHEMA` table `NDB_TRANSID_MYSQL_CONNECTION_MAP`.

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `state` column can have any one of the values `CS_ABORTING`, `CS_COMMITTING`, `CS_COMMIT_SENT`, `CS_COMPLETE_SENT`, `CS_COMPLETING`, `CS_CONNECTED`, `CS_DISCONNECTED`, `CS_FAIL_ABORTED`, `CS_FAIL_ABORTING`, `CS_FAIL_COMMITTED`, `CS_FAIL_COMMITTING`, `CS_FAIL_COMPLETED`, `CS_FAIL_PREPARED`, `CS_PREPARE_TO_COMMIT`, `CS_RECEIVING`, `CS_REC_COMMITTING`, `CS_RESTART`, `CS_SEND_FIRE_TRIG_REQ`, `CS_STARTED`, `CS_START_COMMITTING`, `CS_START_SCAN`, `CS_WAIT_ABORT_CONF`, `CS_WAIT_COMMIT_CONF`, `CS_WAIT_COMPLETE_CONF`, `CS_WAIT_FIRE_TRIG_REQ`. (If the MySQL Server is running with `ndbinfo_show_hidden` [2159]

enabled, you can view this list of states by selecting from the `ndb$dbtcc_apiconnect_state` table, which is normally hidden.)

In `client_node_id` and `client_block_ref`, `client` refers to a MySQL Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

This table was added in MySQL Cluster NDB 7.2.2.

17.5.10.17. The `ndbinfo threadblocks` Table

The `threadblocks` table associates data nodes, threads, and instances of NDB kernel blocks.

The following table provides information about the columns in the `threadblocks` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	Node ID
<code>thr_no</code>	integer	Thread ID
<code>block_name</code>	string	Block name
<code>block_instance</code>	integer	Block instance number

The `block_name` is one of the values found in the `block_name` column when selecting from the `ndbinfo.blocks` table. Although the list of possible values is static for a given MySQL Cluster release, the list may vary between releases.

This table was added in MySQL Cluster NDB 7.2.2.

17.5.10.18. The `ndbinfo threadstat` Table

The `threadstat` table provides a rough snapshot of statistics for threads running in the NDB kernel.

The following table provides information about the columns in the `threadstat` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	Node ID
<code>thr_no</code>	integer	Thread ID
<code>thr_nm</code>	string	Thread name
<code>c_loop</code>	string	Number of loops in main loop
<code>c_exec</code>	string	Number of signals executed
<code>c_wait</code>	string	Number of times waiting for additional input
<code>c_l_sent_prioa</code>	integer	Number of priority A signals sent to own node
<code>c_l_sent_priob</code>	integer	Number of priority B signals sent to own node
<code>c_r_sent_prioa</code>	integer	Number of priority A signals sent to remote node
<code>c_r_sent_priob</code>	integer	Number of priority B signals sent to remote node
<code>os_tid</code>	integer	OS thread ID
<code>os_now</code>	integer	OS time (ms)

Column Name	Type	Remarks
<code>os_ru_utime</code>	integer	OS user CPU time (μ s)
<code>os_ru_stime</code>	integer	OS system CPU time (μ s)
<code>os_ru_minflt</code>	integer	OS page reclaims (soft page faults)
<code>os_ru_majflt</code>	integer	OS page faults (hard page faults)
<code>os_ru_nvcsw</code>	integer	OS voluntary context switches
<code>os_ru_nivcsw</code>	integer	OS involuntary context switches

`os_time` uses the system `gettimeofday()` call.

The values of the `os_ru_utime`, `os_ru_stime`, `os_ru_minflt`, `os_ru_majflt`, `os_ru_nvcsw`, and `os_ru_nivcsw` columns are obtained using the system `getrusage()` call, or the equivalent.

Since this table contains counts taken at a given point in time, for best results it is necessary to query this table periodically and store the results in an intermediate table or tables. The MySQL Server's Event Scheduler can be employed to automate such monitoring. For more information, see [Section 19.4, "Using the Event Scheduler"](#).

This table was added in MySQL Cluster NDB 7.2.2.

17.5.10.19. The `ndbinfo transporters` Table

This table contains information about NDB transporters.

The following table provides information about the columns in the `transporters` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Column Name	Type	Remarks
<code>node_id</code>	integer	This data node's unique node ID in the cluster.
<code>remote_node_id</code>	integer	The remote data node's node ID.
<code>status</code>	string	Status of the connection.
<code>remote_address</code>	string	Name or IP address of the remote host
<code>bytes_sent</code>	integer	Number of bytes sent using this connection
<code>bytes_received</code>		Number of bytes received using this connection
<code>connect_count</code>		Number of times connection established on this transporter
<code>overloaded</code>		1 if this transporter is currently overloaded, otherwise 0
<code>overload_count</code>		Number of times this transporter has entered overload state since connecting
<code>slowdown</code>		1 if this transporter is in scan slowdown state, otherwise 0
<code>slowdown_count</code>		Number of times this transporter has entered scan slowdown state since connecting

For each running data node in the cluster, the `transporters` table displays a row showing the status of each of that node's connections with all nodes in the cluster, *including itself*. This information is shown

in the table's `status` column, which can have any one of the following values: `CONNECTING`, `CONNECTED`, `DISCONNECTING`, or `DISCONNECTED`.

Connections to API and management nodes which are configured but not currently connected to the cluster are shown with status `DISCONNECTED`. Rows where the `node_id` is that of a data nodes which is not currently connected are not shown in this table. (This is similar omission of disconnected nodes in the `ndbinfo.nodes` table.

The `remote_address`, `bytes_sent`, and `bytes_received` columns were added in MySQL Cluster NDB 7.2.9. The `remote_address` is the the host name or address for the node whose ID is shown in the `remote_node_id` column. The `bytes_sent` from this node and `bytes_received` by this node are the numbers, respectively, of bytes sent and received by the node using this connection since it was established; for nodes whose status is `CONNECTING` or `DISCONNECTED`, these columns always display 0.

The `connect_count`, `overloaded`, `overload_count`, `slowdown`, and `slowdown_count` columns were added in MySQL Cluster NDB 7.2.10. These counters are reset on connection, and retain their values after the remote node disconnects. Also beginning with MySQL Cluster NDB 7.2.10, the `bytes_send` and `bytes_received` counters are reset on connection as well, and so retain their values following disconnection. (Previously, the values in these columns were reset on disconnection.) (Bug #15935206)

Assume you have a 5-node cluster consisting of 2 data nodes, 2 SQL nodes, and 1 management node, as shown in the output of the `SHOW` command in the `ndb_mgm` client:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)]      2 node(s)
id=1   @10.100.10.1  (5.5.29-ndb-7.2.10, Nodegroup: 0, Master)
id=2   @10.100.10.2  (5.5.29-ndb-7.2.10, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10  @10.100.10.10 (5.5.29-ndb-7.2.10)

[mysqld(API)]   2 node(s)
id=20  @10.100.10.20 (5.5.29-ndb-7.2.10)
id=21  @10.100.10.21 (5.5.29-ndb-7.2.10)
```

There are 10 rows in the `transporters` table—5 for the first data node, and 5 for the second—assuming that all data nodes are running, as shown here:

```
mysql> SELECT node_id, remote_node_id, status
-> FROM ndbinfo.transporters;
+-----+-----+-----+
| node_id | remote_node_id | status |
+-----+-----+-----+
| 1 | 1 | CONNECTED |
| 1 | 2 | CONNECTED |
| 1 | 10 | CONNECTED |
| 1 | 20 | CONNECTED |
| 1 | 21 | CONNECTED |
| 2 | 1 | CONNECTED |
| 2 | 2 | CONNECTED |
| 2 | 10 | CONNECTED |
| 2 | 20 | CONNECTED |
| 2 | 21 | CONNECTED |
+-----+-----+-----+
10 rows in set (0.04 sec)
```

If you shut down one of the data nodes in this cluster using the command `2 STOP` in the `ndb_mgm` client, then repeat the previous query (again using the `mysql` client), this table now shows only 5 rows—1 row for

each connection from the remaining management node to another node, including both itself and the data node that is currently offline—and displays `CONNECTING` for the status of each remaining connection to the data node that is currently offline, as shown here:

```
mysql> SELECT node_id, remote_node_id, status
-> FROM ndbinfo.transporters;
+-----+-----+-----+
| node_id | remote_node_id | status      |
+-----+-----+-----+
| 1       | 1               | CONNECTED  |
| 1       | 2               | CONNECTING |
| 1       | 10              | CONNECTED  |
| 1       | 20              | CONNECTED  |
| 1       | 21              | CONNECTED  |
+-----+-----+-----+
5 rows in set (0.02 sec)
```

17.5.11. MySQL Cluster Security Issues

This section discusses security considerations to take into account when setting up and running MySQL Cluster.

Topics covered in this section include the following:

- MySQL Cluster and network security issues
- Configuration issues relating to running MySQL Cluster securely
- MySQL Cluster and the MySQL privilege system
- MySQL standard security procedures as applicable to MySQL Cluster

17.5.11.1. MySQL Cluster Security and Networking Issues

In this section, we discuss basic network security issues as they relate to MySQL Cluster. It is extremely important to remember that MySQL Cluster “out of the box” is not secure; you or your network administrator must take the proper steps to ensure that your cluster cannot be compromised over the network.

Cluster communication protocols are inherently insecure, and no encryption or similar security measures are used in communications between nodes in the cluster. Because network speed and latency have a direct impact on the cluster's efficiency, it is also not advisable to employ SSL or other encryption to network connections between nodes, as such schemes will effectively slow communications.

It is also true that no authentication is used for controlling API node access to a MySQL Cluster. As with encryption, the overhead of imposing authentication requirements would have an adverse impact on Cluster performance.

In addition, there is no checking of the source IP address for either of the following when accessing the cluster:

- SQL or API nodes using “free slots” created by empty `[mysqld]` or `[api]` sections in the `config.ini` file

This means that, if there are any empty `[mysqld]` or `[api]` sections in the `config.ini` file, then any API nodes (including SQL nodes) that know the management server's host name (or IP address) and port can connect to the cluster and access its data without restriction. (See [Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”](#), for more information about this and related issues.)

**Note**

You can exercise some control over SQL and API node access to the cluster by specifying a `HostName` parameter for all `[mysqld]` and `[api]` sections in the `config.ini` file. However, this also means that, should you wish to connect an API node to the cluster from a previously unused host, you need to add an `[api]` section containing its host name to the `config.ini` file.

More information is available elsewhere in this chapter [2091] about the `HostName` parameter. Also see Section 17.3.1, “Quick Test Setup of MySQL Cluster”, for configuration examples using `HostName` with API nodes.

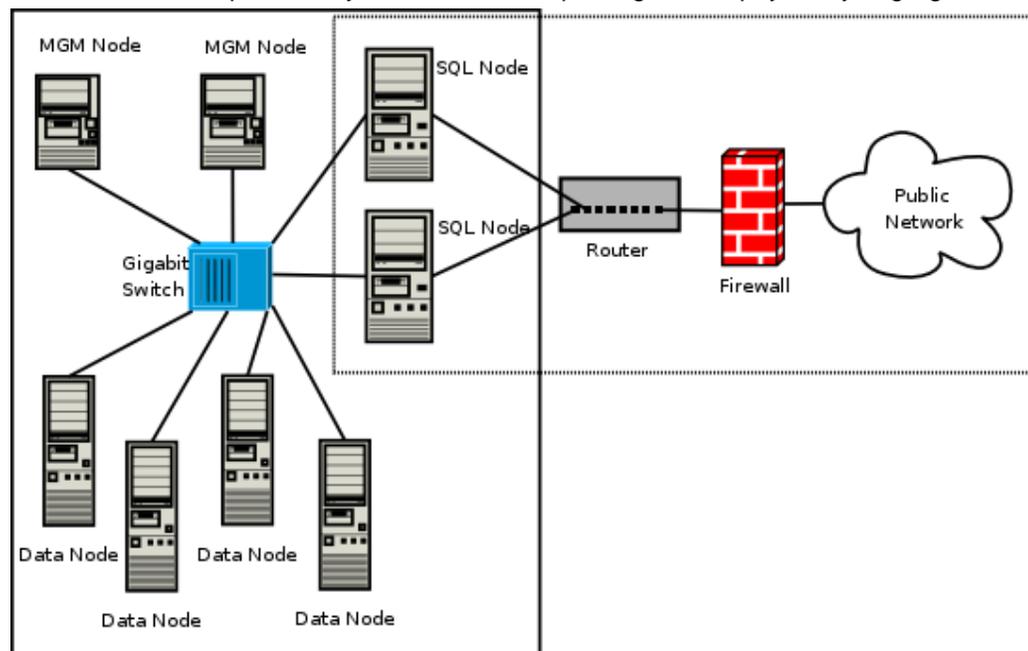
- Any `ndb_mgm` client

This means that any cluster management client that is given the management server's host name (or IP address) and port (if not the standard port) can connect to the cluster and execute any management client command. This includes commands such as `ALL STOP` and `SHUTDOWN`.

For these reasons, it is necessary to protect the cluster on the network level. The safest network configuration for Cluster is one which isolates connections between Cluster nodes from any other network communications. This can be accomplished by any of the following methods:

1. Keeping Cluster nodes on a network that is physically separate from any public networks. This option is the most dependable; however, it is the most expensive to implement.

We show an example of a MySQL Cluster setup using such a physically segregated network here:



This setup has two networks, one private (solid box) for the Cluster management servers and data nodes, and one public (dotted box) where the SQL nodes reside. (We show the management and data nodes connected using a gigabit switch since this provides the best performance.) Both networks are protected from the outside by a hardware firewall, sometimes also known as a *network-based firewall*.

This network setup is safest because no packets can reach the cluster's management or data nodes from outside the network—and none of the cluster's internal communications can reach the outside

—without going through the SQL nodes, as long as the SQL nodes do not permit any packets to be forwarded. This means, of course, that all SQL nodes must be secured against hacking attempts.



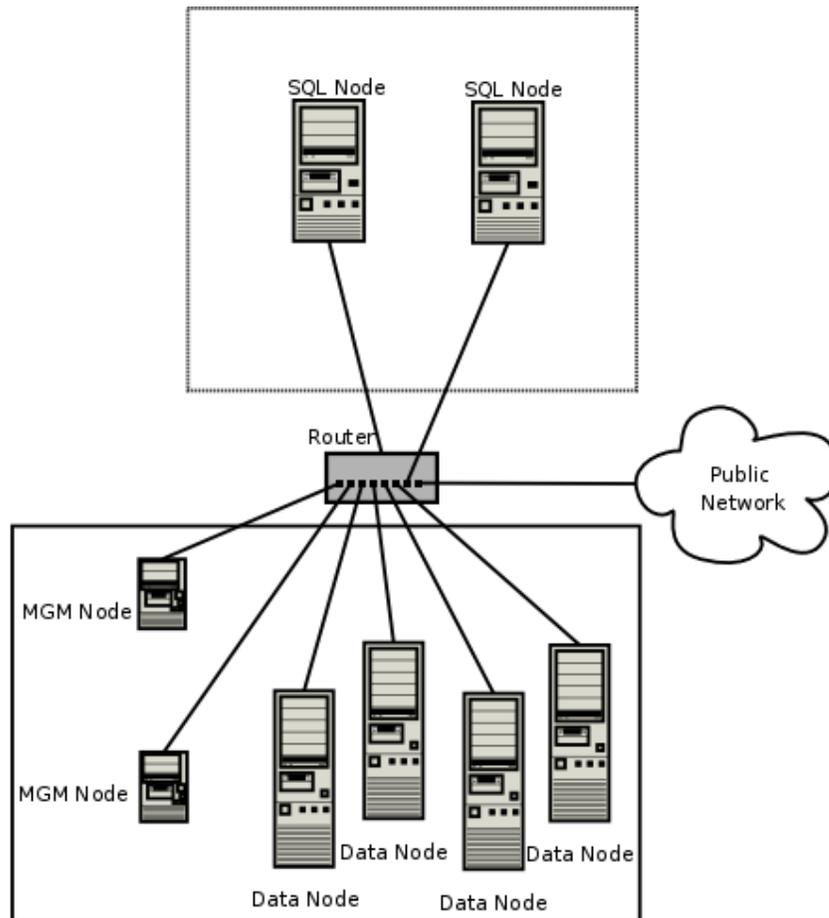
Important

With regard to potential security vulnerabilities, an SQL node is no different from any other MySQL server. See [Section 6.1.3, “Making MySQL Secure Against Attackers”](#), for a description of techniques you can use to secure MySQL servers.

- Using one or more software firewalls (also known as *host-based firewalls*) to control which packets pass through to the cluster from portions of the network that do not require access to it. In this type of setup, a software firewall must be installed on every host in the cluster which might otherwise be accessible from outside the local network.

The host-based option is the least expensive to implement, but relies purely on software to provide protection and so is the most difficult to keep secure.

This type of network setup for MySQL Cluster is illustrated here:



Using this type of network setup means that there are two zones of MySQL Cluster hosts. Each cluster host must be able to communicate with all of the other machines in the cluster, but only those hosting SQL nodes (dotted box) can be permitted to have any contact with the outside, while those in the zone containing the data nodes and management nodes (solid box) must be isolated from any machines that

are not part of the cluster. Applications using the cluster and user of those applications must *not* be permitted to have direct access to the management and data node hosts.

To accomplish this, you must set up software firewalls that limit the traffic to the type or types shown in the following table, according to the type of node that is running on each cluster host computer:

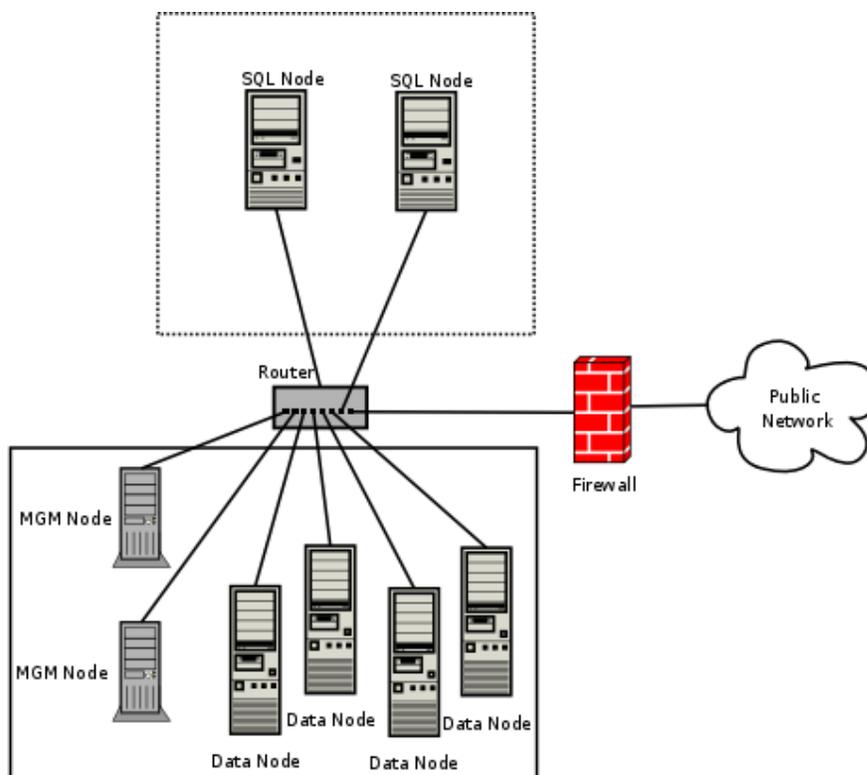
Type of Node to be Accessed	Traffic to Permit
SQL or API node	<ul style="list-style-type: none"> • It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from within the network in which the cluster resides and is on the port that your application is using.
Data node or Management node	<ul style="list-style-type: none"> • It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from the IP address of an SQL or API node.

Any traffic other than that shown in the table for a given node type should be denied.

The specifics of configuring a firewall vary from firewall application to firewall application, and are beyond the scope of this Manual. [iptables](#) is a very common and reliable firewall application, which is often used with [APF](#) as a front end to make configuration easier. You can (and should) consult the documentation for the software firewall that you employ, should you choose to implement a MySQL Cluster network setup of this type, or of a “mixed” type as discussed under the next item.

3. It is also possible to employ a combination of the first two methods, using both hardware and software to secure the cluster—that is, using both network-based and host-based firewalls. This is between the first two schemes in terms of both security level and cost. This type of network setup keeps the cluster behind the hardware firewall, but permits incoming packets to travel beyond the router connecting all cluster hosts to reach the SQL nodes.

One possible network deployment of a MySQL Cluster using hardware and software firewalls in combination is shown here:



In this case, you can set the rules in the hardware firewall to deny any external traffic except to SQL nodes and API nodes, and then permit traffic to them only on the ports required by your application.

Whatever network configuration you use, remember that your objective from the viewpoint of keeping the cluster secure remains the same—to prevent any unessential traffic from reaching the cluster while ensuring the most efficient communication between the nodes in the cluster.

Because MySQL Cluster requires large numbers of ports to be open for communications between nodes, the recommended option is to use a segregated network. This represents the simplest way to prevent unwanted traffic from reaching the cluster.



Note

If you wish to administer a MySQL Cluster remotely (that is, from outside the local network), the recommended way to do this is to use `ssh` or another secure login shell to access an SQL node host. From this host, you can then run the management client to access the management server safely, from within the Cluster's own local network.

Even though it is possible to do so in theory, it is *not* recommended to use `ndb_mgm` to manage a Cluster directly from outside the local network on which the Cluster is running. Since neither authentication nor encryption takes place between the management client and the management server, this represents an extremely insecure means of managing the cluster, and is almost certain to be compromised sooner or later.

17.5.11.2. MySQL Cluster and MySQL Privileges

In this section, we discuss how the MySQL privilege system works in relation to MySQL Cluster and the implications of this for keeping a MySQL Cluster secure.

Standard MySQL privileges apply to MySQL Cluster tables. This includes all MySQL privilege types (`SELECT` [697] privilege, `UPDATE` [697] privilege, `DELETE` [696] privilege, and so on) granted on the database, table, and column level. As with any other MySQL Server, user and privilege information is stored in the `mysql` system database. The SQL statements used to grant and revoke privileges on `NDB` tables, databases containing such tables, and columns within such tables are identical in all respects with the `GRANT` and `REVOKE` statements used in connection with database objects involving any (other) MySQL storage engine. The same thing is true with respect to the `CREATE USER` and `DROP USER` statements.

It is important to keep in mind that, by default, the MySQL grant tables use the `MyISAM` storage engine. Because of this, those tables are not normally duplicated or shared among MySQL servers acting as SQL nodes in a MySQL Cluster. In other words, changes in users and their privileges do not automatically propagate between SQL nodes by default. In MySQL Cluster `NDB` 7.2 (and later), you can enable automatic distribution of MySQL users and privileges across MySQL Cluster SQL nodes; see [Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”](#), for details.

Conversely, because there is no way in MySQL to deny privileges (privileges can either be revoked or not granted in the first place, but not denied as such), there is no special protection for `NDB` tables on one SQL node from users that have privileges on another SQL node; (This is true even if you are not using automatic distribution of user privileges. The definitive example of this is the MySQL `root` account, which can perform any action on any database object. In combination with empty `[mysqld]` or `[api]` sections of the `config.ini` file, this account can be especially dangerous. To understand why, consider the following scenario:

- The `config.ini` file contains at least one empty `[mysqld]` or `[api]` section. This means that the MySQL Cluster management server performs no checking of the host from which a MySQL Server (or other API node) accesses the MySQL Cluster.
- There is no firewall, or the firewall fails to protect against access to the MySQL Cluster from hosts external to the network.
- The host name or IP address of the MySQL Cluster's management server is known or can be determined from outside the network.

If these conditions are true, then anyone, anywhere can start a MySQL Server with `--ndbcluster` [2142] `--ndb-connectstring=management_host` [2141] and access this MySQL Cluster. Using the MySQL `root` account, this person can then perform the following actions:

- Execute metadata statements such as `SHOW DATABASES` statement (to obtain a list of all `NDB` databases on the server) or `SHOW TABLES FROM some_ndb_database` statement to obtain a list of all `NDB` tables in a given database
- Run any legal MySQL statements on any of the discovered tables, such as:
 - `SELECT * FROM some_table` to read all the data from any table
 - `DELETE FROM some_table` to delete all the data from a table
 - `DESCRIBE some_table` or `SHOW CREATE TABLE some_table` to determine the table schema
 - `UPDATE some_table SET column1 = some_value` to fill a table column with “garbage” data; this could actually cause much greater damage than simply deleting all the data

More insidious variations might include statements like these:

```
UPDATE some_table SET an_int_column = an_int_column + 1
```

or

```
UPDATE some_table SET a_varchar_column = REVERSE(a_varchar_column)
```

Such malicious statements are limited only by the imagination of the attacker.

The only tables that would be safe from this sort of mayhem would be those tables that were created using storage engines other than `NDB`, and so not visible to a “rogue” SQL node.

A user who can log in as `root` can also access the `INFORMATION_SCHEMA` database and its tables, and so obtain information about databases, tables, stored routines, scheduled events, and any other database objects for which metadata is stored in `INFORMATION_SCHEMA`.

It is also a very good idea to use different passwords for the `root` accounts on different MySQL Cluster SQL nodes unless you are using distributed privileges.

In sum, you cannot have a safe MySQL Cluster if it is directly accessible from outside your local network.



Important

Never leave the MySQL root account password empty. This is just as true when running MySQL as a MySQL Cluster SQL node as it is when running it as a standalone (non-Cluster) MySQL Server, and should be done as part of the MySQL installation process before configuring the MySQL Server as an SQL node in a MySQL Cluster.

Prior to MySQL Cluster `NDB` 7.2, you should never convert the system tables in the `mysql` database to use the `NDB` storage engine. There are a number of reasons why you should not do this, but the most important reason is this: *Many of the SQL statements that affect `mysql` tables storing information about user privileges, stored routines, scheduled events, and other database objects cease to function if these tables are changed to use any storage engine other than `MyISAM`.* This is a consequence of various MySQL Server internals. Beginning with MySQL Cluster `NDB` 7.2, you can use a stored procedure provided for this purpose (see [Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”](#)), but you are strongly advised not to attempt convert the system tables manually.

Otherwise, if you need to synchronize `mysql` system tables between SQL nodes, you can use standard MySQL replication to do so, or employ a script to copy table entries between the MySQL servers.

Summary. The most important points to remember regarding the MySQL privilege system with regard to MySQL Cluster are listed here:

1. Users and privileges established on one SQL node do not automatically exist or take effect on other SQL nodes in the cluster. Conversely, removing a user or privilege on one SQL node in the cluster does not remove the user or privilege from any other SQL nodes.
2. You can distribute MySQL users and privileges among SQL nodes using the SQL script, and the stored procedures it contains, that are supplied for this purpose in the MySQL Cluster distribution.
3. Once a MySQL user is granted privileges on an `NDB` table from one SQL node in a MySQL Cluster, that user can “see” any data in that table regardless of the SQL node from which the data originated, even if you are not using privilege distribution.

17.5.11.3. MySQL Cluster and MySQL Security Procedures

In this section, we discuss MySQL standard security procedures as they apply to running MySQL Cluster.

In general, any standard procedure for running MySQL securely also applies to running a MySQL Server as part of a MySQL Cluster. First and foremost, you should always run a MySQL Server as the `mysql`

system user; this is no different from running MySQL in a standard (non-Cluster) environment. The `mysql` system account should be uniquely and clearly defined. Fortunately, this is the default behavior for a new MySQL installation. You can verify that the `mysqld` process is running as the system user `mysql` by using the system command such as the one shown here:

```
shell> ps aux | grep mysql
root      10467  0.0  0.1  3616  1380 pts/3    S      11:53   0:00 \
  /bin/sh ./mysqld_safe --ndbcluster --ndb-connectstring=localhost:1186
mysql     10512  0.2  2.5  58528 26636 pts/3    Sl     11:53   0:00 \
  /usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql \
  --datadir=/usr/local/mysql/var --user=mysql --ndbcluster \
  --ndb-connectstring=localhost:1186 --pid-file=/usr/local/mysql/var/mothra.pid \
  --log-error=/usr/local/mysql/var/mothra.err
jon       10579  0.0  0.0   2736   688 pts/0    S+    11:54   0:00 grep mysql
```

If the `mysqld` process is running as any other user than `mysql`, you should immediately shut it down and restart it as the `mysql` user. If this user does not exist on the system, the `mysql` user account should be created, and this user should be part of the `mysql` user group; in this case, you should also make sure that the MySQL data directory on this system (as set using the `--datadir` [439] option for `mysqld`) is owned by the `mysql` user, and that the SQL node's `my.cnf` file includes `user=mysql` in the `[mysqld]` section. Alternatively, you can start the MySQL server process with `--user=mysql` [466] on the command line, but it is preferable to use the `my.cnf` option, since you might forget to use the command-line option and so have `mysqld` running as another user unintentionally. The `mysqld_safe` startup script forces MySQL to run as the `mysql` user.



Important

Never run `mysqld` as the system root user. Doing so means that potentially any file on the system can be read by MySQL, and thus—should MySQL be compromised—by an attacker.

As mentioned in the previous section (see [Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”](#)), you should always set a root password for the MySQL Server as soon as you have it running. You should also delete the anonymous user account that is installed by default. You can accomplish these tasks using the following statements:

```
shell> mysql -u root

mysql> UPDATE mysql.user
  ->   SET Password=PASSWORD('secure_password')
  ->   WHERE User='root';

mysql> DELETE FROM mysql.user
  ->   WHERE User='';

mysql> FLUSH PRIVILEGES;
```

Be very careful when executing the `DELETE` statement not to omit the `WHERE` clause, or you risk deleting *all* MySQL users. Be sure to run the `FLUSH PRIVILEGES` statement as soon as you have modified the `mysql.user` table, so that the changes take immediate effect. Without `FLUSH PRIVILEGES`, the changes do not take effect until the next time that the server is restarted.



Note

Many of the MySQL Cluster utilities such as `ndb_show_tables`, `ndb_desc`, and `ndb_select_all` also work without authentication and can reveal table names, schemas, and data. By default these are installed on Unix-style systems with the permissions `wxr-xr-x` (755), which means they can be executed by any user that can access the `mysql/bin` directory.

See [Section 17.4, “MySQL Cluster Programs”](#), for more information about these utilities.

17.5.12. MySQL Cluster Disk Data Tables

It is possible to store the nonindexed columns of `NDB` tables on disk, rather than in RAM.

As part of implementing MySQL Cluster Disk Data work, a number of improvements were made in MySQL Cluster for the efficient handling of very large amounts (terabytes) of data during node recovery and restart. These include a “no-steal” algorithm for synchronizing a starting node with very large data sets. For more information, see the paper [Recovery Principles of MySQL Cluster 5.1](#), by MySQL Cluster developers Mikael Ronström and Jonas Orelund.

MySQL Cluster Disk Data performance can be influenced by a number of configuration parameters. For information about these parameters and their effects, see [MySQL Cluster Disk Data configuration parameters](#) and [MySQL Cluster Disk Data storage and GCP Stop errors](#)

The performance of a MySQL Cluster that uses Disk Data storage can also be greatly improved by separating data node file systems from undo log files and tablespace data files, which can be done using symbolic links. For more information, see [Section 17.5.12.2, “Using Symbolic Links with Disk Data Objects”](#).

17.5.12.1. MySQL Cluster Disk Data Objects

MySQL Cluster Disk Data storage is implemented using a number of *Disk Data objects*. These include the following:

- *Tablespaces* act as containers for other Disk Data objects.
- *Undo log files* undo information required for rolling back transactions.
- One or more undo log files are assigned to a *log file group*, which is then assigned to a tablespace.
- *Data files* store Disk Data table data. A data file is assigned directly to a tablespace.

Undo log files and data files are actual files in the file system of each data node; by default they are placed in `ndb_node_id_fs` in the `DataDir` specified in the MySQL Cluster `config.ini` file, and where `node_id` is the data node's node ID. It is possible to place these elsewhere by specifying either an absolute or relative path as part of the filename when creating the undo log or data file. Statements that create these files are shown later in this section.

MySQL Cluster tablespaces and log file groups are not implemented as files.



Important

Although not all Disk Data objects are implemented as files, they all share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group both named `ddl`.

Assuming that you have already set up a MySQL Cluster with all nodes (including management and SQL nodes), the basic steps for creating a MySQL Cluster table on disk are as follows:

1. Create a log file group, and assign one or more undo log files to it (an undo log file is also sometimes referred to as an *undofile*).

**Note**

Undo log files are necessary only for Disk Data tables; they are not used for [NDBCLUSTER](#) tables that are stored only in memory.

2. Create a tablespace; assign the log file group, as well as one or more data files, to the tablespace.
3. Create a Disk Data table that uses this tablespace for data storage.

Each of these tasks can be accomplished using SQL statements in the `mysql` client or other MySQL client application, as shown in the example that follows.

1. We create a log file group named `lg_1` using `CREATE LOGFILE GROUP`. This log file group is to be made up of two undo log files, which we name `undo_1.log` and `undo_2.log`, whose initial sizes are 16 MB and 12 MB, respectively. (The default initial size for an undo log file is 128 MB.) Optionally, you can also specify a size for the log file group's undo buffer, or permit it to assume the default value of 8 MB. In this example, we set the UNDO buffer's size at 2 MB. A log file group must be created with an undo log file; so we add `undo_1.log` to `lg_1` in this `CREATE LOGFILE GROUP` statement:

```
CREATE LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_1.log'
  INITIAL_SIZE 16M
  UNDO_BUFFER_SIZE 2M
  ENGINE NDBCLUSTER;
```

To add `undo_2.log` to the log file group, use the following `ALTER LOGFILE GROUP` statement:

```
ALTER LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_2.log'
  INITIAL_SIZE 12M
  ENGINE NDBCLUSTER;
```

Some items of note:

- The `.log` file extension used here is not required. We use it merely to make the log files easily recognisable.
- Every `CREATE LOGFILE GROUP` and `ALTER LOGFILE GROUP` statement must include an `ENGINE` clause. In MySQL Cluster NDB 7.2, the only permitted values for this clause are `NDBCLUSTER` and `NDB`.

**Important**

There can exist at most one log file group in the same MySQL Cluster at any given time.

- When you add an undo log file to a log file group using `ADD UNDOFILE 'filename'`, a file with the name `filename` is created in the `ndb_node_id_fs` directory within the `DataDir [2042]` of each data node in the cluster, where `node_id` is the node ID of the data node. Each undo log file is of the size specified in the SQL statement. For example, if a MySQL Cluster has 4 data nodes, then the `ALTER LOGFILE GROUP` statement just shown creates 4 undo log files, 1 each on in the data directory of each of the 4 data nodes; each of these files is named `undo_2.log` and each file is 12 MB in size.
- `UNDO_BUFFER_SIZE` is limited by the amount of system memory available.

- For more information about the `CREATE LOGFILE GROUP` statement, see [Section 13.1.14](#), “`CREATE LOGFILE GROUP Syntax`”. For more information about `ALTER LOGFILE GROUP`, see [Section 13.1.3](#), “`ALTER LOGFILE GROUP Syntax`”.
2. Now we can create a tablespace, which contains files to be used by MySQL Cluster Disk Data tables for storing their data. A tablespace is also associated with a particular log file group. When creating a new tablespace, you must specify the log file group which it is to use for undo logging; you must also specify a data file. You can add more data files to the tablespace after the tablespace is created; it is also possible to drop data files from a tablespace (an example of dropping data files is provided later in this section).

Assume that we wish to create a tablespace named `ts_1` which uses `lg_1` as its log file group. This tablespace is to contain two data files named `data_1.dat` and `data_2.dat`, whose initial sizes are 32 MB and 48 MB, respectively. (The default value for `INITIAL_SIZE` is 128 MB.) We can do this using two SQL statements, as shown here:

```
CREATE TABLESPACE ts_1
  ADD DATAFILE 'data_1.dat'
  USE LOGFILE GROUP lg_1
  INITIAL_SIZE 32M
  ENGINE NDBCLUSTER;

ALTER TABLESPACE ts_1
  ADD DATAFILE 'data_2.dat'
  INITIAL_SIZE 48M
  ENGINE NDBCLUSTER;
```

The `CREATE TABLESPACE` statement creates a tablespace `ts_1` with the data file `data_1.dat`, and associates `ts_1` with log file group `lg_1`. The `ALTER TABLESPACE` adds the second data file (`data_2.dat`).

Some items of note:

- As is the case with the `.log` file extension used in this example for undo log files, there is no special significance for the `.dat` file extension; it is used merely for easy recognition of data files.
 - When you add a data file to a tablespace using `ADD DATAFILE 'filename'`, a file with the name `filename` is created in the `ndb_node_id_fs` directory within the `DataDir` [2042] of each data node in the cluster, where `node_id` is the node ID of the data node. Each undo log file is of the size specified in the SQL statement. For example, if a MySQL Cluster has 4 data nodes, then the `ALTER TABLESPACE` statement just shown creates 4 undo log files, 1 each on in the data directory of each of the 4 data nodes; each of these files is named `data_2.dat` and each file is 48 MB in size.
 - All `CREATE TABLESPACE` and `ALTER TABLESPACE` statements must contain an `ENGINE` clause; only tables using the same storage engine as the tablespace can be created in the tablespace. In MySQL MySQL Cluster NDB 7.2, the only permitted values for this clause are `NDBCLUSTER` and `NDB`.
 - For more information about the `CREATE TABLESPACE` and `ALTER TABLESPACE` statements, see [Section 13.1.18](#), “`CREATE TABLESPACE Syntax`”, and [Section 13.1.8](#), “`ALTER TABLESPACE Syntax`”.
3. Now it is possible to create a table whose nonindexed columns are stored on disk in the tablespace `ts_1`:

```
CREATE TABLE dt_1 (
  member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  last_name VARCHAR(50) NOT NULL,
```

```

first_name VARCHAR(50) NOT NULL,
dob DATE NOT NULL,
joined DATE NOT NULL,
INDEX(last_name, first_name)
)
TABLESPACE ts_1 STORAGE DISK
ENGINE NDBCLUSTER;

```

The `TABLESPACE ... STORAGE DISK` option tells the `NDBCLUSTER` storage engine to use tablespace `ts_1` for disk data storage.



Note

It is also possible to specify whether an individual column is stored on disk or in memory by using a `STORAGE` clause as part of the column's definition in a `CREATE TABLE` or `ALTER TABLE` statement. `STORAGE DISK` causes the column to be stored on disk, and `STORAGE MEMORY` causes in-memory storage to be used. See [Section 13.1.17, “CREATE TABLE Syntax”](#), for more information.

Once table `ts_1` has been created as shown, you can perform `INSERT`, `SELECT`, `UPDATE`, and `DELETE` statements on it just as you would with any other MySQL table.

For table `dt_1` as it has been defined here, only the `dob` and `joined` columns are stored on disk. This is because there are indexes on the `id`, `last_name`, and `first_name` columns, and so data belonging to these columns is stored in RAM. In MySQL Cluster NDB 7.2, only nonindexed columns can be held on disk; indexes and indexed column data continue to be stored in memory. This tradeoff between the use of indexes and conservation of RAM is something you must keep in mind as you design Disk Data tables.

Performance note. The performance of a cluster using Disk Data storage is greatly improved if Disk Data files are kept on a separate physical disk from the data node file system. This must be done for each data node in the cluster to derive any noticeable benefit.

You may use absolute and relative file system paths with `ADD UNDOFILE` and `ADD DATAFILE`. Relative paths are calculated relative to the data node's data directory. You may also use symbolic links; see [Section 17.5.12.2, “Using Symbolic Links with Disk Data Objects”](#), for more information and examples.

A log file group, a tablespace, and any Disk Data tables using these must be created in a particular order. The same is true for dropping any of these objects:

- A log file group cannot be dropped as long as any tablespaces are using it.
- A tablespace cannot be dropped as long as it contains any data files.
- You cannot drop any data files from a tablespace as long as there remain any tables which are using the tablespace.
- It is not possible to drop files created in association with a different tablespace than the one with which the files were created. (Bug #20053)

For example, to drop all the objects created so far in this section, you would use the following statements:

```

mysql> DROP TABLE dt_1;

mysql> ALTER TABLESPACE ts_1
-> DROP DATAFILE 'data_2.dat'
-> ENGINE NDBCLUSTER;

mysql> ALTER TABLESPACE ts_1

```

```

-> DROP DATAFILE 'data_1.dat'
-> ENGINE NDBCLUSTER;

mysql> DROP TABLESPACE ts_1
-> ENGINE NDBCLUSTER;

mysql> DROP LOGFILE GROUP lg_1
-> ENGINE NDBCLUSTER;

```

These statements must be performed in the order shown, except that the two `ALTER TABLESPACE ... DROP DATAFILE` statements may be executed in either order.

You can obtain information about data files used by Disk Data tables by querying the `FILES` table in the `INFORMATION_SCHEMA` database. An extra “NULL row” provides additional information about undo log files. For more information and examples, see [Section 20.29.1](#), “The `INFORMATION_SCHEMA FILES` Table”.

It is also possible to view information about allocated and free disk space for each Disk Data table or table partition using the `ndb_desc` utility. For more information, see [Section 17.4.9](#), “`ndb_desc` — Describe NDB Tables”.

17.5.12.2. Using Symbolic Links with Disk Data Objects

The performance of a MySQL Cluster that uses Disk Data storage can be greatly improved by separating data node file systems from undo log files and tablespace data files and placing these on different disks. In early versions of MySQL Cluster, there was no direct support for this in MySQL Cluster, but it was possible to achieve this separation using symbolic links as described in this section. MySQL Cluster NDB 7.2 supports the data node configuration parameters `FileSystemPathDD` [2084], `FileSystemPathDataFiles` [2084], and `FileSystemPathUndoFiles` [2085], which make the use of symbolic links for this purpose unnecessary. For more information about these parameters, see [Disk Data file system parameters](#).

Each data node in the cluster creates a file system in the directory named `ndb_node_id_fs` under the data node's `DataDir` [2042] as defined in the `config.ini` file. In this example, we assume that each data node host has 3 disks, aliased as `/data0`, `/data1`, and `/data2`, and that the cluster's `config.ini` includes the following:

```
[ndbd default]
DataDir= /data0
```

Our objective is to place all Disk Data log files in `/data1`, and all Disk Data data files in `/data2`, on each data node host.



Note

In this example, we assume that the cluster's data node hosts are all using Linux operating systems. For other platforms, you may need to substitute your operating system's commands for those shown here.

To accomplish this, perform the following steps:

- Under the data node file system create symbolic links pointing to the other drives:

```

shell> cd /data0/ndb_2_fs
shell> ls
D1 D10 D11 D2 D8 D9 LCP
shell> ln -s /data0 dnlogs
shell> ln -s /data1 dndata

```

You should now have two symbolic links:

```
shell> ls -l --hide=D*
lrwxrwxrwx 1 user group 30 2007-03-19 13:58 dndata -> /data1
lrwxrwxrwx 1 user group 30 2007-03-19 13:59 dnlogs -> /data2
```

We show this only for the data node with node ID 2; however, you must do this for *each* data node.

- Now, in the `mysql` client, create a log file group and tablespace using the symbolic links, as shown here:

```
mysql> CREATE LOGFILE GROUP lg1
-> ADD UNDOFILE 'dnlogs/undo1.log'
-> INITIAL_SIZE 150M
-> UNDO_BUFFER_SIZE = 1M
-> ENGINE=NDBCLUSTER;

mysql> CREATE TABLESPACE ts1
-> ADD DATAFILE 'dndata/data1.log'
-> USE LOGFILE GROUP lg1
-> INITIAL_SIZE 1G
-> ENGINE=NDBCLUSTER;
```

Verify that the files were created and placed correctly as shown here:

```
shell> cd /data1
shell> ls -l
total 2099304
-rw-rw-r-- 1 user group 157286400 2007-03-19 14:02 undo1.dat

shell> cd /data2
shell> ls -l
total 2099304
-rw-rw-r-- 1 user group 1073741824 2007-03-19 14:02 data1.dat
```

- If you are running multiple data nodes on one host, you must take care to avoid having them try to use the same space for Disk Data files. You can make this easier by creating a symbolic link in each data node file system. Suppose you are using `/data0` for both data node file systems, but you wish to have the Disk Data files for both nodes on `/data1`. In this case, you can do something similar to what is shown here:

```
shell> cd /data0
shell> ln -s /data1/dn2 ndb_2_fs/dd
shell> ln -s /data1/dn3 ndb_3_fs/dd
shell> ls -l --hide=D* ndb_2_fs
lrwxrwxrwx 1 user group 30 2007-03-19 14:22 dd -> /data1/dn2
shell> ls -l --hide=D* ndb_3_fs
lrwxrwxrwx 1 user group 30 2007-03-19 14:22 dd -> /data1/dn3
```

- Now you can create a logfile group and tablespace using the symbolic link, like this:

```
mysql> CREATE LOGFILE GROUP lg1
-> ADD UNDOFILE 'dd/undo1.log'
-> INITIAL_SIZE 150M
-> UNDO_BUFFER_SIZE = 1M
-> ENGINE=NDBCLUSTER;

mysql> CREATE TABLESPACE ts1
-> ADD DATAFILE 'dd/data1.log'
-> USE LOGFILE GROUP lg1
-> INITIAL_SIZE 1G
-> ENGINE=NDBCLUSTER;
```

Verify that the files were created and placed correctly as shown here:

```
shell> cd /data1
shell> ls
```

```

dn2          dn3
shell> ls dn2
undo1.log   data1.log
shell> ls dn3
undo1.log   data1.log

```

17.5.12.3. MySQL Cluster Disk Data Storage Requirements

The following items apply to Disk Data storage requirements:

- Variable-length columns of Disk Data tables take up a fixed amount of space. For each row, this is equal to the space required to store the largest possible value for that column.

For general information about calculating these values, see [Section 11.6, “Data Type Storage Requirements”](#).

You can obtain an estimate the amount of space available in data files and undo log files by querying the `INFORMATION_SCHEMA.FILES` table. For more information and examples, see [Section 20.29.1, “The INFORMATION_SCHEMA FILES Table”](#).



Note

The `OPTIMIZE TABLE` statement does not have any effect on Disk Data tables.

- In a Disk Data table, the first 256 bytes of a `TEXT` or `BLOB` column are stored in memory; only the remainder is stored on disk.
- Each row in a Disk Data table uses 8 bytes in memory to point to the data stored on disk. This means that, in some cases, converting an in-memory column to the disk-based format can actually result in greater memory usage. For example, converting a `CHAR(4)` column from memory-based to disk-based format increases the amount of `DataMemory` [2043] used per row from 4 to 8 bytes.



Important

Starting the cluster with the `--initial` option does *not* remove Disk Data files. You must remove these manually prior to performing an initial restart of the cluster.

Performance of Disk Data tables can be improved by minimizing the number of disk seeks by making sure that `DiskPageBufferMemory` [2083] is of sufficient size. You can query the `diskpagebuffer` table to help determine whether the value for this parameter needs to be increased.

17.5.13. Adding MySQL Cluster Data Nodes Online

This section describes how to add MySQL Cluster data nodes “online”—that is, without needing to shut down the cluster completely and restart it as part of the process.



Important

Currently, you must add new data nodes to a MySQL Cluster as part of a new node group. In addition, it is not possible to change the number of replicas (or the number of nodes per node group) online.

17.5.13.1. Adding MySQL Cluster Data Nodes Online: General Issues

This section provides general information about the behavior of and current limitations in adding MySQL Cluster nodes online.

Redistribution of Data. The ability to add new nodes online includes a means to reorganize `NDBCLUSTER` table data and indexes so that they are distributed across all data nodes, including the new ones. Table reorganization of both in-memory and Disk Data tables is supported. This redistribution does not currently include unique indexes (only ordered indexes are redistributed) or `BLOB` table data.

The redistribution for `NDBCLUSTER` tables already existing before the new data nodes were added is not automatic, but can be accomplished using simple SQL statements in `mysql` or another MySQL client application. However, all data and indexes added to tables created after a new node group has been added are distributed automatically among all cluster data nodes, including those added as part of the new node group.

Partial starts. It is possible to add a new node group without all of the new data nodes being started. It is also possible to add a new node group to a degraded cluster—that is, a cluster that is only partially started, or where one or more data nodes are not running. In the latter case, the cluster must have enough nodes running to be viable before the new node group can be added.

Effects on ongoing operations. Normal DML operations using MySQL Cluster data are not prevented by the creation or addition of a new node group, or by table reorganization. However, it is not possible to perform DDL concurrently with table reorganization—that is, no other DDL statements can be issued while an `ALTER TABLE ... REORGANIZE PARTITION` statement is executing. In addition, during the execution of `ALTER TABLE ... REORGANIZE PARTITION` (or the execution of any other DDL statement), it is not possible to restart cluster data nodes.

Failure handling. Failures of data nodes during node group creation and table reorganization are handled as hown in the following table:

Failure occurs during:	Failure occurs in:		
	“Old” data nodes	“New” data nodes	System
Node group creation	<ul style="list-style-type: none"> • If a node other than the master fails: The creation of the node group is always rolled forward. • If the master fails: <ul style="list-style-type: none"> • If the internal commit point has been reached: The creation of the node group is rolled forward. • If the internal commit point has not yet been reached. The creation of the node group is rolled back 	<ul style="list-style-type: none"> • If a node other than the master fails: The creation of the node group is always rolled forward. • If the master fails: <ul style="list-style-type: none"> • If the internal commit point has been reached: The creation of the node group is rolled forward. • If the internal commit point has not yet been reached. The creation of the node group is rolled back 	<ul style="list-style-type: none"> • If the execution of <code>CREATE NODEGROUP</code> has reached the internal commit point: When restarted, the cluster includes the new node group. Otherwise it without. • If the execution of <code>CREATE NODEGROUP</code> has not yet reached the internal commit point: When restarted, the cluster does not include the new node group.
Table reorganization	<ul style="list-style-type: none"> • If a node other than the master fails: The table 	<ul style="list-style-type: none"> • If a node other than the master fails: The table 	<ul style="list-style-type: none"> • If the execution of an <code>ALTER ONLINE TABLE table REORGANIZE</code>

Failure occurs during:	Failure occurs in:		
	“Old” data nodes	“New” data nodes	System
	reorganization is always rolled forward. <ul style="list-style-type: none"> • If the master fails: <ul style="list-style-type: none"> • If the internal commit point has been reached: The table reorganization is rolled forward. • If the internal commit point has not yet been reached. The table reorganization is rolled back. 	reorganization is always rolled forward. <ul style="list-style-type: none"> • If the master fails: <ul style="list-style-type: none"> • If the internal commit point has been reached: The table reorganization is rolled forward. • If the internal commit point has not yet been reached. The table reorganization is rolled back. 	PARTITION statement has reached the internal commit point: When the cluster is restarted, the data and indexes belonging to <i>table</i> are distributed using the “new” data nodes. <ul style="list-style-type: none"> • If the execution of an ALTER ONLINE TABLE table REORGANIZE PARTITION statement has not yet reached the internal commit point: When the cluster is restarted, the data and indexes belonging to <i>table</i> are distributed using only the “old” data nodes.

Dropping node groups. The `ndb_mgm` client supports a `DROP NODEGROUP` command, but it is possible to drop a node group only when no data nodes in the node group contain any data. Since there is currently no way to “empty” a specific data node or node group, this command works only the following two cases:

1. After issuing `CREATE NODEGROUP` in the `ndb_mgm` client, but before issuing any `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statements in the `mysql` client.
2. After dropping all `NDBCLUSTER` tables using `DROP TABLE`.

`TRUNCATE TABLE` does not work for this purpose because the data nodes continue to store the table definitions.

17.5.13.2. Adding MySQL Cluster Data Nodes Online: Basic procedure

In this section, we list the basic steps required to add new data nodes to a MySQL Cluster. This procedure applies whether you are using `ndbd` or `ndbmta` binaries for the data node processes. For a more detailed example, see [Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”](#).

Assuming that you already have a running MySQL Cluster, adding data nodes online requires the following steps:

1. Edit the cluster configuration `config.ini` file, adding new `[ndbd]` sections corresponding to the nodes to be added. In the case where the cluster uses multiple management servers, these changes need to be made to all `config.ini` files used by the management servers.

You must be careful that node IDs for any new data nodes added in the `config.ini` file do not overlap node IDs used by existing nodes. In the event that you have API nodes using dynamically

allocated node IDs and these IDs match node IDs that you want to use for new data nodes, it is possible to force any such API nodes to “migrate”, as described later in this procedure.

2. Perform a rolling restart of all MySQL Cluster management servers.



Important

All management servers must be restarted with the `--reload [2191]` or `--initial [2190]` option to force the reading of the new configuration.

3. Perform a rolling restart of all existing MySQL Cluster data nodes. It is not necessary (or usually even desirable) to use `--initial [2178]` when restarting the existing data nodes.

If you are using API nodes with dynamically allocated IDs matching any node IDs that you wish to assign to new data nodes, you must restart all API nodes (including SQL nodes) before restarting any of the data nodes processes in this step. This causes any API nodes with node IDs that were previously not explicitly assigned to relinquish those node IDs and acquire new ones.

4. Perform a rolling restart of any SQL or API nodes connected to the MySQL Cluster.
5. Start the new data nodes.

The new data nodes may be started in any order. They can also be started concurrently, as long as they are started after the rolling restarts of all existing data nodes have been completed, and before proceeding to the next step.

6. Execute one or more `CREATE NODEGROUP` commands in the MySQL Cluster management client to create the new node group or node groups to which the new data nodes will belong.
7. Redistribute the cluster's data among all data nodes (including the new ones) by issuing an `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement in the `mysql` client for each `NDBCLUSTER` table.



Note

This needs to be done only for tables already existing at the time the new node group is added. Data in tables created after the new node group is added is distributed automatically; however, data added to any given table `tbl` that existed before the new nodes were added is not distributed using the new nodes until that table has been reorganized using `ALTER ONLINE TABLE tbl REORGANIZE PARTITION`.

8. Reclaim the space freed on the “old” nodes by issuing, for each `NDBCLUSTER` table, an `OPTIMIZE TABLE` statement in the `mysql` client.

You can add all the nodes desired, then issue several `CREATE NODEGROUP` commands in succession to add the new node groups to the cluster.

17.5.13.3. Adding MySQL Cluster Data Nodes Online: Detailed Example

In this section we provide a detailed example illustrating how to add new MySQL Cluster data nodes online, starting with a MySQL Cluster having 2 data nodes in a single node group and concluding with a cluster having 4 data nodes in 2 node groups.

Starting configuration. For purposes of illustration, we assume a minimal configuration, and that the cluster uses a `config.ini` file containing only the following information:

```
[ndb default]
```

```
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 192.168.0.1

[ndbd]
Id = 2
HostName = 192.168.0.2

[mgm]
HostName = 192.168.0.10
Id = 10

[api]
Id=20
HostName = 192.168.0.20

[api]
Id=21
HostName = 192.168.0.21
```



Note

We have left a gap in the sequence between data node IDs and other nodes. This make it easier later to assign node IDs that are not already in use to data nodes which are newly added.

We also assume that you have already started the cluster using the appropriate command line or `my.cnf` options, and that running `SHOW` in the management client produces output similar to what is shown here:

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 192.168.0.10:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=1 @192.168.0.1 (5.5.29-ndb-7.2.10, Nodegroup: 0, Master)
id=2 @192.168.0.2 (5.5.29-ndb-7.2.10, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10 @192.168.0.10 (5.5.29-ndb-7.2.10)

[mysqld(API)] 2 node(s)
id=20 @192.168.0.20 (5.5.29-ndb-7.2.10)
id=21 @192.168.0.21 (5.5.29-ndb-7.2.10)
```

Finally, we assume that the cluster contains a single `NDBCLUSTER` table created as shown here:

```
USE n;

CREATE TABLE ips (
  id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  country_code CHAR(2) NOT NULL,
  type CHAR(4) NOT NULL,
  ip_address varchar(15) NOT NULL,
  addresses BIGINT UNSIGNED DEFAULT NULL,
  date BIGINT UNSIGNED DEFAULT NULL
) ENGINE NDBCLUSTER;
```

The memory usage and related information shown later in this section was generated after inserting approximately 50000 rows into this table.

**Note**

In this example, we show the single-threaded `ndbd` being used for the data node processes. However—beginning with MySQL Cluster NDB 7.0.4—you can also apply this example, if you are using the multi-threaded `ndbmt_d` by substituting `ndbmt_d` for `ndbd` wherever it appears in the steps that follow. (Bug #43108)

Step 1: Update configuration file. Open the cluster global configuration file in a text editor and add `[ndbd]` sections corresponding to the 2 new data nodes. (We give these data nodes IDs 3 and 4, and assume that they are to be run on host machines at addresses 192.168.0.3 and 192.168.0.4, respectively.) After you have added the new sections, the contents of the `config.ini` file should look like what is shown here, where the additions to the file are shown in bold type:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 192.168.0.1

[ndbd]
Id = 2
HostName = 192.168.0.2
[ndbd]
Id = 3
HostName = 192.168.0.3

[ndbd]
Id = 4
HostName = 192.168.0.4

[mgm]
HostName = 192.168.0.10
Id = 10

[api]
Id=20
HostName = 192.168.0.20

[api]
Id=21
HostName = 192.168.0.21
```

Once you have made the necessary changes, save the file.

Step 2: Restart the management server. Restarting the cluster management server requires that you issue separate commands to stop the management server and then to start it again, as follows:

1. Stop the management server using the management client `STOP` command, as shown here:

```
ndb_mgm> 10 STOP
Node 10 has shut down.
Disconnecting to allow Management Server to shutdown

shell>
```

2. Because shutting down the management server causes the management client to terminate, you must start the management server from the system shell. For simplicity, we assume that `config.ini` is in the same directory as the management server binary, but in practice, you must supply the correct path to the configuration file. You must also supply the `--reload [2191]` or `--initial [2190]` option

so that the management server reads the new configuration from the file rather than its configuration cache. If your shell's current directory is also the same as the directory where the management server binary is located, then you can invoke the management server as shown here:

```
shell> ndb_mgmd -f config.ini --reload
2008-12-08 17:29:23 [MgmSrvr] INFO      -- NDB Cluster Management Server. 5.5.29-ndb-7.2.10
2008-12-08 17:29:23 [MgmSrvr] INFO      -- Reading cluster configuration from 'config.ini'
```

If you check the output of `SHOW` in the management client after restarting the `ndb_mgm` process, you should now see something like this:

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 192.168.0.10:1186
Cluster Configuration
-----
[ndbd(NDB)]      2 node(s)
id=1      @192.168.0.1  (5.5.29-ndb-7.2.10, Nodegroup: 0, Master)
id=2      @192.168.0.2  (5.5.29-ndb-7.2.10, Nodegroup: 0)
id=3 (not connected, accepting connect from 192.168.0.3)
id=4 (not connected, accepting connect from 192.168.0.4)

[ndb_mgmd(MGM)] 1 node(s)
id=10     @192.168.0.10  (5.5.29-ndb-7.2.10)

[mysqld(API)]   2 node(s)
id=20     @192.168.0.20  (5.5.29-ndb-7.2.10)
id=21     @192.168.0.21  (5.5.29-ndb-7.2.10)
```

Step 3: Perform a rolling restart of the existing data nodes. This step can be accomplished entirely within the cluster management client using the `RESTART` command, as shown here:

```
ndb_mgm> 1 RESTART
Node 1: Node shutdown initiated
Node 1: Node shutdown completed, restarting, no start.
Node 1 is being restarted

ndb_mgm> Node 1: Start initiated (version 7.2.10)
Node 1: Started (version 7.1.26)

ndb_mgm> 2 RESTART
Node 2: Node shutdown initiated
Node 2: Node shutdown completed, restarting, no start.
Node 2 is being restarted

ndb_mgm> Node 2: Start initiated (version 7.2.10)

ndb_mgm> Node 2: Started (version 7.2.10)
```



Important

After issuing each `X RESTART` command, wait until the management client reports `Node X: Started (version ...)` before proceeding any further.

You can verify that all existing data nodes were restarted using the updated configuration by checking the `ndbinfo.nodes` table in the `mysql` client.

Step 4: Perform a rolling restart of all cluster API nodes. Shut down and restart each MySQL server acting as an SQL node in the cluster using `mysqladmin shutdown` followed by `mysqld_safe` (or another startup script). This should be similar to what is shown here, where `password` is the MySQL `root` password for a given MySQL server instance:

```
shell> mysqladmin -uroot -ppassword shutdown
081208 20:19:56 mysqld_safe mysqld from pid file
```

```

/usr/local/mysql/var/tonfisk.pid ended
shell> mysqld_safe --ndbcluster --ndb-connectstring=192.168.0.10 &
081208 20:20:06 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
081208 20:20:06 mysqld_safe Starting mysqld daemon with databases
from /usr/local/mysql/var
    
```

Of course, the exact input and output depend on how and where MySQL is installed on the system, as well as which options you choose to start it (and whether or not some or all of these options are specified in a `my.cnf` file).

Step 5: Perform an initial start of the new data nodes. From a system shell on each of the hosts for the new data nodes, start the data nodes as shown here, using the `--initial` [2178] option:

```
shell> ndbd -c 192.168.0.10 --initial
```



Note

Unlike the case with restarting the existing data nodes, you can start the new data nodes concurrently; you do not need to wait for one to finish starting before starting the other.

Wait until both of the new data nodes have started before proceeding with the next step. Once the new data nodes have started, you can see in the output of the management client `SHOW` command that they do not yet belong to any node group (as indicated with bold type here):

```

ndb_mgm> SHOW
Connected to Management Server at: 192.168.0.10:1186
Cluster Configuration
-----
[ndbd(NDB)]      2 node(s)
id=1   @192.168.0.1  (5.5.29-ndb-7.2.10, Nodegroup: 0, Master)
id=2   @192.168.0.2  (5.5.29-ndb-7.2.10, Nodegroup: 0)
id=3   @192.168.0.3  (5.5.29-ndb-7.2.10, no nodegroup)
id=4   @192.168.0.4  (5.5.29-ndb-7.2.10, no nodegroup)

[ndb_mgmd(MGM)] 1 node(s)
id=10  @192.168.0.10  (5.5.29-ndb-7.2.10)

[mysqld(API)]   2 node(s)
id=20  @192.168.0.20  (5.5.29-ndb-7.2.10)
id=21  @192.168.0.21  (5.5.29-ndb-7.2.10)
    
```

Step 6: Create a new node group. You can do this by issuing a `CREATE NODEGROUP` command in the cluster management client. This command takes as its argument a comma-separated list of the node IDs of the data nodes to be included in the new node group, as shown here:

```
ndb_mgm> CREATE NODEGROUP 3,4
Nodegroup 1 created
```

By issuing `SHOW` again, you can verify that data nodes 3 and 4 have joined the new node group (again indicated in bold type):

```

ndb_mgm> SHOW
Connected to Management Server at: 192.168.0.10:1186
Cluster Configuration
-----
[ndbd(NDB)]      2 node(s)
id=1   @192.168.0.1  (5.5.29-ndb-7.2.10, Nodegroup: 0, Master)
id=2   @192.168.0.2  (5.5.29-ndb-7.2.10, Nodegroup: 0)
id=3   @192.168.0.3  (5.5.29-ndb-7.2.10, Nodegroup: 1)
id=4   @192.168.0.4  (5.5.29-ndb-7.2.10, Nodegroup: 1)

[ndb_mgmd(MGM)] 1 node(s)
id=10  @192.168.0.10  (5.5.29-ndb-7.2.10)
    
```

```
[mysqld(API)] 2 node(s)
id=20 @192.168.0.20 (5.5.29-ndb-7.2.10)
id=21 @192.168.0.21 (5.5.29-ndb-7.2.10)
```

Step 7: Redistribute cluster data. When a node group is created, existing data and indexes are not automatically distributed to the new node group's data nodes, as you can see by issuing the appropriate `REPORT` command in the management client:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(177 32K pages of total 3200)
Node 1: Index usage is 0%(108 8K pages of total 12832)
Node 2: Data usage is 5%(177 32K pages of total 3200)
Node 2: Index usage is 0%(108 8K pages of total 12832)
Node 3: Data usage is 0%(0 32K pages of total 3200)
Node 3: Index usage is 0%(0 8K pages of total 12832)
Node 4: Data usage is 0%(0 32K pages of total 3200)
Node 4: Index usage is 0%(0 8K pages of total 12832)
```

By using `ndb_desc` with the `-p` option, which causes the output to include partitioning information, you can see that the table still uses only 2 partitions (in the `Per partition info` section of the output, shown here in bold text):

```
shell> ndb_desc -c 192.168.0.10 -d n ips -p
-- ips --
Version: 1
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 340
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition   Row count   Commit count   Frag fixed memory   Frag varsized memory
0             26086         26086             1572864               557056
1             26329         26329             1605632               557056

NDBT_ProgramExit: 0 - OK
```

You can cause the data to be redistributed among all of the data nodes by performing, for each `NDBCLUSTER` table, an `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement in the `mysql` client. After issuing the statement `ALTER ONLINE TABLE ips REORGANIZE PARTITION`, you can see using `ndb_desc` that the data for this table is now stored using 4 partitions, as shown here (with the relevant portions of the output in bold type):

```

shell> ndb_desc -c 192.168.0.10 -d n ips -p
-- ips --
Version: 16777217
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 341
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 4
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory  Frag var sized memory
0          12981      52296         1572864             557056
1          13236      52515         1605632             557056
2          13105      13105         819200              294912
3          13093      13093         819200              294912

NDBT_ProgramExit: 0 - OK

```



Note

Normally, `ALTER [ONLINE] TABLE table_name REORGANIZE PARTITION` is used with a list of partition identifiers and a set of partition definitions to create a new partitioning scheme for a table that has already been explicitly partitioned. Its use here to redistribute data onto a new MySQL Cluster node group is an exception in this regard; when used in this way, only the name of the table is used following the `TABLE` keyword, and no other keywords or identifiers follow `REORGANIZE PARTITION`.

For more information, see [Section 13.1.7, “ALTER TABLE Syntax”](#).

In addition, for each table, the `ALTER ONLINE TABLE` statement should be followed by an `OPTIMIZE TABLE` to reclaim wasted space. You can obtain a list of all `NDBCLUSTER` tables using the following query against the `INFORMATION_SCHEMA.TABLES` table:

```

SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE ENGINE = 'NDBCLUSTER';

```



Note

The `INFORMATION_SCHEMA.TABLES.ENGINE` value for a MySQL Cluster table is always `NDBCLUSTER`, regardless of whether the `CREATE TABLE` statement used to

create the table (or `ALTER TABLE` statement used to convert an existing table from a different storage engine) used `NDB` or `NDBCLUSTER` in its `ENGINE` option.

You can see after performing these statements in the output of `ALL REPORT MEMORY` that the data and indexes are now redistributed between all cluster data nodes, as shown here:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(176 32K pages of total 3200)
Node 1: Index usage is 0%(76 8K pages of total 12832)
Node 2: Data usage is 5%(176 32K pages of total 3200)
Node 2: Index usage is 0%(76 8K pages of total 12832)
Node 3: Data usage is 2%(80 32K pages of total 3200)
Node 3: Index usage is 0%(51 8K pages of total 12832)
Node 4: Data usage is 2%(80 32K pages of total 3200)
Node 4: Index usage is 0%(50 8K pages of total 12832)
```



Note

Since only one DDL operation on `NDBCLUSTER` tables can be executed at a time, you must wait for each `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement to finish before issuing the next one.

It is not necessary to issue `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statements for `NDBCLUSTER` tables created *after* the new data nodes have been added; data added to such tables is distributed among all data nodes automatically. However, in `NDBCLUSTER` tables that existed *prior* to the addition of the new nodes, neither existing nor new data is distributed using the new nodes until these tables have been reorganized using `ALTER ONLINE TABLE ... REORGANIZE PARTITION`.

Alternative procedure, without rolling restart. It is possible to avoid the need for a rolling restart by configuring the extra data nodes, but not starting them, when first starting the cluster. We assume, as before, that you wish to start with two data nodes—nodes 1 and 2—in one node group and later to expand the cluster to four data nodes, by adding a second node group consisting of nodes 3 and 4:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 192.168.0.1

[ndbd]
Id = 2
HostName = 192.168.0.2

[ndbd]
Id = 3
HostName = 192.168.0.3
Nodegroup = 65536

[ndbd]
Id = 4
HostName = 192.168.0.4
Nodegroup = 65536

[mgm]
HostName = 192.168.0.10
Id = 10

[api]
Id=20
```

```
HostName = 192.168.0.20
```

```
[api]
Id=21
HostName = 192.168.0.21
```

**Note**

In MySQL Cluster NDB 7.2, it is no longer necessary to perform the initial start of the cluster using `--nowait-nodes` [2180] option with `ndbd` or `ndbmtbd` as it was in some earlier versions of MySQL Cluster.

The data nodes to be brought online at a later time (nodes 3 and 4) can be configured with `NodeGroup = 65536` [2040], in which case nodes 1 and 2 can each be started as shown here:

```
shell> ndbd -c 192.168.0.10 --initial
```

The data nodes configured with `NodeGroup = 65536` [2040] are treated by the management server as though you had started nodes 1 and 2 using `--nowait-nodes=3,4` [2180] after waiting for a period of time determined by the setting for the `StartNoNodeGroupTimeout` [2062] data node configuration parameter. By default, this is 15 seconds (15000 milliseconds).

**Note**

`StartNoNodegroupTimeout` [2062] must be the same for all data nodes in the cluster; for this reason, you should always set it in the `[ndbd default]` section of the `config.ini` file, rather than for individual data nodes.

When you are ready to add the second node group, you need only perform the following additional steps:

1. Start data nodes 3 and 4, invoking the data node process once for each new node:

```
shell> ndbd -c 192.168.0.10 --initial
```

2. Issue the appropriate `CREATE NODEGROUP` command in the management client:

```
ndb_mgm> CREATE NODEGROUP 3,4
```

3. In the `mysql` client, issue `ALTER ONLINE TABLE ... REORGANIZE PARTITION` and `OPTIMIZE TABLE` statements for each existing `NDBCLUSTER` table. (As noted elsewhere in this section, existing MySQL Cluster tables cannot use the new nodes for data distribution until this has been done.)

17.5.14. Distributed MySQL Privileges for MySQL Cluster

MySQL Cluster NDB 7.2 introduces support for distributing MySQL users and privileges across all SQL nodes in a MySQL Cluster. This support is not enabled by default; you should follow the procedure outlined in this section in order to do so.

Normally, each MySQL server's user privilege tables in the `mysql` database must use the `MyISAM` storage engine, which means that a user account and its associated privileges created on one SQL node are not available on the cluster's other SQL nodes. In MySQL Cluster NDB 7.2 and later, an SQL file `ndb_dist_priv.sql` is provided with the MySQL Cluster distribution. This file can be found in the `share` directory in the MySQL installation directory.

The first step in enabling distributed privileges is to load this script into a MySQL Server that functions as an SQL node (which we refer to after this as the *target* SQL node or MySQL Server). You can do this by executing the following command from the system shell on the target SQL node after changing to its MySQL installation directory (where `options` stands for any additional options needed to connect to this SQL node):

```
shell> mysql options -uroot < share/ndb_dist_priv.sql
```

Importing `ndb_dist_priv.sql` creates a number of stored routines (six stored procedures and one stored function) in the `mysql` database on the target SQL node. After connecting to the SQL node in the `mysql` client (as the MySQL `root` user), you can verify that these were created as shown here:

```
mysql> SELECT ROUTINE_NAME, ROUTINE_SCHEMA, ROUTINE_TYPE
-> FROM INFORMATION_SCHEMA.ROUTINES
-> WHERE ROUTINE_NAME LIKE 'mysql_cluster%'
-> ORDER BY ROUTINE_TYPE;
```

ROUTINE_NAME	ROUTINE_SCHEMA	ROUTINE_TYPE
mysql_cluster_privileges_are_distributed	mysql	FUNCTION
mysql_cluster_backup_privileges	mysql	PROCEDURE
mysql_cluster_move_grant_tables	mysql	PROCEDURE
mysql_cluster_move_privileges	mysql	PROCEDURE
mysql_cluster_restore_local_privileges	mysql	PROCEDURE
mysql_cluster_restore_privileges	mysql	PROCEDURE
mysql_cluster_restore_privileges_from_local	mysql	PROCEDURE

```
7 rows in set (0.01 sec)
```

The stored procedure named `mysql_cluster_move_privileges` creates backup copies of the existing privilege tables, then converts them to `NDB`. Two sets of copies are created in the `mysql` database:

- A set of local copies that use the `MyISAM` storage engine, and named by adding the suffix `_backup` to the original privilege table names.
- A set of distributed copies (using `NDBCLUSTER`). These tables are named by prefixing `ndb_` and appending `_backup` to the names of the original tables.

Although the original privilege tables are backed up automatically, it is always a good idea to create backups manually of the existing privilege tables on all affected SQL nodes before proceeding. You can do this using `mysqldump` in a manner similar to what is shown here:

```
shell> mysqldump options -uroot \
mysql host user db tables_priv columns_priv procs_priv proxies_priv > backup_file
```

To perform the conversion, you must be connected to the target SQL node using the `mysql` client (again, as the MySQL `root` user). Invoke the stored procedure like this:

```
mysql> CALL mysql.mysql_cluster_move_privileges();
Query OK, 0 rows affected (22.32 sec)
```

Depending on the number of rows in the privilege tables, this procedure may take some time to execute. If some of the privilege tables are empty, you may see one or more `No data - zero rows fetched, selected, or processed` warnings when `mysql_cluster_move_privileges` returns. In such cases, the warnings may be safely ignored. To verify that the conversion was successful, you can use the stored function `mysql_cluster_privileges_are_distributed` as shown here:

```
mysql> SELECT CONCAT(
-> 'Conversion ',
-> IF(mysql.mysql_cluster_privileges_are_distributed(), 'succeeded', 'failed'),
-> '.')
-> AS Result;
```

Result
Conversion succeeded.

```
1 row in set (0.00 sec)
```

`mysql_cluster_privileges_are_distributed` checks for the existence of the distributed privilege tables and returns 1 if all of the privilege tables are distributed; otherwise, it returns 0.

You can verify that the backups have been created using a query such as this one:

```
mysql> SELECT TABLE_NAME, ENGINE FROM INFORMATION_SCHEMA.TABLES
->     WHERE TABLE_SCHEMA = 'mysql' AND TABLE_NAME LIKE '%backup'
->     ORDER BY ENGINE;
```

TABLE_NAME	ENGINE
host_backup	MyISAM
db_backup	MyISAM
columns_priv_backup	MyISAM
user_backup	MyISAM
tables_priv_backup	MyISAM
proxies_priv_backup	MyISAM
procs_priv_backup	MyISAM
ndb_user_backup	ndbcluster
ndb_tables_priv_backup	ndbcluster
ndb_proxies_priv_backup	ndbcluster
ndb_procs_priv_backup	ndbcluster
ndb_host_backup	ndbcluster
ndb_db_backup	ndbcluster
ndb_columns_priv_backup	ndbcluster

```
14 rows in set (0.00 sec)
```

Once the conversion to distributed privileges has been made, any time a MySQL user account is created, dropped, or has its privileges updated on any SQL node, the changes take effect immediately on all other MySQL servers attached to the cluster. Once privileges are distributed, any new MySQL Servers that connect to the cluster automatically participate in the distribution.



Note

Formerly, after `mysql_cluster_move_privileges` was executed, you sometimes needed to execute `FLUSH PRIVILEGES` on those SQL nodes, or to disconnect and then reconnect the clients, in order for those clients to be able to see the changes in privileges. In MySQL Cluster NDB 7.2.4 and later, this is no longer necessary, as SQL nodes automatically execute a `FLUSH PRIVILEGES` statement when joining a MySQL Cluster where distributed privileges are in use.

All MySQL user privileges are distributed across all connected MySQL Servers. This includes privileges associated with views and stored routines. While automatic distribution of views and stored routines is not currently supported, you can attempt to distribute stored routines by issuing a statement such as `ALTER TABLE mysql.proc ENGINE = NDB`, but you must verify manually that any tables referenced by the stored routines exist on all SQL nodes, since MySQL Cluster has no support at the present time for doing this automatically. There is currently no way to distribute views among MySQL Cluster SQL nodes, other than by creating them manually on each SQL node. If you do this, you must make certain that all base tables referenced by the views use the `NDB` storage engine; otherwise, the views are likely to diverge very quickly.

In the event that an SQL node becomes disconnected from the cluster while `mysql_cluster_move_privileges` is running, you must drop its privilege tables after reconnecting to the cluster, using a statement such as `DROP TABLE IF EXISTS mysql.user mysql.db mysql.tables_priv mysql.columns_priv mysql.procs_priv`. This causes the SQL node to use the shared privilege tables rather than its own local versions of them. This is not needed when connecting a new SQL node to the cluster for the first time.

In the event of an initial restart of the entire cluster (all data nodes shut down, then started again with `--initial` [2178]), the shared privilege tables are lost. If this happens, you can restore them using the original target SQL node either from the backups made by `mysql_cluster_move_privileges` or from a dump file created with `mysqldump`. If you need to use a new MySQL Server to perform the restoration, you should start it with `--skip-grant-tables` [461] when connecting to the cluster for the first time; after this, you can restore the privilege tables locally, then distribute them again using `mysql_cluster_move_privileges`. After restoring and distributing the tables, you should restart this MySQL Server without the `--skip-grant-tables` [461] option.

You can also restore the distributed tables using `ndb_restore --restore-privilege-tables` [2222] from a backup made using `START BACKUP` in the `ndb_mgm` client. (The MyISAM tables created by `mysql_cluster_move_privileges` are not backed up by the `START BACKUP` command.) `ndb_restore` does not restore the privilege tables by default; the `--restore-privilege-tables` [2222] option causes it to do so.



Important

Applications that access MySQL Cluster data directly, including NDB API and ClusterJ applications, are not subject to the MySQL privilege system. This means that, once you have distributed the grant tables, they can be freely accessed by such applications, just as they can any other NDB tables. In particular, you should keep in mind that *NDB API and ClusterJ applications can read and write user names, host names, password hashes, and any other contents of the distributed grant tables without any restrictions.*

17.5.15. NDB API Statistics Counters and Variables

A number of types of statistical counters relating to actions performed by or affecting Ndb objects are available. Such actions include starting and closing (or aborting) transactions; primary key and unique key operations; table, range, and pruned scans; threads blocked while waiting for the completion of various operations; and data and events sent and received by NDBCLUSTER. The counters are incremented inside the NDB kernel whenever NDB API calls are made or data is sent to or received by the data nodes. `mysqld` exposes these counters as system status variables; their values can be read in the output of `SHOW STATUS`, or by querying the `INFORMATION_SCHEMA.SESSION_STATUS` or `INFORMATION_SCHEMA.GLOBAL_STATUS` table. By comparing the values before and after statements operating on NDB tables, you can observe the corresponding actions taken on the API level, and thus the cost of performing the statement.

You can list all of these status variables using the following `SHOW STATUS` statement:

```
mysql> SHOW STATUS LIKE 'ndb_api%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_api_wait_exec_complete_count_session | 0 |
| Ndb_api_wait_scan_result_count_session | 0 |
| Ndb_api_wait_meta_request_count_session | 0 |
| Ndb_api_wait_nanos_count_session | 0 |
| Ndb_api_bytes_sent_count_session | 0 |
| Ndb_api_bytes_received_count_session | 0 |
| Ndb_api_trans_start_count_session | 0 |
| Ndb_api_trans_commit_count_session | 0 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 0 |
| Ndb_api_pk_op_count_session | 0 |
| Ndb_api_uk_op_count_session | 0 |
| Ndb_api_table_scan_count_session | 0 |
| Ndb_api_range_scan_count_session | 0 |
| Ndb_api_pruned_scan_count_session | 0 |
+-----+-----+
```

NDB API Statistics Counters and Variables

Ndb_api_scan_batch_count_session	0
Ndb_api_read_row_count_session	0
Ndb_api_trans_local_read_row_count_session	0
Ndb_api_event_data_count_injector	0
Ndb_api_event_nodata_count_injector	0
Ndb_api_event_bytes_count_injector	0
Ndb_api_wait_exec_complete_count_slave	0
Ndb_api_wait_scan_result_count_slave	0
Ndb_api_wait_meta_request_count_slave	0
Ndb_api_wait_nanos_count_slave	0
Ndb_api_bytes_sent_count_slave	0
Ndb_api_bytes_received_count_slave	0
Ndb_api_trans_start_count_slave	0
Ndb_api_trans_commit_count_slave	0
Ndb_api_trans_abort_count_slave	0
Ndb_api_trans_close_count_slave	0
Ndb_api_pk_op_count_slave	0
Ndb_api_uk_op_count_slave	0
Ndb_api_table_scan_count_slave	0
Ndb_api_range_scan_count_slave	0
Ndb_api_pruned_scan_count_slave	0
Ndb_api_scan_batch_count_slave	0
Ndb_api_read_row_count_slave	0
Ndb_api_trans_local_read_row_count_slave	0
Ndb_api_wait_exec_complete_count	2
Ndb_api_wait_scan_result_count	3
Ndb_api_wait_meta_request_count	27
Ndb_api_wait_nanos_count	45612023
Ndb_api_bytes_sent_count	992
Ndb_api_bytes_received_count	9640
Ndb_api_trans_start_count	2
Ndb_api_trans_commit_count	1
Ndb_api_trans_abort_count	0
Ndb_api_trans_close_count	2
Ndb_api_pk_op_count	1
Ndb_api_uk_op_count	0
Ndb_api_table_scan_count	1
Ndb_api_range_scan_count	0
Ndb_api_pruned_scan_count	0
Ndb_api_scan_batch_count	0
Ndb_api_read_row_count	1
Ndb_api_trans_local_read_row_count	1
Ndb_api_event_data_count	0
Ndb_api_event_nodata_count	0
Ndb_api_event_bytes_count	0

+-----+-----+

60 rows in set (0.02 sec)

These status variables are also available from the [SESSION_STATUS](#) and [GLOBAL_STATUS](#) tables of the [INFORMATION_SCHEMA](#) database, as shown here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS
-> WHERE VARIABLE_NAME LIKE 'ndb_api%';
```

VARIABLE_NAME	VARIABLE_VALUE
NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION	2
NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION	0
NDB_API_WAIT_META_REQUEST_COUNT_SESSION	1
NDB_API_WAIT_NANOS_COUNT_SESSION	8144375
NDB_API_BYTES_SENT_COUNT_SESSION	68
NDB_API_BYTES_RECEIVED_COUNT_SESSION	84
NDB_API_TRANS_START_COUNT_SESSION	1
NDB_API_TRANS_COMMIT_COUNT_SESSION	1
NDB_API_TRANS_ABORT_COUNT_SESSION	0
NDB_API_TRANS_CLOSE_COUNT_SESSION	1
NDB_API_PK_OP_COUNT_SESSION	1

NDB API Statistics Counters and Variables

NDB_API_UK_OP_COUNT_SESSION	0
NDB_API_TABLE_SCAN_COUNT_SESSION	0
NDB_API_RANGE_SCAN_COUNT_SESSION	0
NDB_API_PRUNED_SCAN_COUNT_SESSION	0
NDB_API_SCAN_BATCH_COUNT_SESSION	0
NDB_API_READ_ROW_COUNT_SESSION	1
NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION	1
NDB_API_EVENT_DATA_COUNT_INJECTOR	0
NDB_API_EVENT_NONDATA_COUNT_INJECTOR	0
NDB_API_EVENT_BYTES_COUNT_INJECTOR	0
NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE	0
NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE	0
NDB_API_WAIT_META_REQUEST_COUNT_SLAVE	0
NDB_API_WAIT_NANOS_COUNT_SLAVE	0
NDB_API_BYTES_SENT_COUNT_SLAVE	0
NDB_API_BYTES_RECEIVED_COUNT_SLAVE	0
NDB_API_TRANS_START_COUNT_SLAVE	0
NDB_API_TRANS_COMMIT_COUNT_SLAVE	0
NDB_API_TRANS_ABORT_COUNT_SLAVE	0
NDB_API_TRANS_CLOSE_COUNT_SLAVE	0
NDB_API_PK_OP_COUNT_SLAVE	0
NDB_API_UK_OP_COUNT_SLAVE	0
NDB_API_TABLE_SCAN_COUNT_SLAVE	0
NDB_API_RANGE_SCAN_COUNT_SLAVE	0
NDB_API_PRUNED_SCAN_COUNT_SLAVE	0
NDB_API_SCAN_BATCH_COUNT_SLAVE	0
NDB_API_READ_ROW_COUNT_SLAVE	0
NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE	0
NDB_API_WAIT_EXEC_COMPLETE_COUNT	4
NDB_API_WAIT_SCAN_RESULT_COUNT	3
NDB_API_WAIT_META_REQUEST_COUNT	28
NDB_API_WAIT_NANOS_COUNT	53756398
NDB_API_BYTES_SENT_COUNT	1060
NDB_API_BYTES_RECEIVED_COUNT	9724
NDB_API_TRANS_START_COUNT	3
NDB_API_TRANS_COMMIT_COUNT	2
NDB_API_TRANS_ABORT_COUNT	0
NDB_API_TRANS_CLOSE_COUNT	3
NDB_API_PK_OP_COUNT	2
NDB_API_UK_OP_COUNT	0
NDB_API_TABLE_SCAN_COUNT	1
NDB_API_RANGE_SCAN_COUNT	0
NDB_API_PRUNED_SCAN_COUNT	0
NDB_API_SCAN_BATCH_COUNT	0
NDB_API_READ_ROW_COUNT	2
NDB_API_TRANS_LOCAL_READ_ROW_COUNT	2
NDB_API_EVENT_DATA_COUNT	0
NDB_API_EVENT_NONDATA_COUNT	0
NDB_API_EVENT_BYTES_COUNT	0

60 rows in set (0.00 sec)

```
mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_STATUS
-> WHERE VARIABLE_NAME LIKE 'ndb_api%';
```

VARIABLE_NAME	VARIABLE_VALUE
NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION	2
NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION	0
NDB_API_WAIT_META_REQUEST_COUNT_SESSION	1
NDB_API_WAIT_NANOS_COUNT_SESSION	8144375
NDB_API_BYTES_SENT_COUNT_SESSION	68
NDB_API_BYTES_RECEIVED_COUNT_SESSION	84
NDB_API_TRANS_START_COUNT_SESSION	1
NDB_API_TRANS_COMMIT_COUNT_SESSION	1
NDB_API_TRANS_ABORT_COUNT_SESSION	0
NDB_API_TRANS_CLOSE_COUNT_SESSION	1

NDB API Statistics Counters and Variables

NDB_API_PK_OP_COUNT_SESSION	1
NDB_API_UK_OP_COUNT_SESSION	0
NDB_API_TABLE_SCAN_COUNT_SESSION	0
NDB_API_RANGE_SCAN_COUNT_SESSION	0
NDB_API_PRUNED_SCAN_COUNT_SESSION	0
NDB_API_SCAN_BATCH_COUNT_SESSION	0
NDB_API_READ_ROW_COUNT_SESSION	1
NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION	1
NDB_API_EVENT_DATA_COUNT_INJECTOR	0
NDB_API_EVENT_NONDATA_COUNT_INJECTOR	0
NDB_API_EVENT_BYTES_COUNT_INJECTOR	0
NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE	0
NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE	0
NDB_API_WAIT_META_REQUEST_COUNT_SLAVE	0
NDB_API_WAIT_NANOS_COUNT_SLAVE	0
NDB_API_BYTES_SENT_COUNT_SLAVE	0
NDB_API_BYTES_RECEIVED_COUNT_SLAVE	0
NDB_API_TRANS_START_COUNT_SLAVE	0
NDB_API_TRANS_COMMIT_COUNT_SLAVE	0
NDB_API_TRANS_ABORT_COUNT_SLAVE	0
NDB_API_TRANS_CLOSE_COUNT_SLAVE	0
NDB_API_PK_OP_COUNT_SLAVE	0
NDB_API_UK_OP_COUNT_SLAVE	0
NDB_API_TABLE_SCAN_COUNT_SLAVE	0
NDB_API_RANGE_SCAN_COUNT_SLAVE	0
NDB_API_PRUNED_SCAN_COUNT_SLAVE	0
NDB_API_SCAN_BATCH_COUNT_SLAVE	0
NDB_API_READ_ROW_COUNT_SLAVE	0
NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE	0
NDB_API_WAIT_EXEC_COMPLETE_COUNT	4
NDB_API_WAIT_SCAN_RESULT_COUNT	3
NDB_API_WAIT_META_REQUEST_COUNT	28
NDB_API_WAIT_NANOS_COUNT	53756398
NDB_API_BYTES_SENT_COUNT	1060
NDB_API_BYTES_RECEIVED_COUNT	9724
NDB_API_TRANS_START_COUNT	3
NDB_API_TRANS_COMMIT_COUNT	2
NDB_API_TRANS_ABORT_COUNT	0
NDB_API_TRANS_CLOSE_COUNT	3
NDB_API_PK_OP_COUNT	2
NDB_API_UK_OP_COUNT	0
NDB_API_TABLE_SCAN_COUNT	1
NDB_API_RANGE_SCAN_COUNT	0
NDB_API_PRUNED_SCAN_COUNT	0
NDB_API_SCAN_BATCH_COUNT	0
NDB_API_READ_ROW_COUNT	2
NDB_API_TRANS_LOCAL_READ_ROW_COUNT	2
NDB_API_EVENT_DATA_COUNT	0
NDB_API_EVENT_NONDATA_COUNT	0
NDB_API_EVENT_BYTES_COUNT	0

+-----+-----+
60 rows in set (0.00 sec)

Each `Ndb` object has its own counters. NDB API applications can read the values of the counters for use in optimization or monitoring. For multithreaded clients which use more than one `Ndb` object concurrently, it is also possible to obtain a summed view of counters from all `Ndb` objects belonging to a given `Ndb_cluster_connection`.

Four sets of these counters are exposed. One set applies to the current session only; the other 3 are global. *This is in spite of the fact that their values can be obtained as either session or global status variables in the `mysql` client.* This means that specifying the `SESSION` or `GLOBAL` keyword with `SHOW STATUS` has no effect on the values reported for NDB API statistics status variables, and the value for each of these variables is the same whether the value is obtained from the equivalent column of the `SESSION_STATUS` or the `GLOBAL_STATUS` table.

- *Session counters (session specific)*

Session counters relate to the `Ndb` objects in use by (only) the current session. Use of such objects by other MySQL clients does not influence these counts.

In order to minimize confusion with standard MySQL session variables, we refer to the variables that correspond to these NDB API session counters as “`_session` variables”, with a leading underscore.

- *Slave counters (global)*

This set of counters relates to the `Ndb` objects used by the replication slave SQL thread, if any. If this `mysqld` does not act as a replication slave, or does not use `NDB` tables, then all of these counts are 0.

We refer to the related status variables as “`_slave` variables” (with a leading underscore).

- *Injector counters (global)*

Injector counters relate to the `Ndb` object used to listen to cluster events by the binlog injector thread. Even when not writing a binary log, `mysqld` processes attached to a MySQL Cluster continue to listen for some events, such as schema changes.

We refer to the status variables that correspond to NDB API injector counters as “`_injector` variables” (with a leading underscore).

- *Server (Global) counters (global)*

This set of counters relates to all `Ndb` objects currently used by this `mysqld`. This includes all MySQL client applications, the slave SQL thread (if any), the binlog injector, and the `NDB` utility thread.

We refer to the status variables that correspond to these counters as “global variables” or “`mysqld`-level variables”.

You can obtain values for a particular set of variables by additionally filtering for the substring `session`, `slave`, or `injector` in the variable name (along with the common prefix `Ndb_api`). For `_session` variables, this can be done as shown here:

```
mysql> SHOW STATUS LIKE 'ndb_api%session';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_api_wait_exec_complete_count_session | 2 |
| Ndb_api_wait_scan_result_count_session | 0 |
| Ndb_api_wait_meta_request_count_session | 1 |
| Ndb_api_wait_nanos_count_session | 8144375 |
| Ndb_api_bytes_sent_count_session | 68 |
| Ndb_api_bytes_received_count_session | 84 |
| Ndb_api_trans_start_count_session | 1 |
| Ndb_api_trans_commit_count_session | 1 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 1 |
| Ndb_api_pk_op_count_session | 1 |
| Ndb_api_uk_op_count_session | 0 |
| Ndb_api_table_scan_count_session | 0 |
| Ndb_api_range_scan_count_session | 0 |
| Ndb_api_pruned_scan_count_session | 0 |
| Ndb_api_scan_batch_count_session | 0 |
| Ndb_api_read_row_count_session | 1 |
| Ndb_api_trans_local_read_row_count_session | 1 |
+-----+-----+
18 rows in set (0.50 sec)
```

To obtain a listing of the NDB API `mysqld`-level status variables, filter for variable names beginning with `ndb_api` and ending in `_count`, like this:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS
-> WHERE VARIABLE_NAME LIKE 'ndb_api%count';
```

VARIABLE_NAME	VARIABLE_VALUE
NDB_API_WAIT_EXEC_COMPLETE_COUNT	4
NDB_API_WAIT_SCAN_RESULT_COUNT	3
NDB_API_WAIT_META_REQUEST_COUNT	28
NDB_API_WAIT_NANOS_COUNT	53756398
NDB_API_BYTES_SENT_COUNT	1060
NDB_API_BYTES_RECEIVED_COUNT	9724
NDB_API_TRANS_START_COUNT	3
NDB_API_TRANS_COMMIT_COUNT	2
NDB_API_TRANS_ABORT_COUNT	0
NDB_API_TRANS_CLOSE_COUNT	3
NDB_API_PK_OP_COUNT	2
NDB_API_UK_OP_COUNT	0
NDB_API_TABLE_SCAN_COUNT	1
NDB_API_RANGE_SCAN_COUNT	0
NDB_API_PRUNED_SCAN_COUNT	0
NDB_API_SCAN_BATCH_COUNT	0
NDB_API_READ_ROW_COUNT	2
NDB_API_TRANS_LOCAL_READ_ROW_COUNT	2
NDB_API_EVENT_DATA_COUNT	0
NDB_API_EVENT_NONDATA_COUNT	0
NDB_API_EVENT_BYTES_COUNT	0

```
21 rows in set (0.09 sec)
```

Not all counters are reflected in all 4 sets of status variables. For the event counters `DataEventsRecvdCount`, `NondataEventsRecvdCount`, and `EventBytesRecvdCount`, only `_injector` and `mysqld`-level NDB API status variables are available:

```
mysql> SHOW STATUS LIKE 'ndb_api%event%';
```

Variable_name	Value
Ndb_api_event_data_count_injector	0
Ndb_api_event_nondata_count_injector	0
Ndb_api_event_bytes_count_injector	0
Ndb_api_event_data_count	0
Ndb_api_event_nondata_count	0
Ndb_api_event_bytes_count	0

```
6 rows in set (0.00 sec)
```

`_injector` status variables are not implemented for any other NDB API counters, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb_api%injector%';
```

Variable_name	Value
Ndb_api_event_data_count_injector	0
Ndb_api_event_nondata_count_injector	0
Ndb_api_event_bytes_count_injector	0

```
3 rows in set (0.00 sec)
```

The names of the status variables can easily be associated with the names of the corresponding counters. Each NDB API statistics counter is listed in the following table with a description as well as the names of any MySQL server status variables corresponding to this counter.

Counter Name	<p>Description</p> <p>Status Variables (by statistic type):</p> <ul style="list-style-type: none"> • Session • Slave • Injector • Server
WaitExecCompleteCount	<p>Number of times thread has been blocked while waiting for execution of an operation to complete. Includes all <code>execute()</code> calls as well as implicit executes for blob operations and auto-increment not visible to clients.</p> <ul style="list-style-type: none"> • Ndb_api_wait_exec_complete_count_session [2161] • Ndb_api_wait_exec_complete_count_slave [2161] • [none] • Ndb_api_wait_exec_complete_count [2162]
WaitScanResultCount	<p>Number of times thread has been blocked while waiting for a scan-based signal, such waiting for additional results, or for a scan to close.</p> <ul style="list-style-type: none"> • Ndb_api_wait_scan_result_count_session [2162] • Ndb_api_wait_scan_result_count_slave [2162] • [none] • Ndb_api_wait_scan_result_count [2162]
WaitMetaRequestCount	<p>Number of times thread has been blocked waiting for a metadata-based signal; this can occur when waiting for a DDL operation or for an epoch to be started (or ended).</p> <ul style="list-style-type: none"> • Ndb_api_wait_meta_request_count_session [2162] • Ndb_api_wait_meta_request_count_slave [2162] • [none] • Ndb_api_wait_meta_request_count [2163]
WaitNanosCount	<p>Total time (in nanoseconds) spent waiting for some type of signal from the data nodes.</p> <ul style="list-style-type: none"> • Ndb_api_wait_nanos_count_session [2163] • Ndb_api_wait_nanos_count_slave [2163] • [none] • Ndb_api_wait_nanos_count [2163]
BytesSentCount	<p>Amount of data (in bytes) sent to the data nodes</p> <ul style="list-style-type: none"> • Ndb_api_bytes_sent_count_session [2163]

Counter Name	Description Status Variables (by statistic type): <ul style="list-style-type: none"> • Session • Slave • Injector • Server
	<ul style="list-style-type: none"> • Ndb_api_bytes_sent_count_slave [2164] • [none] • Ndb_api_bytes_sent_count [2164]
BytesRecvdCount	Amount of data (in bytes) received from the data nodes <ul style="list-style-type: none"> • Ndb_api_bytes_received_count_session [2164] • Ndb_api_bytes_received_count_slave [2164] • [none] • Ndb_api_bytes_received_count [2164]
TransStartCount	Number of transactions started. <ul style="list-style-type: none"> • Ndb_api_trans_start_count_session [2164] • Ndb_api_trans_start_count_slave [2165] • [none] • Ndb_api_trans_start_count [2165]
TransCommitCount	Number of transactions committed. <ul style="list-style-type: none"> • Ndb_api_trans_commit_count_session [2165] • Ndb_api_trans_commit_count_slave [2165] • [none] • Ndb_api_trans_commit_count [2165]
TransAbortCount	Number of transactions aborted. <ul style="list-style-type: none"> • Ndb_api_trans_abort_count_session [2165] • Ndb_api_trans_abort_count_slave [2165] • [none] • Ndb_api_trans_abort_count [2166]
TransCloseCount	Number of transactions aborted. (This value may be greater than the sum of TransCommitCount and TransAbortCount .) <ul style="list-style-type: none"> • Ndb_api_trans_close_count_session [2166] • Ndb_api_trans_close_count_slave [2166]

Counter Name	Description
	<ul style="list-style-type: none"> • [none] • Ndb_api_trans_close_count [2166]
PkOpCount	<p>Number of operations based on or using primary keys. This count includes blob-part table operations, implicit unlocking operations, and auto-increment operations, as well as primary key operations normally visible to MySQL clients.</p> <ul style="list-style-type: none"> • Ndb_api_pk_op_count_session [2166] • Ndb_api_pk_op_count_slave [2167] • [none] • Ndb_api_pk_op_count [2167]
UkOpCount	<p>Number of operations based on or using unique keys.</p> <ul style="list-style-type: none"> • Ndb_api_uk_op_count_session [2167] • Ndb_api_uk_op_count_slave [2167] • [none] • Ndb_api_uk_op_count [2167]
TableScanCount	<p>Number of table scans that have been started. This includes scans of internal tables.</p> <ul style="list-style-type: none"> • Ndb_api_table_scan_count_session [2167] • Ndb_api_table_scan_count_slave [2168] • [none] • Ndb_api_table_scan_count [2168]
RangeScanCount	<p>Number of range scans that have been started.</p> <ul style="list-style-type: none"> • Ndb_api_range_scan_count_session [2168] • Ndb_api_range_scan_count_slave [2168] • [none] • Ndb_api_range_scan_count [2168]
PrunedScanCount	<p>Number of scans that have been pruned to a single partition.</p> <ul style="list-style-type: none"> • Ndb_api_pruned_scan_count_session [2168]

Counter Name	<p>Description</p> <p>Status Variables (by statistic type):</p> <ul style="list-style-type: none"> • Session • Slave • Injector • Server
	<ul style="list-style-type: none"> • Ndb_api_pruned_scan_count_slave [2169] • [none] • Ndb_api_pruned_scan_count [2169]
ScanBatchCount	<p>Number of batches of rows received. (A <i>batch</i> in this context is a set of scan results from a single fragment.)</p> <ul style="list-style-type: none"> • Ndb_api_scan_batch_count_session [2169] • Ndb_api_scan_batch_count_slave [2169] • [none] • Ndb_api_scan_batch_count [2169]
ReadRowCount	<p>Total number of rows that have been read. Includes rows read using primary key, unique key, and scan operations.</p> <ul style="list-style-type: none"> • Ndb_api_read_row_count_session [2169] • Ndb_api_read_row_count_slave [2170] • [none] • Ndb_api_read_row_count [2170]
TransLocalReadRowCount	<p>Number of rows read from the data same node on which the transaction was being run.</p> <ul style="list-style-type: none"> • Ndb_api_trans_local_read_row_count_session [2170] • Ndb_api_trans_local_read_row_count_slave [2170] • [none] • Ndb_api_trans_local_read_row_count [2170]
DataEventsRecvdCount	<p>Number of row change events received.</p> <ul style="list-style-type: none"> • [none] • [none] • Ndb_api_event_data_count_injector [2171] • Ndb_api_event_data_count [2171]
NondataEventsRecvdCount	<p>Number of events received, other than row change events.</p> <ul style="list-style-type: none"> • [none]

Counter Name	Description
	<ul style="list-style-type: none"> • [none] • Ndb_api_event_nondata_count_injector [2171] • Ndb_api_event_nondata_count [2171]
<code>EventBytesRecvdCount</code>	Number of bytes of events received. <ul style="list-style-type: none"> • [none] • [none] • Ndb_api_event_bytes_count_injector [2171] • Ndb_api_event_bytes_count [2171]

To see all counts of committed transactions—that is, all `TransCommitCount` counter status variables—you can filter the results of `SHOW STATUS` for the substring `trans_commit_count`, like this:

```
mysql> SHOW STATUS LIKE '%trans_commit_count%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_api_trans_commit_count_session | 1 |
| Ndb_api_trans_commit_count_slave | 0 |
| Ndb_api_trans_commit_count | 2 |
+-----+-----+
3 rows in set (0.00 sec)
```

From this you can determine that 1 transaction has been committed in the current `mysql` client session, and 2 transactions have been committed on this `mysqld` since it was last restarted.

You can see how various NDB API counters are incremented by a given SQL statement by comparing the values of the corresponding `_session` status variables immediately before and after performing the statement. In this example, after getting the initial values from `SHOW STATUS`, we create in the `test` database an NDB table, named `t`, that has a single column:

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_api_wait_exec_complete_count_session | 2 |
| Ndb_api_wait_scan_result_count_session | 0 |
| Ndb_api_wait_meta_request_count_session | 3 |
| Ndb_api_wait_nanos_count_session | 820705 |
| Ndb_api_bytes_sent_count_session | 132 |
| Ndb_api_bytes_received_count_session | 372 |
| Ndb_api_trans_start_count_session | 1 |
| Ndb_api_trans_commit_count_session | 1 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 1 |
+-----+-----+
```

NDB API Statistics Counters and Variables

```
| Ndb_api_pk_op_count_session | 1 |
| Ndb_api_uk_op_count_session | 0 |
| Ndb_api_table_scan_count_session | 0 |
| Ndb_api_range_scan_count_session | 0 |
| Ndb_api_pruned_scan_count_session | 0 |
| Ndb_api_scan_batch_count_session | 0 |
| Ndb_api_read_row_count_session | 1 |
| Ndb_api_trans_local_read_row_count_session | 1 |
+-----+
18 rows in set (0.00 sec)

mysql> USE test;
Database changed
mysql> CREATE TABLE t (c INT) ENGINE NDBCLUSTER;
Query OK, 0 rows affected (0.85 sec)
```

Now you can execute a new `SHOW STATUS` statement and observe the changes, as shown here (with the changed rows highlighted in the output):

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----+
| Variable_name | Value |
+-----+
| Ndb_api_wait_exec_complete_count_session | 8 |
| Ndb_api_wait_scan_result_count_session | 0 |
| Ndb_api_wait_meta_request_count_session | 17 |
| Ndb_api_wait_nanos_count_session | 706871709 |
| Ndb_api_bytes_sent_count_session | 2376 |
| Ndb_api_bytes_received_count_session | 3844 |
| Ndb_api_trans_start_count_session | 4 |
| Ndb_api_trans_commit_count_session | 4 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 4 |
| Ndb_api_pk_op_count_session | 6 |
| Ndb_api_uk_op_count_session | 0 |
| Ndb_api_table_scan_count_session | 0 |
| Ndb_api_range_scan_count_session | 0 |
| Ndb_api_pruned_scan_count_session | 0 |
| Ndb_api_scan_batch_count_session | 0 |
| Ndb_api_read_row_count_session | 2 |
| Ndb_api_trans_local_read_row_count_session | 1 |
+-----+
18 rows in set (0.00 sec)
```

Similarly, you can see the changes in the NDB API statistics counters caused by inserting a row into `t`: Insert the row, then run the same `SHOW STATUS` statement used in the previous example, as shown here:

```
mysql> INSERT INTO t VALUES (100);
Query OK, 1 row affected (0.00 sec)

mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----+
| Variable_name | Value |
+-----+
| Ndb_api_wait_exec_complete_count_session | 11 |
| Ndb_api_wait_scan_result_count_session | 6 |
| Ndb_api_wait_meta_request_count_session | 20 |
| Ndb_api_wait_nanos_count_session | 707370418 |
| Ndb_api_bytes_sent_count_session | 2724 |
| Ndb_api_bytes_received_count_session | 4116 |
| Ndb_api_trans_start_count_session | 7 |
| Ndb_api_trans_commit_count_session | 6 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 7 |
| Ndb_api_pk_op_count_session | 8 |
| Ndb_api_uk_op_count_session | 0 |
```

```

/ Ndb_api_table_scan_count_session          / 1          /
| Ndb_api_range_scan_count_session         | 0          |
| Ndb_api_pruned_scan_count_session       | 0          |
| Ndb_api_scan_batch_count_session        | 0          |
/ Ndb_api_read_row_count_session           / 3          /
/ Ndb_api_trans_local_read_row_count_session / 2          /
+-----+-----+-----+
18 rows in set (0.00 sec)

```

We can make a number of observations from these results:

- Although we created `t` with no explicit primary key, 5 primary key operations were performed in doing so (the difference in the “before” and “after” values of `Ndb_api_pk_op_count_session` [2166], or 6 minus 1). This reflects the creation of the hidden primary key that is a feature of all tables using the `NDB` storage engine.
- By comparing successive values for `Ndb_api_wait_nanos_count_session` [2163], we can see that the `NDB` API operations implementing the `CREATE TABLE` statement waited much longer ($706871709 - 820705 = 706051004$ nanoseconds, or approximately 0.7 second) for responses from the data nodes than those executed by the `INSERT` ($707370418 - 706871709 = 498709$ ns or roughly .0005 second). The execution times reported for these statements in the `mysql` client correlate roughly with these figures.

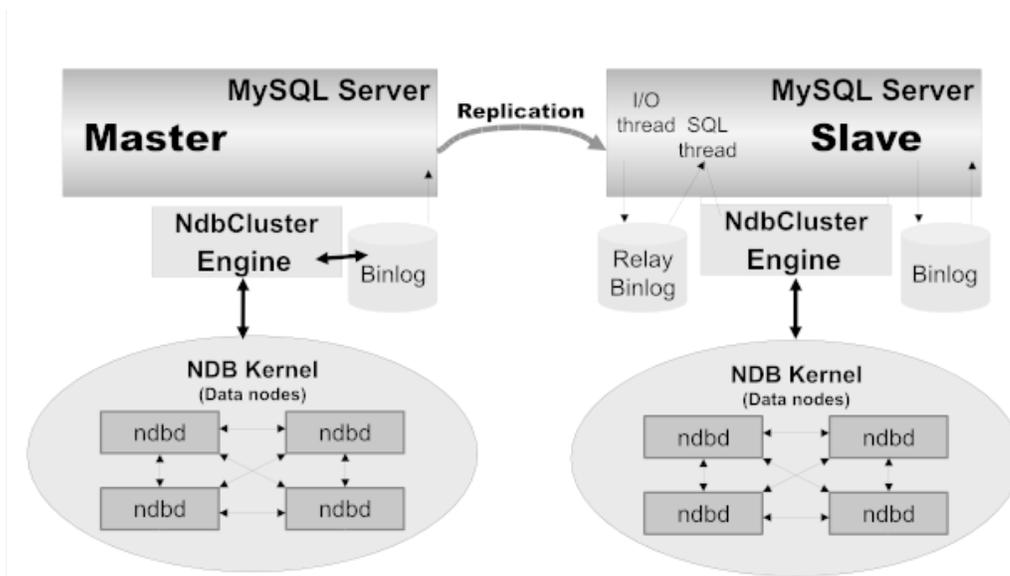
On platforms with without sufficient (nanosecond) time resolution, small changes in the value of the `WaitNanosCount` `NDB` API counter due to SQL statements that execute very quickly may not always be visible in the values of `Ndb_api_wait_nanos_count_session` [2163], `Ndb_api_wait_nanos_count_slave` [2163], or `Ndb_api_wait_nanos_count` [2163].

- The `INSERT` statement incremented both the `ReadRowCount` and `TransLocalReadRowCount` `NDB` API statistics counters, as reflected by the increased values of `Ndb_api_read_row_count_session` [2169] and `Ndb_api_trans_local_read_row_count_session` [2170].

17.6. MySQL Cluster Replication

MySQL Cluster supports *asynchronous replication*, more usually referred to simply as “replication”. This section explains how to set up and manage a configuration in which one group of computers operating as a MySQL Cluster replicates to a second computer or group of computers. We assume some familiarity on the part of the reader with standard MySQL replication as discussed elsewhere in this Manual. (See [Chapter 16, Replication](#)).

Normal (non-clustered) replication involves a “master” server and a “slave” server, the master being the source of the operations and data to be replicated and the slave being the recipient of these. In MySQL Cluster, replication is conceptually very similar but can be more complex in practice, as it may be extended to cover a number of different configurations including replicating between two complete clusters. Although a MySQL Cluster itself depends on the `NDB` storage engine for clustering functionality, it is not necessary to use `NDB` as the storage engine for the slave's copies of the replicated tables (see [Replication from `NDB` to non-`NDB` tables](#)). However, for maximum availability, it is possible (and preferable) to replicate from one MySQL Cluster to another, and it is this scenario that we discuss, as shown in the following figure:



In this scenario, the replication process is one in which successive states of a master cluster are logged and saved to a slave cluster. This process is accomplished by a special thread known as the NDB binlog injector thread, which runs on each MySQL server and produces a binary log (*binlog*). This thread ensures that all changes in the cluster producing the binary log—and not just those changes that are effected through the MySQL Server—are inserted into the binary log with the correct serialization order. We refer to the MySQL replication master and replication slave servers as replication servers or replication nodes, and the data flow or line of communication between them as a *replication channel*.

For information about performing point-in-time recovery with MySQL Cluster and MySQL Cluster Replication, see [Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”](#).

NDB API *_slave* status variables. NDB API counters can provide enhanced monitoring capabilities on MySQL Cluster replication slaves. These are implemented as NDB statistics *_slave* status variables, as seen in the output of `SHOW STATUS`, or in the results of queries against the `SESSION_STATUS` or `GLOBAL_STATUS` table in a `mysql` client session connected to a MySQL Server that is acting as a slave in MySQL Cluster Replication. By comparing the values of these status variables before and after the execution of statements affecting replicated NDB tables, you can observe the corresponding actions taken on the NDB API level by the slave, which can be useful when monitoring or troubleshooting MySQL Cluster Replication. [Section 17.5.15, “NDB API Statistics Counters and Variables”](#), provides additional information.

Replication from NDB to non-NDB tables. It is possible to replicate NDB tables from a MySQL Cluster acting as the master to tables using other MySQL storage engines such as `InnoDB` or `MyISAM` on a slave `mysqld`. However, because of differences between the version of `mysqld` provided with MySQL Cluster and that included with MySQL Server 5.5, the slave server must also use a `mysqld` binary from the MySQL Cluster distribution. See [Section 17.6.2, “General Requirements for MySQL Cluster Replication”](#).

17.6.1. MySQL Cluster Replication: Abbreviations and Symbols

Throughout this section, we use the following abbreviations or symbols for referring to the master and slave clusters, and to processes and commands run on the clusters or cluster nodes:

Symbol or Abbreviation	Description (Refers to...)
<i>M</i>	The cluster serving as the (primary) replication master

Symbol or Abbreviation	Description (Refers to...)
<i>S</i>	The cluster acting as the (primary) replication slave
<code>shellM></code>	Shell command to be issued on the master cluster
<code>mysqlM></code>	MySQL client command issued on a single MySQL server running as an SQL node on the master cluster
<code>mysqlM*></code>	MySQL client command to be issued on all SQL nodes participating in the replication master cluster
<code>shellS></code>	Shell command to be issued on the slave cluster
<code>mysqlS></code>	MySQL client command issued on a single MySQL server running as an SQL node on the slave cluster
<code>mysqlS*></code>	MySQL client command to be issued on all SQL nodes participating in the replication slave cluster
<i>C</i>	Primary replication channel
<i>C'</i>	Secondary replication channel
<i>M'</i>	Secondary replication master
<i>S'</i>	Secondary replication slave

17.6.2. General Requirements for MySQL Cluster Replication

A replication channel requires two MySQL servers acting as replication servers (one each for the master and slave). For example, this means that in the case of a replication setup with two replication channels (to provide an extra channel for redundancy), there will be a total of four replication nodes, two per cluster.

Replication of a MySQL Cluster as described in this section and those following is dependent on row-based replication. This means that the replication master MySQL server must be started with `--binlog-format=ROW` [436], as described in [Section 17.6.6, “Starting MySQL Cluster Replication \(Single Replication Channel\)”](#). For general information about row-based replication, see [Section 16.1.2, “Replication Formats”](#).



Important

If you attempt to use MySQL Cluster Replication with `--binlog-format=STATEMENT` [436], replication fails to work properly because the `ndb_binlog_index` table on the master and the `epoch` column of the `ndb_apply_status` table on the slave are not updated (see [Section 17.6.4, “MySQL Cluster Replication Schema and Tables”](#)). Instead, only updates on the MySQL server acting as the replication master propagate to the slave, and no updates from any other SQL nodes on the master cluster are replicated.

In MySQL Cluster NDB 7.2.1 and later, the default value for the `--binlog-format` [436] option is `STATEMENT` (for MySQL Cluster releases based on MySQL Server 5.1, the default was `MIXED`). You must change the binary logging format to `ROW` manually on all MySQL Servers on the master MySQL Cluster, prior to starting MySQL Cluster replication. As just described previously, you can do this on startup using the `--binlog-format` [436] option. You can also change the binary logging format at runtime by setting the global `binlog_format` [1899] system variable. The latter method represents a change from MySQL Cluster NDB 7.1 and earlier versions of MySQL Cluster, where MySQL Servers used as MySQL Cluster SQL nodes switched their logging format automatically when the cluster acted as a replication master.

Each MySQL server used for replication in either cluster must be uniquely identified among all the MySQL replication servers participating in either cluster (you cannot have replication servers on both the master and slave clusters sharing the same ID). This can be done by starting each SQL node using the `--server-id=id` option, where `id` is a unique integer. Although it is not strictly necessary, we will assume for purposes of this discussion that all MySQL Cluster binaries are of the same release version.

It is generally true in MySQL Replication that both MySQL servers (`mysqld` processes) involved must be compatible with one another with respect to both the version of the replication protocol used and the SQL feature sets which they support (see [Section 16.4.2, “Replication Compatibility Between MySQL Versions”](#)). It is due to such differences between the binaries in the MySQL Cluster and MySQL Server 5.5 distributions that MySQL Cluster Replication has the additional requirement that both `mysqld` binaries come from a MySQL Cluster distribution. The simplest and easiest way to assure that the `mysqld` servers are compatible is to use the same MySQL Cluster distribution for all master and slave `mysqld` binaries.

We assume that the slave server or cluster is dedicated to replication of the master, and that no other data is being stored on it.



Note

It is possible to replicate a MySQL Cluster using statement-based replication. However, in this case, the following restrictions apply:

- All updates to data rows on the cluster acting as the master must be directed to a single MySQL server.
- It is not possible to replicate a cluster using multiple simultaneous MySQL replication processes.
- Only changes made at the SQL level are replicated.

These are in addition to the other limitations of statement-based replication as opposed to row-based replication; see [Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”](#), for more specific information concerning the differences between the two replication formats.

17.6.3. Known Issues in MySQL Cluster Replication

This section discusses known problems or issues when using replication with MySQL Cluster NDB 7.2.

Loss of master-slave connection. A loss of connection can occur either between the replication master SQL node and the replication slave SQL node, or between the replication master SQL node and the data nodes in the master cluster. In the latter case, this can occur not only as a result of loss of physical connection (for example, a broken network cable), but due to the overflow of data node event buffers; if the SQL node is too slow to respond, it may be dropped by the cluster (this is controllable to some degree by adjusting the `MaxBufferedEpochs` [2066] and `TimeBetweenEpochs` [2065] configuration parameters). If this occurs, *it is entirely possible for new data to be inserted into the master cluster without being recorded in the replication master's binary log*. For this reason, to guarantee high availability, it is extremely important to maintain a backup replication channel, to monitor the primary channel, and to fail over to the secondary replication channel when necessary to keep the slave cluster synchronized with the master. MySQL Cluster is not designed to perform such monitoring on its own; for this, an external application is required.

The replication master issues a “gap” event when connecting or reconnecting to the master cluster. (A gap event is a type of “incident event,” which indicates an incident that occurs that affects the contents of the database but that cannot easily be represented as a set of changes. Examples of incidents are

server crashes, database resynchronization, (some) software updates, and (some) hardware changes.) When the slave encounters a gap in the replication log, it stops with an error message. This message is available in the output of `SHOW SLAVE STATUS`, and indicates that the SQL thread has stopped due to an incident registered in the replication stream, and that manual intervention is required. See [Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”](#), for more information about what to do in such circumstances.



Important

Because MySQL Cluster is not designed on its own to monitor replication status or provide failover, if high availability is a requirement for the slave server or cluster, then you must set up multiple replication lines, monitor the master `mysqld` on the primary replication line, and be prepared fail over to a secondary line if and as necessary. This must be done manually, or possibly by means of a third-party application. For information about implementing this type of setup, see [Section 17.6.7, “Using Two Replication Channels for MySQL Cluster Replication”](#), and [Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”](#).

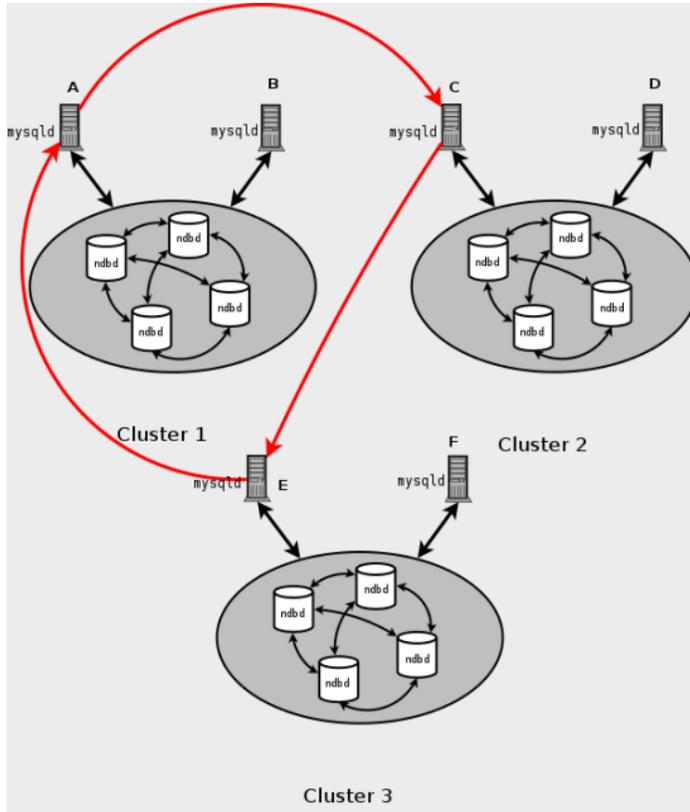
However, if you are replicating from a standalone MySQL server to a MySQL Cluster, one channel is usually sufficient.

Circular replication. MySQL Cluster Replication supports circular replication, as shown in the next example. The replication setup involves three MySQL Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication master for Cluster 2, Cluster 2 acts as the master for Cluster 3, and Cluster 3 acts as the master for Cluster 1, thus completing the circle. Each MySQL Cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as the following conditions are met:

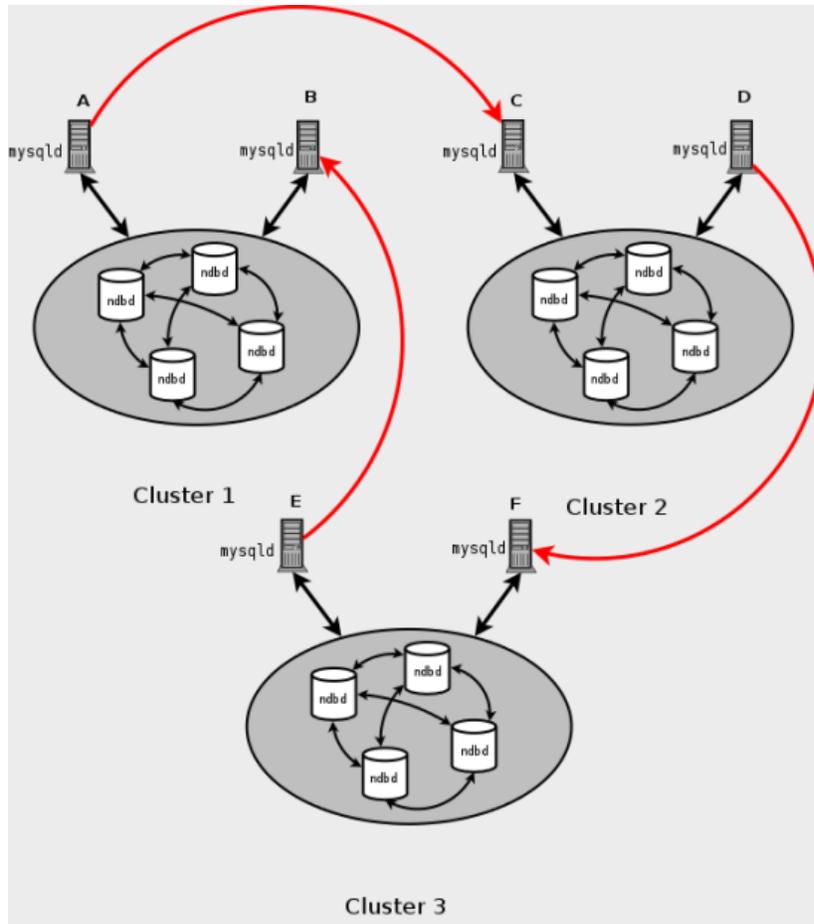
- The SQL nodes on all masters and slaves are the same
- All SQL nodes acting as replication masters and slaves are started using the `--log-slave-updates [1870]` option

This type of circular replication setup is shown in the following diagram:



In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line (indicated by the red arrows in the diagram) directly connects all SQL nodes used as replication masters and slaves.

It should also be possible to set up circular replication in which not all master SQL nodes are also slaves, as shown here:



In this case, different SQL nodes in each cluster are used as replication masters and slaves. However, you must *not* start any of the SQL nodes using `--log-slave-updates` [1870]. This type of circular replication scheme for MySQL Cluster, in which the line of replication (again indicated by the red arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.



Note

The NDB storage engine uses *idempotent execution mode*, which suppresses duplicate-key and other errors that otherwise break circular replication of MySQL Cluster. This is equivalent to setting the global `slave_exec_mode` [1887] system variable to `IDEMPOTENT`. This is also required for multi-master replication when using MySQL Cluster. (Bug #31609)

It is not necessary to set `slave_exec_mode` [1887] in MySQL Cluster replication; MySQL Cluster does this automatically for all NDB tables and ignores any attempts to set this variable explicitly.

MySQL Cluster replication and primary keys. In the event of a node failure, errors in replication of NDB tables without primary keys can still occur, due to the possibility of duplicate rows being inserted in such cases. For this reason, it is highly recommended that all NDB tables being replicated have primary keys.

MySQL Cluster Replication and Unique Keys. In older versions of MySQL Cluster, operations that updated values of unique key columns of NDB tables could result in duplicate-key errors when replicated.

This issue is solved for replication between [NDB](#) tables by deferring unique key checks until after all table row updates have been performed.

Deferring constraints in this way is currently supported only by [NDB](#). Thus, updates of unique keys when replicating from [NDB](#) to a different storage engine such as [MyISAM](#) or [InnoDB](#) are still not supported.

The problem encountered when replicating without deferred checking of unique key updates can be illustrated using [NDB](#) table such as `t`, is created and populated on the master (and replicated to a slave that does not support deferred unique key updates) as shown here:

```
CREATE TABLE t (
  p INT PRIMARY KEY,
  c INT,
  UNIQUE KEY u (c)
) ENGINE NDB;

INSERT INTO t
VALUES (1,1), (2,2), (3,3), (4,4), (5,5);
```

The following `UPDATE` statement on `t` succeeded on the master, since the rows affected are processed in the order determined by the `ORDER BY` option, performed over the entire table:

```
UPDATE t SET c = c - 1 ORDER BY p;
```

However, the same statement failed with a duplicate key error or other constraint violation on the slave, because the ordering of the row updates was done for one partition at a time, rather than for the table as a whole.



Note

Every [NDB](#) table is implicitly partitioned by key when it is created. See [Section 18.2.5, “KEY Partitioning”](#), for more information.

Restarting with `--initial`. Restarting the cluster with the `--initial` [2178] option causes the sequence of GCI and epoch numbers to start over from 0. (This is generally true of MySQL Cluster and not limited to replication scenarios involving Cluster.) The MySQL servers involved in replication should in this case be restarted. After this, you should use the `RESET MASTER` and `RESET SLAVE` statements to clear the invalid `ndb_binlog_index` and `ndb_apply_status` tables, respectively.

Replication from [NDB](#) to other storage engines. It is possible to replicate an [NDB](#) table on the master to a table using a different storage engine on the slave, taking into account the restrictions listed here:

- Multi-master and circular replication are not supported (tables on both the master and the slave must use the [NDB](#) storage engine for this to work).
- Using a storage engine does not perform binary logging for slave tables requires special handling.
- Use of a non-transactional storage engine for slave tables also requires special handling.

The next few paragraphs provide additional information about each of the issues just described.

Multiple masters not supported when replicating [NDB](#) to other storage engines. For replication from [NDB](#) to a different storage engine, the relationship between the two databases must be a simple master-slave one. This means that circular or master-master replication is not supported between MySQL Cluster and other storage engines.

In addition, it is not possible to configure more than one replication channel when replicating between [NDB](#) and a different storage engine. (However, a MySQL Cluster database *can* simultaneously replicate to multiple slave MySQL Cluster databases.) If the master uses [NDB](#) tables, it is still possible to have more

than one MySQL Server maintain a binary log of all changes; however, for the slave to change masters (fail over), the new master-slave relationship must be explicitly defined on the slave.

Replicating NDB to a slave storage engine that does not perform binary logging. If you attempt to replicate from a MySQL Cluster to a slave that uses a storage engine that does not handle its own binary logging, the replication process aborts with the error `Binary logging not possible ... Statement cannot be written atomically since more than one engine involved and at least one engine is self-logging` (Error 1595). It is possible to work around this issue in one of the following ways:

- **Turn off binary logging on the slave.** This can be accomplished by setting `sql_log_bin = 0` [555].
- **Change the storage engine used for the `mysql.ndb_apply_status` table.** Causing this table to use an engine that does not handle its own binary logging can also eliminate the conflict. This can be done by issuing a statement such as `ALTER TABLE mysql.ndb_apply_status ENGINE=MyISAM` on the slave. It is safe to do this when using a non-NDB storage engine on the slave, since you do not then need to worry about keeping multiple slave SQL nodes synchronized.
- **Filter out changes to the `mysql.ndb_apply_status` table on the slave.** This can be done by starting the slave SQL node with `--replicate-ignore-table=mysql.ndb_apply_status` [1878]. If you need for other tables to be ignored by replication, you might wish to use an appropriate `--replicate-wild-ignore-table` [1880] option instead.



Important

You should *not* disable replication or binary logging of `mysql.ndb_apply_status` or change the storage engine used for this table when replicating from one MySQL Cluster to another. See [Replication and binary log filtering rules with replication between MySQL Clusters](#), for details.

Replication from NDB to a nontransactional storage engine. When replicating from NDB to a nontransactional storage engine such as MyISAM, you may encounter unnecessary duplicate key errors when replicating `INSERT ... ON DUPLICATE KEY UPDATE` statements. You can suppress these by using `--ndb-log-update-as-write=0` [2373], which forces updates to be logged as writes (rather than as updates).

In addition, when replicating from NDB to a storage engine that does not implement transactions, if the slave fails to apply any row changes from a given transaction, it does not roll back the rest of the transaction. (This is true when replicating tables using any transactional storage engine—not only NDB—to a nontransactional storage engine.) Because of this, it cannot be guaranteed that transactional consistency will be maintained on the slave in such cases.

Replication and binary log filtering rules with replication between MySQL Clusters. If you are using any of the options `--replicate-do-*`, `--replicate-ignore-*`, `--binlog-do-db` [1894], or `--binlog-ignore-db` [1896] to filter databases or tables being replicated, care must be taken not to block replication or binary logging of the `mysql.ndb_apply_status`, which is required for replication between MySQL Clusters to operate properly. In particular, you must keep in mind the following:

1. Using `--replicate-do-db=db_name` [1875] (and no other `--replicate-do-*` or `--replicate-ignore-*` options) means that *only* tables in database `db_name` are replicated. In this case, you should also use `--replicate-do-db=mysql` [1875], `--binlog-do-db=mysql` [1894], or `--replicate-do-table=mysql.ndb_apply_status` [1878] to ensure that `mysql.ndb_apply_status` is populated on slaves.

Using `--binlog-do-db=db_name` [1894] (and no other `--binlog-do-db` [1894] options) means that changes *only* to tables in database `db_name` are written to the binary log. In this case, you should

also use `--replicate-do-db=mysql` [1875], `--binlog-do-db=mysql` [1894], or `--replicate-do-table=mysql.ndb_apply_status` [1878] to ensure that `mysql.ndb_apply_status` is populated on slaves.

- Using `--replicate-ignore-db=mysql` [1877] means that no tables in the `mysql` database are replicated. In this case, you should also use `--replicate-do-table=mysql.ndb_apply_status` [1878] to ensure that `mysql.ndb_apply_status` is replicated.

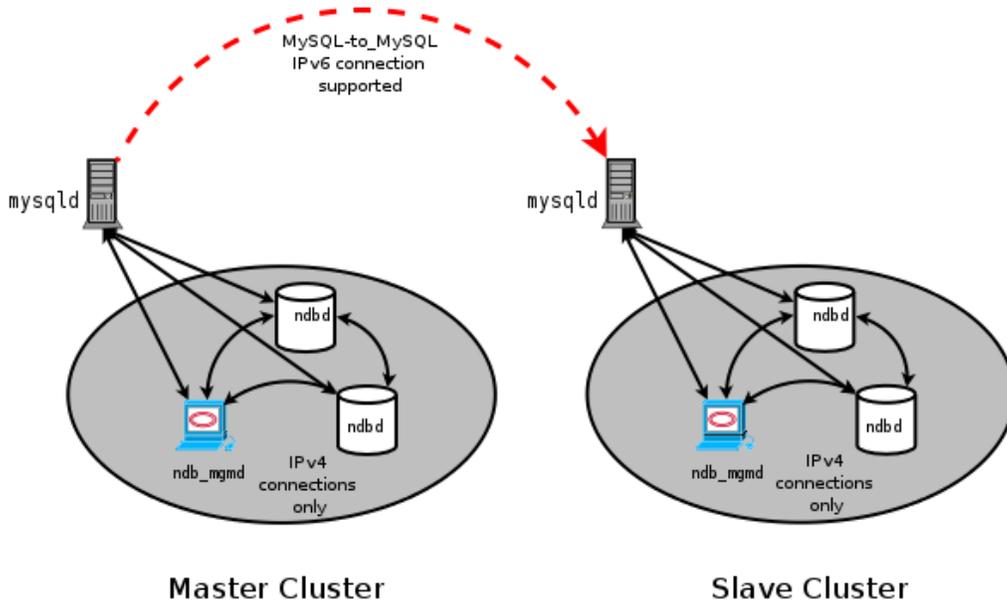
Using `--binlog-ignore-db=mysql` [1896] means that no changes to tables in the `mysql` database are written to the binary log. In this case, you should also use `--replicate-do-table=mysql.ndb_apply_status` [1878] to ensure that `mysql.ndb_apply_status` is replicated.

You should also remember that each replication rule requires the following:

- Its own `--replicate-do-*` or `--replicate-ignore-*` option, and that multiple rules cannot be expressed in a single replication filtering option. For information about these rules, see [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).
- Its own `--binlog-do-db` [1894] or `--binlog-ignore-db` [1896] option, and that multiple rules cannot be expressed in a single binary log filtering option. For information about these rules, see [Section 5.2.4, “The Binary Log”](#).

If you are replicating a MySQL Cluster to a slave that uses a storage engine other than `NDB`, the considerations just given previously may not apply, as discussed elsewhere in this section.

MySQL Cluster Replication and IPv6. Currently, the `NDB` API and `MGM` API do not support IPv6. However, MySQL Servers—including those acting as SQL nodes in a MySQL Cluster—can use IPv6 to contact other MySQL Servers. This means that you can replicate between MySQL Clusters using IPv6 to connect the master and slave SQL nodes as shown by the dotted arrow in the following diagram:



However, all connections originating *within* the MySQL Cluster—represented in the preceding diagram by solid arrows—must use IPv4. In other words, all MySQL Cluster data nodes, management servers, and management clients must be accessible from one another using IPv4. In addition, SQL nodes must use IPv4 to communicate with the cluster.

Since there is currently no support in the NDB and MGM APIs for IPv6, any applications written using these APIs must also make all connections using IPv4.

Attribute promotion and demotion. MySQL Cluster Replication includes support for attribute promotion and demotion. The implementation of the latter distinguishes between lossy and non-lossy type conversions, and their use on the slave can be controlled by setting the `slave_type_conversions` [1889] global server system variable.

For more information about attribute promotion and demotion in MySQL Cluster, see [Row-based replication: attribute promotion and demotion](#).

17.6.4. MySQL Cluster Replication Schema and Tables

Replication in MySQL Cluster makes use of a number of dedicated tables in the `mysql` database on each MySQL Server instance acting as an SQL node in both the cluster being replicated and the replication slave (whether the slave is a single server or a cluster). These tables are created during the MySQL installation process by the `mysql_install_db` script, and include a table for storing the binary log's indexing data. Since the `ndb_binlog_index` table is local to each MySQL server and does not participate in clustering, it uses the `MyISAM` storage engine. This means that it must be created separately on each `mysqld` participating in the master cluster. (However, the binary log itself contains updates from all MySQL servers in the cluster to be replicated.) This table is defined as follows:

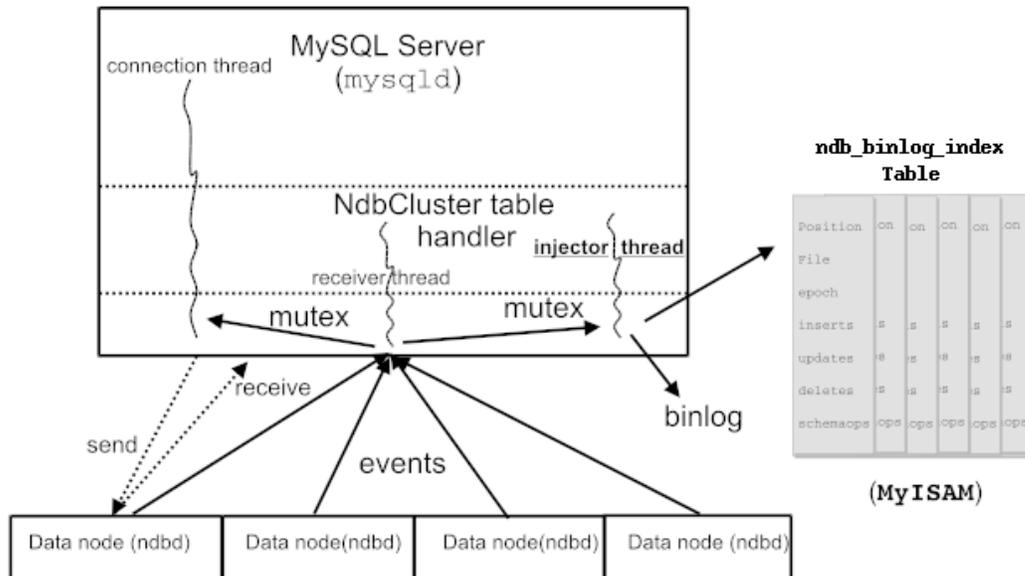
```
CREATE TABLE `ndb_binlog_index` (  
  `Position` BIGINT(20) UNSIGNED NOT NULL,  
  `File` VARCHAR(255) NOT NULL,  
  `epoch` BIGINT(20) UNSIGNED NOT NULL,  
  `inserts` INT(10) UNSIGNED NOT NULL,  
  `updates` INT(10) UNSIGNED NOT NULL,  
  `deletes` INT(10) UNSIGNED NOT NULL,  
  `schemaops` INT(10) UNSIGNED NOT NULL,  
  `orig_server_id` INT(10) UNSIGNED NOT NULL,  
  `orig_epoch` BIGINT(20) UNSIGNED NOT NULL,  
  `gci` INT(10) UNSIGNED NOT NULL,  
  `next_position` bigint(20) unsigned NOT NULL,  
  `next_file` varchar(255) NOT NULL,  
  PRIMARY KEY (`epoch`,`orig_server_id`,`orig_epoch`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

The `next_position` and `next_file` columns were added in MySQL Cluster NDB 7.2.6; see [Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”](#), for more information about using these columns.

When `mysqld` is started with the `--ndb-log-orig` option, the `orig_server_id` and `orig_epoch` columns store, respectively, the ID of the server on which the event originated and the epoch in which the event took place on the originating server.

The following figure shows the relationship of the MySQL Cluster replication master server, its binlog injector thread, and the `mysql.ndb_binlog_index` table.

MySQL Replication Between Clusters, Injecting into Binlog



An additional table, named `ndb_apply_status`, is used to keep a record of the operations that have been replicated from the master to the slave. Unlike the case with `ndb_binlog_index`, the data in this table is not specific to any one SQL node in the (slave) cluster, and so `ndb_apply_status` can use the `NDBCLUSTER` storage engine, as shown here:

```
CREATE TABLE `ndb_apply_status` (
  `server_id` INT(10) UNSIGNED NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  `log_name` VARCHAR(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
  `start_pos` BIGINT(20) UNSIGNED NOT NULL,
  `end_pos` BIGINT(20) UNSIGNED NOT NULL,
  PRIMARY KEY (`server_id`) USING HASH
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;
```

This table is populated only on slaves; on the master, no `DataMemory [2043]` is allocated to it. Since this table is populated from data originating on the master, it should be allowed to replicate; any replication filtering or binary log filtering rules that inadvertently prevent the slave from updating `ndb_apply_status` or the master from writing into the binary log may prevent replication between clusters from operating properly. For more information about potential problems arising from such filtering rules, see [Replication and binary log filtering rules with replication between MySQL Clusters](#).

The `ndb_binlog_index` and `ndb_apply_status` tables are created in the `mysql` database because they should not be explicitly replicated by the user. User intervention is normally not required to create or maintain either of these tables, since both `ndb_binlog_index` and the `ndb_apply_status` are maintained by the `NDB` binary log (binlog) injector thread. This keeps the master `mysqld` process updated to changes performed by the `NDB` storage engine. The `NDB binlog injector thread` receives events directly from the `NDB` storage engine. The `NDB` injector is responsible for capturing all the data events within the cluster, and ensures that all events which change, insert, or delete data are recorded in the `ndb_binlog_index` table. The slave I/O thread transfers the events from the master's binary log to the slave's relay log.

However, it is advisable to check for the existence and integrity of these tables as an initial step in preparing a MySQL Cluster for replication. It is possible to view event data recorded in the binary log by querying the `mysql.ndb_binlog_index` table directly on the master. This can be also be accomplished using the `SHOW BINLOG EVENTS` statement on either the replication master or slave MySQL servers. (See [Section 13.7.5.3, "SHOW BINLOG EVENTS Syntax"](#)).

You can also obtain useful information from the output of `SHOW ENGINE NDB STATUS`.

The `ndb_schema` table is used to track schema changes made to `NDB` tables. It is defined as shown here:

```
CREATE TABLE ndb_schema (
  `db` VARBINARY(63) NOT NULL,
  `name` VARBINARY(63) NOT NULL,
  `slock` BINARY(32) NOT NULL,
  `query` BLOB NOT NULL,
  `node_id` INT UNSIGNED NOT NULL,
  `epoch` BIGINT UNSIGNED NOT NULL,
  `id` INT UNSIGNED NOT NULL,
  `version` INT UNSIGNED NOT NULL,
  `type` INT UNSIGNED NOT NULL,
  PRIMARY KEY USING HASH (db,name)
) ENGINE=NDB DEFAULT CHARSET=latin1;
```

Unlike the two tables previously mentioned in this section, the `ndb_schema` table is not visible either to MySQL `SHOW` statements, or in any `INFORMATION_SCHEMA` tables; however, it can be seen in the output of `ndb_show_tables`, as shown here:

```
shell> ndb_show_tables -t 2
id   type           state   logging database  schema  name
4    UserTable      Online Yes      mysql     def      ndb_apply_status
5    UserTable      Online Yes      ndbworld def      City
6    UserTable      Online Yes      ndbworld def      Country
3    UserTable      Online Yes      mysql     def      NDB$BLOB_2_3
7    UserTable      Online Yes      ndbworld def      CountryLanguage
2    UserTable      Online Yes      mysql     def      ndb_schema

NDBT_ProgramExit: 0 - OK
```

It is also possible to `SELECT` from this table in `mysql` and other MySQL client applications, as shown here:

```
mysql> SELECT * FROM mysql.ndb_schema WHERE name='City' \G
***** 1. row *****
      db: ndbworld
      name: City
      slock:
      query: alter table City engine=ndb
node_id: 4
  epoch: 0
     id: 0
version: 0
   type: 7
1 row in set (0.00 sec)
```

This can sometimes be useful when debugging applications.



Note

When performing schema changes on `NDB` tables, applications should wait until the `ALTER TABLE` statement has returned in the MySQL client connection that issued the statement before attempting to use the updated definition of the table.

If the `ndb_apply_status` table or the `ndb_schema` table does not exist on the slave, `ndb_restore` re-creates the missing table or tables (Bug #14612).

Conflict resolution for MySQL Cluster Replication requires the presence of an additional `mysql.ndb_replication` table. Currently, this table must be created manually. For information about how to do this, see [Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”](#).

17.6.5. Preparing the MySQL Cluster for Replication

Preparing the MySQL Cluster for replication consists of the following steps:

1. Check all MySQL servers for version compatibility (see [Section 17.6.2, “General Requirements for MySQL Cluster Replication”](#)).
2. Create a slave account on the master Cluster with the appropriate privileges:

```
mysqlM> GRANT REPLICATION SLAVE
-> ON *.* TO 'slave_user'@'slave_host'
-> IDENTIFIED BY 'slave_password';
```

In the previous statement, `slave_user` is the slave account user name, `slave_host` is the host name or IP address of the replication slave, and `slave_password` is the password to assign to this account.

For example, to create a slave user account with the name “myslave,” logging in from the host named “rep-slave,” and using the password “53cr37,” use the following `GRANT` statement:

```
mysqlM> GRANT REPLICATION SLAVE
-> ON *.* TO 'myslave'@'rep-slave'
-> IDENTIFIED BY '53cr37';
```

For security reasons, it is preferable to use a unique user account—not employed for any other purpose—for the replication slave account.

3. Configure the slave to use the master. Using the MySQL Monitor, this can be accomplished with the `CHANGE MASTER TO` statement:

```
mysqlS> CHANGE MASTER TO
-> MASTER_HOST='master_host',
-> MASTER_PORT=master_port,
-> MASTER_USER='slave_user',
-> MASTER_PASSWORD='slave_password';
```

In the previous statement, `master_host` is the host name or IP address of the replication master, `master_port` is the port for the slave to use for connecting to the master, `slave_user` is the user name set up for the slave on the master, and `slave_password` is the password set for that user account in the previous step.

For example, to tell the slave to replicate from the MySQL server whose host name is “rep-master,” using the replication slave account created in the previous step, use the following statement:

```
mysqlS> CHANGE MASTER TO
-> MASTER_HOST='rep-master',
-> MASTER_PORT=3306,
-> MASTER_USER='myslave',
-> MASTER_PASSWORD='53cr37';
```

For a complete list of options that can be used with this statement, see [Section 13.4.2.1, “CHANGE MASTER TO Syntax”](#).

To provide replication backup capability, you also need to add an `ndb-connectstring` option to the slave's `my.cnf` file prior to starting the replication process. See [Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”](#), for details.

For additional options that can be set in `my.cnf` for replication slaves, see [Section 16.1.3, “Replication and Binary Logging Options and Variables”](#).

- If the master cluster is already in use, you can create a backup of the master and load this onto the slave to cut down on the amount of time required for the slave to synchronize itself with the master. If the slave is also running MySQL Cluster, this can be accomplished using the backup and restore procedure described in [Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”](#).

```
ndb-connectstring=management_host[:port]
```

In the event that you are *not* using MySQL Cluster on the replication slave, you can create a backup with this command on the replication master:

```
shellM> mysqldump --master-data=1
```

Then import the resulting data dump onto the slave by copying the dump file over to the slave. After this, you can use the `mysql` client to import the data from the dumpfile into the slave database as shown here, where `dump_file` is the name of the file that was generated using `mysqldump` on the master, and `db_name` is the name of the database to be replicated:

```
shellS> mysql -u root -p db_name < dump_file
```

For a complete list of options to use with `mysqldump`, see [Section 4.5.4, “mysqldump — A Database Backup Program”](#).



Note

If you copy the data to the slave in this fashion, you should make sure that the slave is started with the `--skip-slave-start` option on the command line, or else include `skip-slave-start` in the slave's `my.cnf` file to keep it from trying to connect to the master to begin replicating before all the data has been loaded. Once the data loading has completed, follow the additional steps outlined in the next two sections.

- Ensure that each MySQL server acting as a replication master is configured with a unique server ID, and with binary logging enabled, using the row format. (See [Section 16.1.2, “Replication Formats”](#).) These options can be set either in the master server's `my.cnf` file, or on the command line when starting the master `mysqld` process. See [Section 17.6.6, “Starting MySQL Cluster Replication \(Single Replication Channel\)”](#), for information regarding the latter option.

17.6.6. Starting MySQL Cluster Replication (Single Replication Channel)

This section outlines the procedure for starting MySQL Cluster replication using a single replication channel.

- Start the MySQL replication master server by issuing this command:

```
shellM> mysqld --ndbcluster --server-id=id \
--log-bin --binlog-format=ROW &
```

In the previous statement, `id` is this server's unique ID (see [Section 17.6.2, “General Requirements for MySQL Cluster Replication”](#)). This starts the server's `mysqld` process with binary logging enabled using the proper logging format.

- Start the MySQL replication slave server as shown here:

```
shellS> mysqld --ndbcluster --server-id=id &
```

In the command just shown, *id* is the slave server's unique ID. It is not necessary to enable logging on the replication slave.



Note

You should use the `--skip-slave-start` option with this command or else you should include `skip-slave-start` in the slave server's `my.cnf` file, unless you want replication to begin immediately. With the use of this option, the start of replication is delayed until the appropriate `START SLAVE` statement has been issued, as explained in Step 4 below.

- It is necessary to synchronize the slave server with the master server's replication binary log. If binary logging has not previously been running on the master, run the following statement on the slave:

```
mysqlS> CHANGE MASTER TO
-> MASTER_LOG_FILE='',
-> MASTER_LOG_POS=4;
```

This instructs the slave to begin reading the master's binary log from the log's starting point. Otherwise—that is, if you are loading data from the master using a backup—see [Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”](#), for information on how to obtain the correct values to use for `MASTER_LOG_FILE` and `MASTER_LOG_POS` in such cases.

- Finally, you must instruct the slave to begin applying replication by issuing this command from the `mysql` client on the replication slave:

```
mysqlS> START SLAVE;
```

This also initiates the transmission of replication data from the master to the slave.

It is also possible to use two replication channels, in a manner similar to the procedure described in the next section; the differences between this and using a single replication channel are covered in [Section 17.6.7, “Using Two Replication Channels for MySQL Cluster Replication”](#).

It is also possible to improve cluster replication performance by enabling *batched updates*. This can be accomplished by setting the `slave_allow_batching` [2155] system variable on the slave `mysqld` processes. Normally, updates are applied as soon as they are received. However, the use of batching causes updates to be applied in 32 KB batches, which can result in higher throughput and less CPU usage, particularly where individual updates are relatively small.



Note

Slave batching works on a per-epoch basis; updates belonging to more than one transaction can be sent as part of the same batch.

All outstanding updates are applied when the end of an epoch is reached, even if the updates total less than 32 KB.

Batching can be turned on and off at runtime. To activate it at runtime, you can use either of these two statements:

```
SET GLOBAL slave_allow_batching = 1;
SET GLOBAL slave_allow_batching = ON;
```

If a particular batch causes problems (such as a statement whose effects do not appear to be replicated correctly), slave batching can be deactivated using either of the following statements:

```
SET GLOBAL slave_allow_batching = 0;
SET GLOBAL slave_allow_batching = OFF;
```

You can check whether slave batching is currently being used by means of an appropriate `SHOW VARIABLES` statement, like this one:

```
mysql> SHOW VARIABLES LIKE 'slave%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slave_allow_batching | ON |
| slave_compressed_protocol | OFF |
| slave_load_tmpdir | /tmp |
| slave_net_timeout | 3600 |
| slave_skip_errors | OFF |
| slave_transaction_retries | 10 |
+-----+-----+
6 rows in set (0.00 sec)
```

17.6.7. Using Two Replication Channels for MySQL Cluster Replication

In a more complete example scenario, we envision two replication channels to provide redundancy and thereby guard against possible failure of a single replication channel. This requires a total of four replication servers, two masters for the master cluster and two slave servers for the slave cluster. For purposes of the discussion that follows, we assume that unique identifiers are assigned as shown here:

Server ID	Description
1	Master - primary replication channel (<i>M</i>)
2	Master - secondary replication channel (<i>M'</i>)
3	Slave - primary replication channel (<i>S</i>)
4	Slave - secondary replication channel (<i>S'</i>)

Setting up replication with two channels is not radically different from setting up a single replication channel. First, the `mysqld` processes for the primary and secondary replication masters must be started, followed by those for the primary and secondary slaves. Then the replication processes may be initiated by issuing the `START SLAVE` statement on each of the slaves. The commands and the order in which they need to be issued are shown here:

1. Start the primary replication master:

```
shellM> mysqld --ndbcluster --server-id=1 \
             --log-bin --binlog-format=ROW &
```

2. Start the secondary replication master:

```
shellM'> mysqld --ndbcluster --server-id=2 \
            --log-bin --binlog-format=ROW &
```

3. Start the primary replication slave server:

```
shellS> mysqld --ndbcluster --server-id=3 \
            --skip-slave-start &
```

4. Start the secondary replication slave:

```
shellS'> mysqld --ndbcluster --server-id=4 \
            --skip-slave-start &
```

5. Finally, initiate replication on the primary channel by executing the `START SLAVE` statement on the primary slave as shown here:

```
mysqlS> START SLAVE;
```



Warning

Only the primary channel is to be started at this point. The secondary replication channel is to be started only in the event that the primary replication channel fails, as described in [Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”](#). Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replication slaves.

As mentioned previously, it is not necessary to enable binary logging on replication slaves.

17.6.8. Implementing Failover with MySQL Cluster Replication

In the event that the primary Cluster replication process fails, it is possible to switch over to the secondary replication channel. The following procedure describes the steps required to accomplish this.

1. Obtain the time of the most recent global checkpoint (GCP). That is, you need to determine the most recent epoch from the `ndb_apply_status` table on the slave cluster, which can be found using the following query:

```
mysqlS'> SELECT @latest:=MAX(epoch)
-> FROM mysql.ndb_apply_status;
```

2. Using the information obtained from the query shown in Step 1, obtain the corresponding records from the `ndb_binlog_index` table on the master cluster.

Prior to MySQL Cluster NDB 7.2.6, you should use the following query to accomplish this task:

```
mysqlM'> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch > @latest
-> ORDER BY epoch ASC LIMIT 1;
```

Beginning with MySQL Cluster NDB 7.2.6, you can take advantage of the improved binary logging of DDL statements implemented in those and later versions by using the following query to obtain the needed records from the master's `ndb_binlog_index` table:

```
mysqlM'> SELECT
-> @file:=SUBSTRING_INDEX(next_file, '/', -1),
-> @pos:=next_position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch = @latest
-> ORDER BY epoch ASC LIMIT 1;
```

In either case, these are the records saved on the master since the failure of the primary replication channel. We have employed a user variable `@latest` here to represent the value obtained in Step 1. Of course, it is not possible for one `mysqld` instance to access user variables set on another server instance directly. These values must be “plugged in” to the second query manually or in application code.



Important

If (and only if) you use the second of the two queries just shown against `ndb_binlog_index` (that is, the query that employs the `next_position` and `next_file` columns), you must ensure that the slave `mysqld` is started

with `--slave-skip-errors=ddl_exist_errors [1883]` before executing `START SLAVE`. Otherwise, replication may stop with duplicate DDL errors.

- Now it is possible to synchronize the secondary channel by running the following query on the secondary slave server:

```
mysqlS'> CHANGE MASTER TO
->     MASTER_LOG_FILE='@file',
->     MASTER_LOG_POS=@pos;
```

Again we have employed user variables (in this case `@file` and `@pos`) to represent the values obtained in Step 2 and applied in Step 3; in practice these values must be inserted manually or using application code that can access both of the servers involved.



Note

`@file` is a string value such as `'/var/log/mysql/replication-master-bin.00001'`, and so must be quoted when used in SQL or application code. However, the value represented by `@pos` must *not* be quoted. Although MySQL normally attempts to convert strings to numbers, this case is an exception.

- You can now initiate replication on the secondary channel by issuing the appropriate command on the secondary slave `mysqlS`:

```
mysqlS'> START SLAVE;
```

Once the secondary replication channel is active, you can investigate the failure of the primary and effect repairs. The precise actions required to do this will depend upon the reasons for which the primary channel failed.



Warning

The secondary replication channel is to be started only if and when the primary replication channel has failed. Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replication slaves.

If the failure is limited to a single server, it should (in theory) be possible to replicate from `M` to `S'`, or from `M'` to `S`; however, this has not yet been tested.

17.6.9. MySQL Cluster Backups With MySQL Cluster Replication

This section discusses making backups and restoring from them using MySQL Cluster replication. We assume that the replication servers have already been configured as covered previously (see [Section 17.6.5, “Preparing the MySQL Cluster for Replication”](#), and the sections immediately following). This having been done, the procedure for making a backup and then restoring from it is as follows:

- There are two different methods by which the backup may be started.
 - Method A.** This method requires that the cluster backup process was previously enabled on the master server, prior to starting the replication process. This can be done by including the following line in a `[mysql_cluster]` section in the `my.cnf` file, where `management_host` is the IP address or host name of the NDB management server for the master cluster, and `port` is the management server's port number:

```
ndb-connectstring=management_host[:port]
```

**Note**

The port number needs to be specified only if the default port (1186) is not being used. See [Section 17.2.3, “Initial Configuration of MySQL Cluster”](#), for more information about ports and port allocation in MySQL Cluster.

In this case, the backup can be started by executing this statement on the replication master:

```
shellM> ndb_mgm -e "START BACKUP"
```

- **Method B.** If the `my.cnf` file does not specify where to find the management host, you can start the backup process by passing this information to the NDB management client as part of the `START BACKUP` command. This can be done as shown here, where `management_host` and `port` are the host name and port number of the management server:

```
shellM> ndb_mgm management_host:port -e "START BACKUP"
```

In our scenario as outlined earlier (see [Section 17.6.5, “Preparing the MySQL Cluster for Replication”](#)), this would be executed as follows:

```
shellM> ndb_mgm rep-master:1186 -e "START BACKUP"
```

2. Copy the cluster backup files to the slave that is being brought on line. Each system running an `ndbd` process for the master cluster will have cluster backup files located on it, and *all* of these files must be copied to the slave to ensure a successful restore. The backup files can be copied into any directory on the computer where the slave management host resides, so long as the MySQL and NDB binaries have read permissions in that directory. In this case, we will assume that these files have been copied into the directory `/var/BACKUPS/BACKUP-1`.

It is not necessary that the slave cluster have the same number of `ndbd` processes (data nodes) as the master; however, it is highly recommended this number be the same. It *is* necessary that the slave be started with the `--skip-slave-start` option, to prevent premature startup of the replication process.

3. Create any databases on the slave cluster that are present on the master cluster that are to be replicated to the slave.

**Important**

A `CREATE DATABASE` (or `CREATE SCHEMA`) statement corresponding to each database to be replicated must be executed on each SQL node in the slave cluster.

4. Reset the slave cluster using this statement in the MySQL Monitor:

```
mysqlS> RESET SLAVE;
```

It is important to make sure that the slave's `apply_status` table does not contain any records prior to running the restore process. You can accomplish this by running this SQL statement on the slave:

```
mysqlS> DELETE FROM mysql.ndb_apply_status;
```

5. You can now start the cluster restoration process on the replication slave using the `ndb_restore` command for each backup file in turn. For the first of these, it is necessary to include the `-m` option to restore the cluster metadata:

```
shellS> ndb_restore -c slave_host:port -n node-id \  
-b backup-id -m -r dir
```

`dir` is the path to the directory where the backup files have been placed on the replication slave. For the `ndb_restore` commands corresponding to the remaining backup files, the `-m` option should *not* be used.

For restoring from a master cluster with four data nodes (as shown in the figure in [Section 17.6, “MySQL Cluster Replication”](#)) where the backup files have been copied to the directory `/var/BACKUPS/BACKUP-1`, the proper sequence of commands to be executed on the slave might look like this:

```
shellS> ndb_restore -c rep-slave:1186 -n 2 -b 1 -m \
-r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 3 -b 1 \
-r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 4 -b 1 \
-r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 5 -b 1 -e \
-r ./var/BACKUPS/BACKUP-1
```



Important

The `-e` (or `--restore-epoch`) option in the final invocation of `ndb_restore` in this example is required in order that the epoch is written to the slave `mysql.ndb_apply_status`. Without this information, the slave will not be able to synchronize properly with the master. (See [Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”](#).)

- Now you need to obtain the most recent epoch from the `ndb_apply_status` table on the slave (as discussed in [Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”](#)):

```
mysqlS> SELECT @latest:=MAX(epoch)
FROM mysql.ndb_apply_status;
```

- Using `@latest` as the epoch value obtained in the previous step, you can obtain the correct starting position `@pos` in the correct binary log file `@file` from the master's `mysql.ndb_binlog_index` table using the query shown here:

```
mysqlM> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch > @latest
-> ORDER BY epoch ASC LIMIT 1;
```

In the event that there is currently no replication traffic, you can get this information by running `SHOW MASTER STATUS` on the master and using the value in the `Position` column for the file whose name has the suffix with the greatest value for all files shown in the `File` column. However, in this case, you must determine this and supply it in the next step manually or by parsing the output with a script.

- Using the values obtained in the previous step, you can now issue the appropriate `CHANGE MASTER TO` statement in the slave's `mysql` client:

```
mysqlS> CHANGE MASTER TO
-> MASTER_LOG_FILE='@file',
-> MASTER_LOG_POS=@pos;
```

- Now that the slave “knows” from what point in which `binlog` file to start reading data from the master, you can cause the slave to begin replicating with this standard MySQL statement:

```
mysqlS> START SLAVE;
```

To perform a backup and restore on a second replication channel, it is necessary only to repeat these steps, substituting the host names and IDs of the secondary master and slave for those of the primary master and slave replication servers where appropriate, and running the preceding statements on them.

For additional information on performing Cluster backups and restoring Cluster from backups, see [Section 17.5.3, “Online Backup of MySQL Cluster”](#).

17.6.9.1. MySQL Cluster Replication: Automating Synchronization of the Replication Slave to the Master Binary Log

It is possible to automate much of the process described in the previous section (see [Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”](#)). The following Perl script `reset-slave.pl` serves as an example of how you can do this.

```
#!/user/bin/perl -w

# file: reset-slave.pl

# Copyright ©2005 MySQL AB

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to:
# Free Software Foundation, Inc.
# 59 Temple Place, Suite 330
# Boston, MA 02111-1307 USA
#
# Version 1.1

##### Includes #####

use DBI;

##### Globals #####

my $m_host='';
my $m_port='';
my $m_user='';
my $m_pass='';
my $s_host='';
my $s_port='';
my $s_user='';
my $s_pass='';
my $dbhM='';
my $dbhS='';

##### Sub Prototypes #####

sub CollectCommandPromptInfo;
sub ConnectToDatabases;
sub DisconnectFromDatabases;
sub GetSlaveEpoch;
sub GetMasterInfo;
sub UpdateSlave;
```

MySQL Cluster Backups With MySQL Cluster Replication

```
##### Program Main #####

CollectCommandPromptInfo;
ConnectToDatabases;
GetSlaveEpoch;
GetMasterInfo;
UpdateSlave;
DisconnectFromDatabases;

##### Collect Command Prompt Info #####

sub CollectCommandPromptInfo
{
  ### Check that user has supplied correct number of command line args
  die "Usage:\n
      reset-slave >master MySQL host< >master MySQL port< \n
              >master user< >master pass< >slave MySQL host< \n
              >slave MySQL port< >slave user< >slave pass< \n
  All 8 arguments must be passed. Use BLANK for NULL passwords\n"
  unless @ARGV == 8;

  $m_host = $ARGV[0];
  $m_port = $ARGV[1];
  $m_user = $ARGV[2];
  $m_pass = $ARGV[3];
  $s_host = $ARGV[4];
  $s_port = $ARGV[5];
  $s_user = $ARGV[6];
  $s_pass = $ARGV[7];

  if ($m_pass eq "BLANK") { $m_pass = '';}
  if ($s_pass eq "BLANK") { $s_pass = '';}
}

##### Make connections to both databases #####

sub ConnectToDatabases
{
  ### Connect to both master and slave cluster databases

  ### Connect to master
  $dbhM
  = DBI->connect(
    "dbi:mysql:database=mysql;host=$m_host;port=$m_port",
    "$m_user", "$m_pass")
  or die "Can't connect to Master Cluster MySQL process!
        Error: $DBI::errstr\n";

  ### Connect to slave
  $dbhS
  = DBI->connect(
    "dbi:mysql:database=mysql;host=$s_host",
    "$s_user", "$s_pass")
  or die "Can't connect to Slave Cluster MySQL process!
        Error: $DBI::errstr\n";
}

##### Disconnect from both databases #####

sub DisconnectFromDatabases
{
  ### Disconnect from master

  $dbhM->disconnect
  or warn " Disconnection failed: $DBI::errstr\n";

  ### Disconnect from slave
}
```

```

$dbhS->disconnect
or warn " Disconnection failed: $DBI::errstr\n";
}

##### Find the last good GCI #####

sub GetSlaveEpoch
{
    $sth = $dbhS->prepare("SELECT MAX(epoch)
                          FROM mysql.ndb_apply_status;")
    or die "Error while preparing to select epoch from slave: ",
          $dbhS->errstr;

    $sth->execute
    or die "Selecting epoch from slave error: ", $sth->errstr;

    $sth->bind_col (1, \$epoch);
    $sth->fetch;
    print "\tSlave Epoch = $epoch\n";
    $sth->finish;
}

##### Find the position of the last GCI in the binary log #####

sub GetMasterInfo
{
    $sth = $dbhM->prepare("SELECT
                          SUBSTRING_INDEX(File, '/', -1), Position
                          FROM mysql.ndb_binlog_index
                          WHERE epoch > $epoch
                          ORDER BY epoch ASC LIMIT 1;")
    or die "Prepare to select from master error: ", $dbhM->errstr;

    $sth->execute
    or die "Selecting from master error: ", $sth->errstr;

    $sth->bind_col (1, \$binlog);
    $sth->bind_col (2, \$binpos);
    $sth->fetch;
    print "\tMaster binary log = $binlog\n";
    print "\tMaster binary log position = $binpos\n";
    $sth->finish;
}

##### Set the slave to process from that location #####

sub UpdateSlave
{
    $sth = $dbhS->prepare("CHANGE MASTER TO
                          MASTER_LOG_FILE='$binlog',
                          MASTER_LOG_POS=$binpos;")
    or die "Prepare to CHANGE MASTER error: ", $dbhS->errstr;

    $sth->execute
    or die "CHANGE MASTER on slave error: ", $sth->errstr;
    $sth->finish;
    print "\tSlave has been updated. You may now start the slave.\n";
}

# end reset-slave.pl

```

17.6.9.2. Point-In-Time Recovery Using MySQL Cluster Replication

Point-in-time recovery—that is, recovery of data changes made since a given point in time—is performed after restoring a full backup that returns the server to its state when the backup was made. Performing

point-in-time recovery of MySQL Cluster tables with MySQL Cluster and MySQL Cluster Replication can be accomplished using a native `NDB` data backup (taken by issuing `CREATE BACKUP` in the `ndb_mgm` client) and restoring the `ndb_binlog_index` table (from a dump made using `mysqldump`).

To perform point-in-time recovery of MySQL Cluster, it is necessary to follow the steps shown here:

1. Back up all `NDB` databases in the cluster, using the `START BACKUP` command in the `ndb_mgm` client (see [Section 17.5.3, “Online Backup of MySQL Cluster”](#)).
2. At some later point, prior to restoring the cluster, make a backup of the `mysql.ndb_binlog_index` table. It is probably simplest to use `mysqldump` for this task. Also back up the binary log files at this time.

This backup should be updated regularly—perhaps even hourly—depending on your needs.

3. (*Catastrophic failure or error occurs.*)
4. Locate the last known good backup.
5. Clear the data node file systems (using `ndbd --initial [2178]` or `ndbmtdd --initial [2178]`).



Note

MySQL Cluster Disk Data tablespace and log files are not removed by `--initial [2178]`. You must delete these manually.

6. Use `DROP TABLE` or `TRUNCATE TABLE` with the `mysql.ndb_binlog_index` table.
7. Execute `ndb_restore`, restoring all data. You must include the `--restore_epoch [2222]` option when you run `ndb_restore`, so that the `ndb_apply_status` table is populated correctly. (See [Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”](#), for more information.)
8. Restore the `ndb_binlog_index` table from the output of `mysqldump` and restore the binary log files from backup, if necessary.
9. Find the epoch applied most recently—that is, the maximum `epoch` column value in the `ndb_apply_status` table—as the user variable `@LATEST_EPOCH` (emphasized):

```
SELECT @LATEST_EPOCH:=MAX(epoch)
FROM mysql.ndb_apply_status;
```

10. Find the latest binary log file (`@FIRST_FILE`) and position (`Position` column value) within this file that correspond to `@LATEST_EPOCH` in the `ndb_binlog_index` table:

```
SELECT Position, @FIRST_FILE:=File
FROM mysql.ndb_binlog_index
WHERE epoch > @LATEST_EPOCH ORDER BY epoch ASC LIMIT 1;
```

11. Using `mysqlbinlog`, replay the binary log events from the given file and position up to the point of the failure. (See [Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”](#).)

See also [Section 7.5, “Point-in-Time \(Incremental\) Recovery Using the Binary Log”](#), for more information about the binary log, replication, and incremental recovery.

17.6.10. MySQL Cluster Replication: Multi-Master and Circular Replication

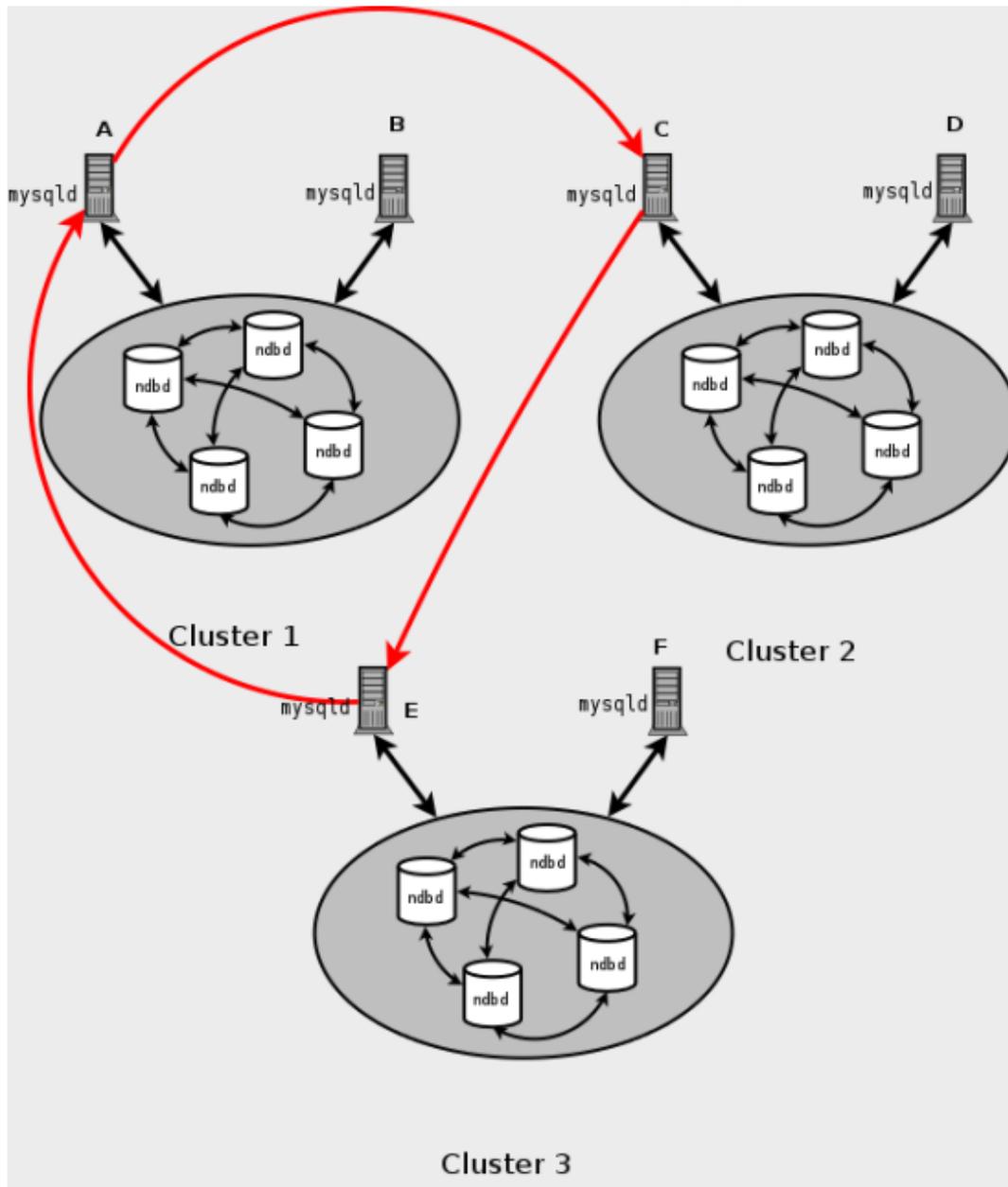
It is possible to use MySQL Cluster in multi-master replication, including circular replication between a number of MySQL Clusters.

Circular replication example. In the next few paragraphs we consider the example of a replication setup involving three MySQL Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication master for Cluster 2, Cluster 2 acts as the master for Cluster 3, and Cluster 3 acts as the master for Cluster 1. Each cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as the following conditions are met:

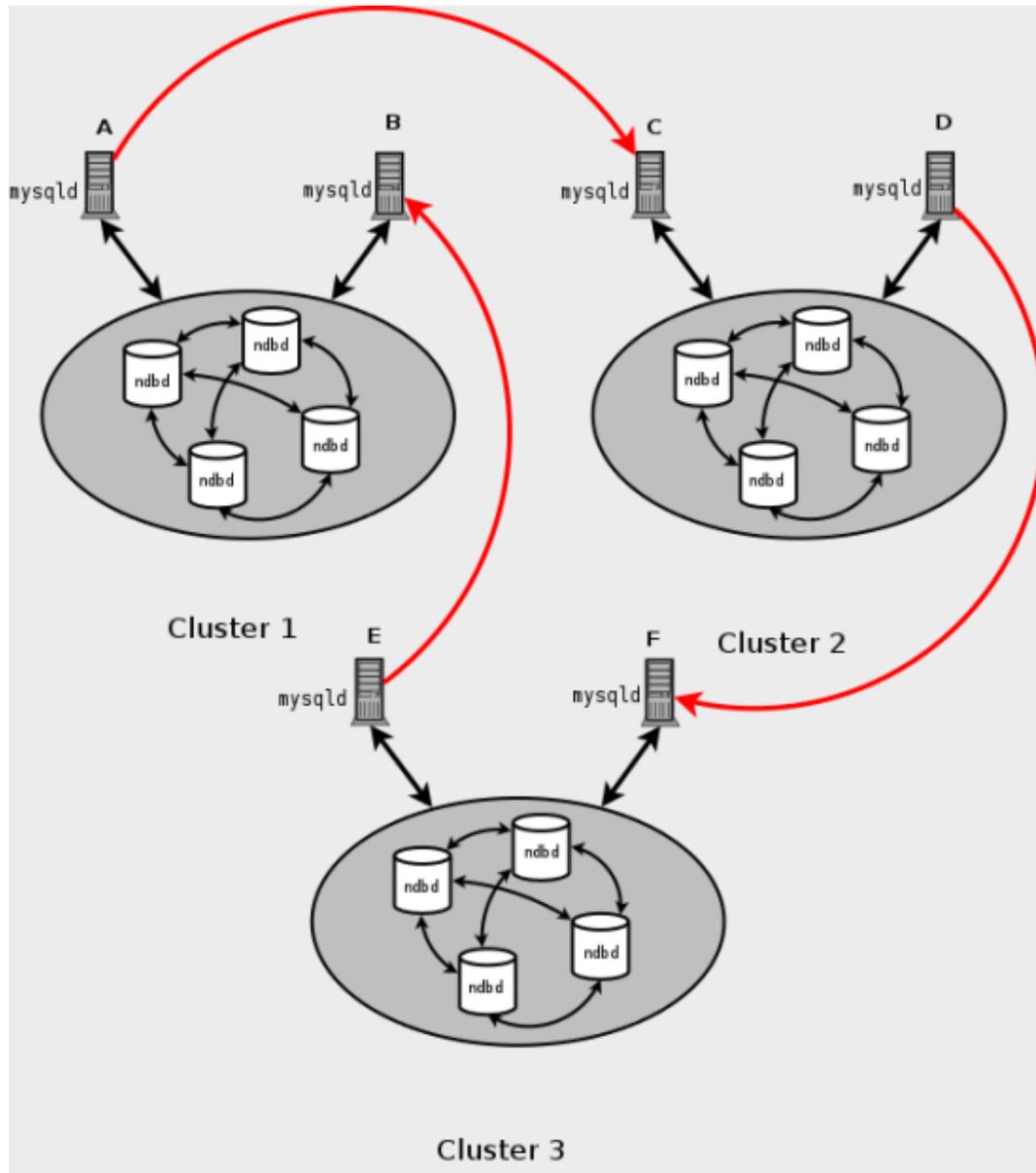
- The SQL nodes on all masters and slaves are the same
- All SQL nodes acting as replication masters and slaves are started using the `--log-slave-updates` option

This type of circular replication setup is shown in the following diagram:



In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line (indicated by the red arrows in the diagram) directly connects all SQL nodes used as replication masters and slaves.

It is also possible to set up circular replication in such a way that not all master SQL nodes are also slaves, as shown here:



In this case, different SQL nodes in each cluster are used as replication masters and slaves. However, you must *not* start any of the SQL nodes using `--log-slave-updates`. This type of circular replication scheme for MySQL Cluster, in which the line of replication (again indicated by the red arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.

**Important**

You should execute the following statement before starting circular replication:

```
mysql> SET GLOBAL SLAVE_EXEC_MODE = 'IDEMPOTENT';
```

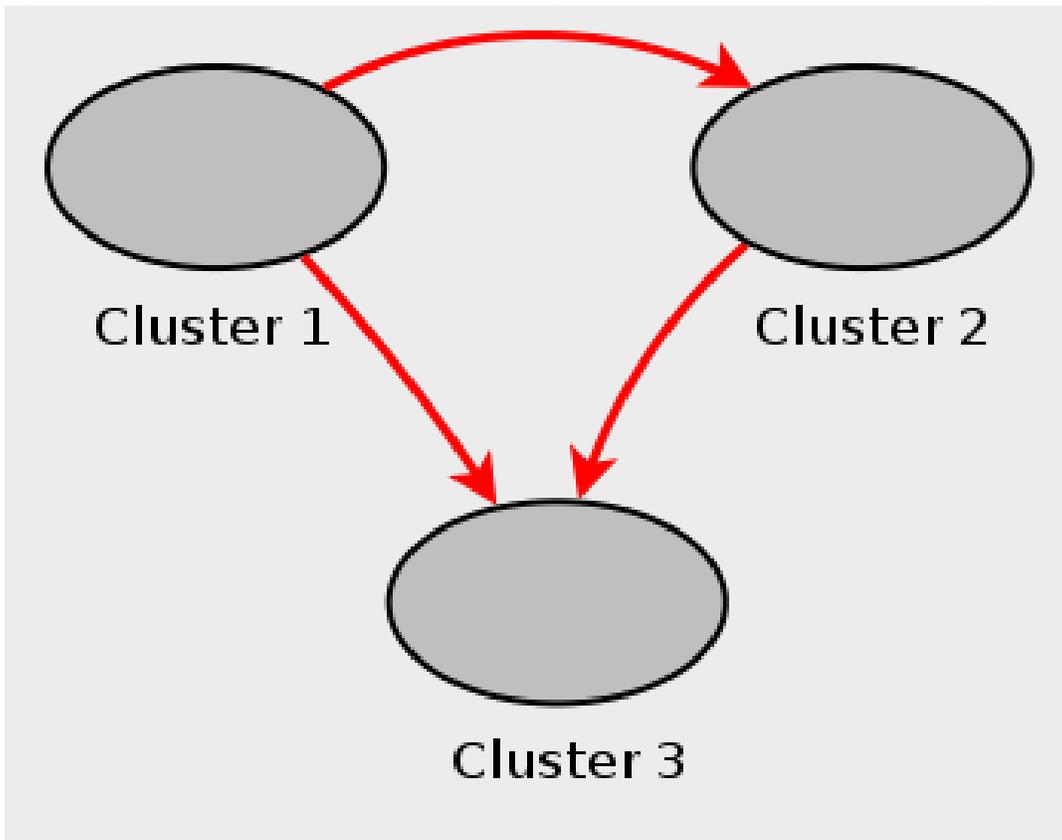
This is necessary to suppress duplicate-key and other errors that otherwise break circular replication in MySQL Cluster. `IDEMPOTENT` mode is also required for multi-master replication when using MySQL Cluster. (Bug #31609)

See [slave_exec_mode \[1887\]](#), for more information.

Using NDB-native backup and restore to initialize a slave MySQL Cluster. When setting up circular replication, it is possible to initialize the slave cluster by using the management client `BACKUP` command on one MySQL Cluster to create a backup and then applying this backup on another MySQL Cluster using `ndb_restore`. However, this does not automatically create binary logs on the second MySQL Cluster's SQL node acting as the replication slave. In order to cause the binary logs to be created, you must issue a `SHOW TABLES` statement on that SQL node; this should be done prior to running `START SLAVE`.

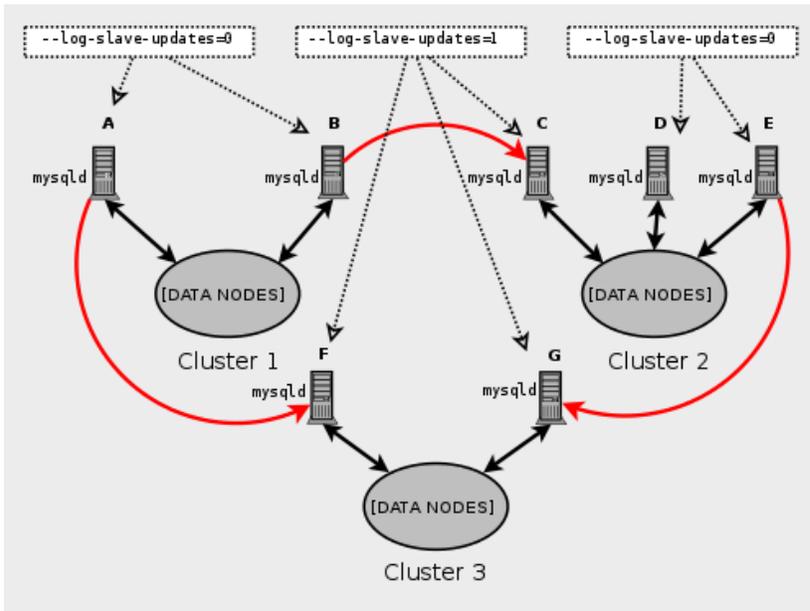
This is a known issue which we intend to address in a future release.

Multi-master failover example. In this section, we discuss failover in a multi-master MySQL Cluster replication setup with three MySQL Clusters having server IDs 1, 2, and 3. In this scenario, Cluster 1 replicates to Clusters 2 and 3; Cluster 2 also replicates to Cluster 3. This relationship is shown here:



In other words, data replicates from Cluster 1 to Cluster 3 through 2 different routes: directly, and by way of Cluster 2.

Not all MySQL servers taking part in multi-master replication must act as both master and slave, and a given MySQL Cluster might use different SQL nodes for different replication channels. Such a case is shown here:



MySQL servers acting as replication slaves must be run with the `--log-slave-updates` [1870] option. Which `mysqld` processes require this option is also shown in the preceding diagram.



Note

Using the `--log-slave-updates` [1870] option has no effect on servers not being run as replication slaves.

The need for failover arises when one of the replicating clusters goes down. In this example, we consider the case where Cluster 1 is lost to service, and so Cluster 3 loses 2 sources of updates from Cluster 1. Because replication between MySQL Clusters is asynchronous, there is no guarantee that Cluster 3's updates originating directly from Cluster 1 are more recent than those received through Cluster 2. You can handle this by ensuring that Cluster 3 catches up to Cluster 2 with regard to updates from Cluster 1. In terms of MySQL servers, this means that you need to replicate any outstanding updates from MySQL server C to server F.

On server C, perform the following queries:

```
mysqlC> SELECT @latest:=MAX(epoch)
-> FROM mysql.ndb_apply_status
-> WHERE server_id=1;

mysqlC> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE orig_epoch >= @latest
-> AND orig_server_id = 1
-> ORDER BY epoch ASC LIMIT 1;
```

Copy over the values for `@file` and `@pos` manually from server C to server F (or have your application perform the equivalent). Then, on server F, execute the following `CHANGE MASTER TO` statement:

```
mysqlF> CHANGE MASTER TO
-> MASTER_HOST = 'serverC'
```

```
-> MASTER_LOG_FILE='@file',
-> MASTER_LOG_POS=@pos;
```

Once this has been done, you can issue a `START SLAVE` statement on MySQL server F, and any missing updates originating from server B will be replicated to server F.

The `CHANGE MASTER TO` statement also supports an `IGNORE_SERVER_IDS` option which takes a comma-separated list of server IDs and causes events originating from the corresponding servers to be ignored. For more information, see [Section 13.4.2.1, “CHANGE MASTER TO Syntax”](#), and [Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”](#).

17.6.11. MySQL Cluster Replication Conflict Resolution

When using a replication setup involving multiple masters (including circular replication), it is possible that different masters may try to update the same row on the slave with different data. Conflict resolution in MySQL Cluster Replication provides a means of resolving such conflicts by permitting a user-defined resolution column to be used to determine whether or not an update to the row on a given master should be applied on the slave.

Some types of conflict resolution supported by MySQL Cluster (`NDB$OLD()`, `NDB$MAX()`, `NDB$MAX_DELETE_WIN()`) implement this user-defined column as a “timestamp” column (although its type cannot be `TIMESTAMP`, as explained later in this section); epoch-based conflict resolution functions introduced in MySQL Cluster NDB 7.2.1 (`NDB$EPOCH()` and `NDB$EPOCH_TRANS()`) compare the order in which epochs are replicated. Different methods can be used to compare resolution column values on the slave when conflicts occur, as explained later in this section; the method used can be set on a per-table basis.



Important

Conflict resolution as described in this section is always applied on a row-by-row basis rather than a transactional basis. In addition, it is the application's responsibility to ensure that the resolution column is correctly populated with relevant values, so that the resolution function can make the appropriate choice when determining whether to apply an update.

Requirements. Preparations for conflict resolution must be made on both the master and the slave. These tasks are described in the following list:

- On the master writing the binary logs, you must determine which columns are sent (all columns or only those that have been updated). This is done for the MySQL Server as a whole by applying the `mysqld` startup option `--ndb-log-updated-only` [2372] (described later in this section) or on a per-table basis by entries in the `mysql.ndb_replication` table (see [The `ndb_replication` system table](#)).



Note

If you are replicating tables with very large columns (such as `TEXT` or `BLOB` columns), `--ndb-log-updated-only` can also be useful for reducing the size of the master and slave binary logs and avoiding possible replication failures due to exceeding `max_allowed_packet` [519].

See [Section 16.4.1.19, “Replication and `max_allowed_packet`”](#), for more information about this issue.

- On the slave, you must determine which type of conflict resolution to apply (“latest timestamp wins”, “same timestamp wins”, “primary wins”, “primary wins, complete transaction”, or none). This is done using the `mysql.ndb_replication` system table, on a per-table basis (see [The `ndb_replication` system table](#)).

- Prior to MySQL Cluster NDB 7.2.5, conflict detection and resolution did not always work properly unless set up for NDB tables created on the same server only (Bug #13578660).

When using the functions `NDB$OLD()`, `NDB$MAX()`, and `NDB$MAX_DELETE_WIN()` for timestamp-based conflict resolution, we often refer to the column used for determining updates as a “timestamp” column. However, the data type of this column is never `TIMESTAMP`; instead, its data type should be `INT` (`INTEGER`) or `BIGINT`. The “timestamp” column should also be `UNSIGNED` and `NOT NULL`.

The `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` functions discussed later in this section work by comparing the relative order of replication epochs applied on a primary and secondary MySQL Cluster, and do not make use of timestamps.

Master column control. We can see update operations in terms of “before” and “after” images—that is, the states of the table before and after the update is applied. Normally, when updating a table with a primary key, the “before” image is not of great interest; however, when we need to determine on a per-update basis whether or not to use the updated values on a replication slave, we need to make sure that both images are written to the master’s binary log. This is done with the `--ndb-log-update-as-write` [2373] option for `mysqld`, as described later in this section.



Important

Whether logging of complete rows or of updated columns only is done is decided when the MySQL server is started, and cannot be changed online; you must either restart `mysqld`, or start a new `mysqld` instance with different logging options.

Logging Full or Partial Rows (`--ndb-log-updated-only` Option)

Command-Line Format	<code>--ndb-log-updated-only</code>
Option-File Format	<code>ndb_log_updated_only</code>
Variable Name	<code>ndb_log_updated_only</code>
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values
	Type <code>boolean</code>
	Default <code>ON</code>

For purposes of conflict resolution, there are two basic methods of logging rows, as determined by the setting of the `--ndb-log-updated-only` option for `mysqld`:

- Log complete rows
- Log only column data that has been updated—that is, column data whose value has been set, regardless of whether or not this value was actually changed. This is the default behavior.

It is usually sufficient—and more efficient—to log updated columns only; however, if you need to log full rows, you can do so by setting `--ndb-log-updated-only` to `0` or `OFF`.

`--ndb-log-update-as-write` [2373] Option: Logging Changed Data as Updates

Command-Line Format	<code>--ndb-log-update-as-write</code>
Option-File Format	<code>ndb-log-update-as-write</code>
Variable Name	<code>ndb_log_update_as_write</code>

Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

The setting of the MySQL Server's `--ndb-log-update-as-write` [2373] option determines whether logging is performed with or without the “before” image. Because conflict resolution is done in the MySQL Server's update handler, it is necessary to control logging on the master such that updates are updates and not writes; that is, such that updates are treated as changes in existing rows rather than the writing of new rows (even though these replace existing rows). This option is turned on by default; in other words, updates are treated as writes. (That is, updates are by default written as `write_row` events in the binary log, rather than as `update_row` events.) To turn off the option, start the master `mysqld` with `--ndb-log-update-as-write=0` or `--ndb-log-update-as-write=OFF`.

Conflict resolution control. Conflict resolution is usually enabled on the server where conflicts can occur. Like logging method selection, it is enabled by entries in the `mysql.ndb_replication` table.

The `ndb_replication` system table. To enable conflict resolution, it is necessary to create an `ndb_replication` table in the `mysql` system database on the master, the slave, or both, depending on the conflict resolution type and method to be employed. This table is used to control logging and conflict resolution functions on a per-table basis, and has one row per table involved in replication. `ndb_replication` is created and filled with control information on the server where the conflict is to be resolved. In a simple master-slave setup where data can also be changed locally on the slave this will typically be the slave. In a more complex master-master (2-way) replication schema this will usually be all of the masters involved. Each row in `mysql.ndb_replication` corresponds to a table being replicated, and specifies how to log and resolve conflicts (that is, which conflict resolution function, if any, to use) for that table. The definition of the `mysql.ndb_replication` table is shown here:

```
CREATE TABLE mysql.ndb_replication (
  db VARBINARY(63),
  table_name VARBINARY(63),
  server_id INT UNSIGNED,
  binlog_type INT UNSIGNED,
  conflict_fn VARBINARY(128),
  PRIMARY KEY USING HASH (db, table_name, server_id)
) ENGINE=NDB
PARTITION BY KEY(db,table_name);
```

The columns in this table are described in the next few paragraphs.

db. The name of the database containing the table to be replicated.

table_name. The name of the table to be replicated.

server_id. The unique server ID of the MySQL instance (SQL node) where the table resides.

binlog_type. The type of binary logging to be employed. This is determined as shown in the following table:

Value	Internal Value	Description
0	<code>NBT_DEFAULT</code>	Use server default
1	<code>NBT_NO_LOGGING</code>	Do not log this table in the binary log
2	<code>NBT_UPDATED_ONLY</code>	Only updated attributes are logged

Value	Internal Value	Description
3	NBT_FULL	Log full row, even if not updated (MySQL server default behavior)
4	NBT_USE_UPDATE	(For generating NBT_UPDATED_ONLY_USE_UPDATE and NBT_FULL_USE_UPDATE values only—not intended for separate use)
5	[Not used]	---
6	NBT_UPDATED_ONLY_USE_UPDATE (equal to NBT_UPDATED_ONLY NBT_USE_UPDATE)	Use updated attributes, even if values are unchanged
7	NBT_FULL_USE_UPDATE (equal to NBT_FULL NBT_USE_UPDATE)	Use full row, even if values are unchanged

conflict_fn. The conflict resolution function to be applied. This function must be specified as one of those shown in the following list:

- NDB\$OLD(*column_name*)
- NDB\$MAX(*column_name*)
- NDB\$MAX_DELETE_WIN(*column_name*)
- NDB\$EPOCH() (MySQL Cluster NDB 7.2.1 and later)
- NDB\$EPOCH_TRANS() (MySQL Cluster NDB 7.2.1 and later)
- NULL

These functions are described in the next few paragraphs.

NDB\$OLD(*column_name*). If the value of *column_name* is the same on both the master and the slave, then the update is applied; otherwise, the update is not applied on the slave and an exception is written to the log. This is illustrated by the following pseudocode:

```
if (master_old_column_value == slave_current_column_value)
  apply_update();
else
  log_exception();
```

This function can be used for “same value wins” conflict resolution. This type of conflict resolution ensures that updates are not applied on the slave from the wrong master.



Important

The column value from the master's “before” image is used by this function.

NDB\$MAX(*column_name*). If the “timestamp” column value for a given row coming from the master is higher than that on the slave, it is applied; otherwise it is not applied on the slave. This is illustrated by the following pseudocode:

```
if (master_new_column_value > slave_current_column_value)
  apply_update();
```

This function can be used for “greatest timestamp wins” conflict resolution. This type of conflict resolution ensures that, in the event of a conflict, the version of the row that was most recently updated is the version that persists.

**Important**

The column value from the master's “after” image is used by this function.

`NDB$MAX_DELETE_WIN(column_name)`. This is a variation on `NDB$MAX()`. Due to the fact that no timestamp is available for a delete operation, a delete using `NDB$MAX()` is in fact processed as `NDB$OLD`. However, for some use cases, this is not optimal. For `NDB$MAX_DELETE_WIN()`, if the “timestamp” column value for a given row adding or updating an existing row coming from the master is higher than that on the slave, it is applied. However, delete operations are treated as always having the higher value. This is illustrated in the following pseudocode:

```
if ( (master_new_column_value > slave_current_column_value)
    ||
    operation.type == "delete" )
    apply_update();
```

This function can be used for “greatest timestamp, delete wins” conflict resolution. This type of conflict resolution ensures that, in the event of a conflict, the version of the row that was deleted or (otherwise) most recently updated is the version that persists.

**Note**

As with `NDB$MAX()`, the column value from the master's “after” image is the value used by this function.

`NDB$EPOCH()`. The `NDB$EPOCH()` function, available beginning with MySQL Cluster NDB 7.2.1, tracks the order in which replicated epochs are applied on a slave MySQL Cluster relative to changes originating on the slave. This relative ordering is used to determine whether changes originating on the slave are concurrent with any changes that originate locally, and are therefore potentially in conflict.

Most of what follows in the description of `NDB$EPOCH()` also applies to `NDB$EPOCH_TRANS()`. Any exceptions are noted in the text.

`NDB$EPOCH()` is asymmetric, operating on one MySQL Cluster in a two-cluster circular replication configuration (sometimes referred to as “active-active” replication). We refer here to cluster on which it operates as the primary, and the other as the secondary. The slave on the primary is responsible for detecting and handling conflicts, while the slave on the secondary is not involved in any conflict detection or handling.

When the slave on the primary detects conflicts, it injects events into its own binary log to compensate for these; this ensures that the secondary MySQL Cluster eventually realigns itself with the primary and so keeps the primary and secondary from diverging. This compensation and realignment mechanism requires that the primary MySQL Cluster always wins any conflicts with the secondary—that is, that the primary's changes are always used rather than those from the secondary in event of a conflict. This “primary always wins” rule has the following implications:

- Operations that change data, once committed on the primary, are fully persistent and will not be undone or rolled back by conflict detection and resolution.
- Data read from the primary is fully consistent. Any changes committed on the Primary (locally or from the slave) will not be reverted later.
- Operations that change data on the secondary may later be reverted if the primary determines that they are in conflict.
- Individual rows read on the secondary are self-consistent at all times, each row always reflecting either a state committed by the secondary, or one committed by the primary.

- Sets of rows read on the secondary may not necessarily be consistent at a given single point in time. For `NDB$EPOCH_TRANS()`, this is a transient state; for `NDB$EPOCH()`, it can be a persistent state.
- Assuming a period of sufficient length without any conflicts, all data on the secondary MySQL Cluster (eventually) becomes consistent with the primary's data.

`NDB$EPOCH()` and `NDB$EPOCH_TRANS()` do not require any user schema modifications, or application changes to provide conflict detection. However, careful thought must be given to the schema used, and the access patterns used, to verify that the complete system behaves within specified limits.

Each of the `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` functions can take an optional parameter; this is the number of bits to use to represent the lower 32 bits of the epoch, and should be set to no less than

```
CEIL( LOG2( TimeBetweenGlobalCheckpoints [2065] / TimeBetweenEpochs [2065] ), 1)
```

For the default values of these configuration parameters (2000 and 100 milliseconds, respectively), this gives a value of 5 bits, so the default value (6) should be sufficient, unless other values are used for `TimeBetweenGlobalCheckpoints` [2065], `TimeBetweenEpochs` [2065], or both. A value that is too small can result in false positives, while one that is too large could lead to excessive wasted space in the database.

Both `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` insert entries for conflicting rows into the relevant exceptions tables, provided that these tables have been defined according to the same exception table schema rules as described elsewhere in this section (see `NDB$OLD(column_name)`). Note that you need to create any exception table before creating the table with which it is to be used.

As with the other conflict detection functions discussed in this section, `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` are activated by including relevant entries in the `mysql.ndb_replication` table (see [The ndb_replication system table](#)). The roles of the primary and secondary MySQL Clusters in this scenario are fully determined by `mysql.ndb_replication` table entries.



Note

Because the conflict detection algorithms employed by `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` are asymmetric, you must use different values for the primary slave's and secondary slave's `server_id` entries.

`NDB$EPOCH()` and `NDB$EPOCH_TRANS()` status variables. MySQL Cluster NDB 7.2.1 introduces several status variables that can be used to monitor `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` conflict detection. You can see how many rows have been found in conflict by `NDB$EPOCH()` since this slave was last restarted from the current value of the `Ndb_conflict_fn_epoch` [2172] system status variable.

`Ndb_conflict_fn_epoch_trans` [2172] provides the number of rows that have been found directly in conflict by `NDB$EPOCH_TRANS()`; the number of rows actually realigned, including those affected due to their membership in or dependency on the same transactions as other conflicting rows, is given by `Ndb_conflict_trans_row_reject_count` [2173].

For more information, see [Section 17.3.4.4, “MySQL Cluster Status Variables”](#).

Limitations on `NDB$EPOCH()`. The following limitations currently apply when using `NDB$EPOCH()` to perform conflict detection:

- Conflicts are detected using MySQL Cluster epoch boundaries, with granularity proportional to `TimeBetweenEpochs` [2065] (default: 100 milliseconds). The minimum conflict window is the minimum time during which concurrent updates to the same data on both clusters always report a conflict. This is always a nonzero length of time, and is roughly proportional to $2 * (\text{latency} + \text{queueing} + \text{TimeBetweenEpochs})$. This implies that—assuming the default for `TimeBetweenEpochs` [2065] and ignoring any latency between clusters (as well as any queuing delays)—the minimum conflict window

size is approximately 200 milliseconds. This minimum window should be considered when looking at expected application “race” patterns.

- Additional storage is required for tables using the `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` functions; from 1 to 32 bits extra space per row is required, depending on the value passed to the function.
- Conflicts between delete operations may result in divergence between the primary and secondary. When a row is deleted on both clusters concurrently, the conflict can be detected, but is not recorded, since the row is deleted. This means that further conflicts during the propagation of any subsequent realignment operations will not be detected, which can lead to divergence.

Deletes should be externally serialized, or routed to one cluster only. Alternatively, a separate row should be updated transactionally with such deletes and any inserts that follow them, so that conflicts can be tracked across row deletes. This may require changes in applications.

- Only two MySQL Clusters in a circular “active-active” configuration are currently supported when using `NDB$EPOCH()` or `NDB$EPOCH_TRANS()` for conflict detection.
- Tables having `BLOB` or `TEXT` columns are not currently supported with `NDB$EPOCH()` or `NDB$EPOCH_TRANS()`.

`NDB$EPOCH_TRANS()`. `NDB$EPOCH_TRANS()` extends the `NDB$EPOCH()` function, and, like `NDB$EPOCH()`, is available beginning with MySQL Cluster NDB 7.2.1. Conflicts are detected and handled in the same way using the “primary wins all” rule (see `NDB$EPOCH()`) but with the extra condition that any other rows updated in the same transaction in which the conflict occurred are also regarded as being in conflict. In other words, where `NDB$EPOCH()` realigns individual conflicting rows on the secondary, `NDB$EPOCH_TRANS()` realigns conflicting transactions.

In addition, any transactions which are detectably dependent on a conflicting transaction are also regarded as being in conflict, these dependencies being determined by the contents of the secondary cluster’s binary log. Since the binary log contains only data modification operations (inserts, updates, and deletes), only overlapping data modifications are used to determine dependencies between transactions.

`NDB$EPOCH_TRANS()` is subject to the same conditions and limitations as `NDB$EPOCH()`, and in addition requires that Version 2 binary log row events are used (`--log-bin-use-v1-row-events` [1893] equal to 0), which adds a storage overhead of 2 bytes per event in the binary log. In addition, all transaction IDs must be recorded in the secondary’s binary log (`--ndb-log-transaction-id` [2143] option), which adds a further variable overhead (up to 13 bytes per row).

See `NDB$EPOCH()`.

`NULL`. Indicates that conflict resolution is not to be used for the corresponding table.

Status information. A server status variable `Ndb_conflict_fn_max` [2172] provides a count of the number of times that a row was not applied on the current SQL node due to “greatest timestamp wins” conflict resolution since the last time that `mysqld` was started.

The number of times that a row was not applied as the result of “same timestamp wins” conflict resolution on a given `mysqld` since the last time it was restarted is given by the global status variable `Ndb_conflict_fn_old` [2172]. In addition to incrementing `Ndb_conflict_fn_old` [2172], the primary key of the row that was not used is inserted into an *exceptions table*, as explained later in this section.

Exceptions table. To use the `NDB$OLD()` conflict resolution function, it is also necessary to create an exceptions table corresponding to each `NDB` table for which this type of conflict resolution is to be employed. This is also true when using `NDB$EPOCH()` or `NDB$EPOCH_TRANS()` in MySQL Cluster NDB 7.2.1 and later. The name of this table is that of the table for which conflict resolution is to be applied,

with the string `$EX` appended. (For example, if the name of the original table is `mytable`, the name of the corresponding exception table name should be `mytable$EX`.) This table is created as follows:

```
CREATE TABLE original_table$EX (
  server_id INT UNSIGNED,
  master_server_id INT UNSIGNED,
  master_epoch BIGINT UNSIGNED,
  count INT UNSIGNED,
  original_table_pk_columns,
  [additional_columns,]
  PRIMARY KEY(server_id, master_server_id, master_epoch, count)
) ENGINE=NDB;
```

The first four columns are required. Following these columns, the columns making up the original table's primary key should be copied in the order in which they are used to define the primary key of the original table.



Note

The names of the first four columns and the columns matching the original table's primary key columns are not critical; however, we suggest for reasons of clarity and consistency, that you use the names shown here for the `server_id` [551], `master_server_id`, `master_epoch`, and `count` columns, and that you use the same names as in the original table for the columns matching those in the original table's primary key.

The data types for the columns duplicating the primary key columns of the original table should be the same as for (or larger than) the original columns.

Additional columns may optionally be defined following these columns, but not before any of them; any such extra columns cannot be `NOT NULL`. The exception table's primary key must be defined as shown. The exception table must use the `NDB` storage engine. An example that uses `NDB$OLD()` with an exceptions table is shown later in this section.



Important

The `mysql.ndb_replication` table is read when a data table is set up for replication, so the row corresponding to a table to be replicated must be inserted into `mysql.ndb_replication` *before* the table to be replicated is created.

Examples. The following examples assume that you have already a working MySQL Cluster replication setup, as described in [Section 17.6.5, “Preparing the MySQL Cluster for Replication”](#), and [Section 17.6.6, “Starting MySQL Cluster Replication \(Single Replication Channel\)”](#).

- **`NDB$MAX()` example.** Suppose you wish to enable “greatest timestamp wins” conflict resolution on table `test.t1`, using column `mycol` as the “timestamp”. This can be done using the following steps:
 1. Make sure that you have started the master `mysqld` with `--ndb-log-update-as-write=OFF`.
 2. On the master, perform this `INSERT` statement:

```
INSERT INTO mysql.ndb_replication
VALUES ('test', 't1', 0, NULL, 'NDB$MAX(mycol)');
```

Inserting a 0 into the `server_id` [551] indicates that all SQL nodes accessing this table should use conflict resolution. If you want to use conflict resolution on a specific `mysqld` only, use the actual server ID.

Inserting `NULL` into the `binlog_type` column has the same effect as inserting 0 (`NBT_DEFAULT`); the server default is used.

3. Create the `test.t1` table:

```
CREATE TABLE test.t1 (
  columns
  mycol INT UNSIGNED,
  columns
) ENGINE=NDB;
```

Now, when updates are done on this table, conflict resolution is applied, and the version of the row having the greatest value for `mycol` is written to the slave.



Note

Other `binlog_type` options—such as `NBT_UPDATED_ONLY_USE_UPDATE`—should be used to control logging on the master using the `ndb_replication` table rather than by using command-line options.

- **`NDB$OLD()` example.** Suppose an `NDB` table such as the one defined here is being replicated, and you wish to enable “same timestamp wins” conflict resolution for updates to this table:

```
CREATE TABLE test.t2 (
  a INT UNSIGNED NOT NULL,
  b CHAR(25) NOT NULL,
  columns,
  mycol INT UNSIGNED NOT NULL,
  columns,
  PRIMARY KEY pk (a, b)
) ENGINE=NDB;
```

The following steps are required, in the order shown:

1. First—and *prior* to creating `test.t2`—you must insert a row into the `mysql.ndb_replication` table, as shown here:

```
INSERT INTO mysql.ndb_replication
VALUES ('test', 't2', 0, NULL, 'NDB$OLD(mycol)');
```

Possible values for the `binlog_type` column are shown earlier in this section. The value `'NDB$OLD(mycol)'` should be inserted into the `conflict_fn` column.

2. Create an appropriate exceptions table for `test.t2`. The table creation statement shown here includes all required columns; any additional columns must be declared following these columns, and before the definition of the table's primary key.

```
CREATE TABLE test.t2$EX (
  server_id SMALLINT UNSIGNED,
  master_server_id INT UNSIGNED,
  master_epoch BIGINT UNSIGNED,
  count BIGINT UNSIGNED,
  a INT UNSIGNED NOT NULL,
  b CHAR(25) NOT NULL,
  [additional_columns,]
  PRIMARY KEY(server_id, master_server_id, master_epoch, count)
) ENGINE=NDB;
```

3. Create the table `test.t2` as shown previously.

These steps must be followed for every table for which you wish to perform conflict resolution using `NDB $OLD()`. For each such table, there must be a corresponding row in `mysql.ndb_replication`, and there must be an exceptions table in the same database as the table being replicated.

17.7. MySQL Cluster Release Notes

MySQL Cluster release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Cluster are located at [MySQL Cluster 7.2 Release Notes](#).

Chapter 18. Partitioning

Table of Contents

18.1. Overview of Partitioning in MySQL	2383
18.2. Partitioning Types	2385
18.2.1. RANGE Partitioning	2387
18.2.2. LIST Partitioning	2391
18.2.3. COLUMNS Partitioning	2393
18.2.4. HASH Partitioning	2401
18.2.5. KEY Partitioning	2404
18.2.6. Subpartitioning	2406
18.2.7. How MySQL Partitioning Handles NULL	2409
18.3. Partition Management	2414
18.3.1. Management of RANGE and LIST Partitions	2415
18.3.2. Management of HASH and KEY Partitions	2420
18.3.3. Maintenance of Partitions	2421
18.3.4. Obtaining Information About Partitions	2422
18.4. Partition Pruning	2424
18.5. Restrictions and Limitations on Partitioning	2427
18.5.1. Partitioning Keys, Primary Keys, and Unique Keys	2433
18.5.2. Partitioning Limitations Relating to Storage Engines	2437
18.5.3. Partitioning Limitations Relating to Functions	2437
18.5.4. Partitioning and Table-Level Locking	2438

This chapter discusses MySQL's implementation of *user-defined partitioning*. You can determine whether your MySQL Server supports partitioning by means of a [SHOW VARIABLES](#) statement such as this one:

```
mysql> SHOW VARIABLES LIKE '%partition%';

+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_partitioning | YES |
+-----+-----+
1 row in set (0.00 sec)
```

You can also check the output of the [SHOW PLUGINS](#) statement, as shown here:

```
mysql> SHOW PLUGINS;

+-----+-----+-----+-----+-----+
| Name          | Status | Type          | Library | License |
+-----+-----+-----+-----+-----+
| binlog        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| partition    | ACTIVE | STORAGE ENGINE | NULL   | GPL   |
| ARCHIVE       | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| BLACKHOLE     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| CSV           | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| FEDERATED     | DISABLED | STORAGE ENGINE | NULL    | GPL     |
| MEMORY        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| InnoDB        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MRG_MYISAM    | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MyISAM        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndbcluster    | DISABLED | STORAGE ENGINE | NULL    | GPL     |
+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

If you do not see the `have_partitioning` [504] variable with the value `YES` listed in the output of an appropriate `SHOW VARIABLES` statement, or if you do not see the `partition` plugin listed with the value `ACTIVE` for the `Status` column in the output of `SHOW PLUGINS` (show in bold text in the example just given), then your version of MySQL was not built with partitioning support.

MySQL 5.5 Community binaries provided by Oracle include partitioning support. For information about partitioning support offered in commercial MySQL Server binaries, see *MySQL Enterprise Server 5.1* on the MySQL Web site at <http://www.mysql.com/products/enterprise/server.html>.

To enable partitioning if you are compiling MySQL 5.5 from source, the build must be configured with the `-DWITH_PARTITION_STORAGE_ENGINE` option. For more information about building MySQL, see [Section 2.10, “Installing MySQL from Source”](#).

If your MySQL binary is built with partitioning support, nothing further needs to be done to enable it (for example, no special entries are required in your `my.cnf` file).

If you want to disable partitioning support, you can start the MySQL Server with the `--skip-partition` [462] option, in which case the value of `have_partitioning` [504] is `DISABLED`. However, if you do this, you cannot access any partitioned tables until the server is once again restarted without the `--skip-partition` [462] option.

An introduction to partitioning and partitioning concepts may be found in [Section 18.1, “Overview of Partitioning in MySQL”](#).

MySQL supports several types of partitioning, which are discussed in [Section 18.2, “Partitioning Types”](#), as well as subpartitioning, which is described in [Section 18.2.6, “Subpartitioning”](#).

Methods of adding, removing, and altering partitions in existing partitioned tables are covered in [Section 18.3, “Partition Management”](#).

Table maintenance commands for use with partitioned tables are discussed in [Section 18.3.3, “Maintenance of Partitions”](#).

The `PARTITIONS` table in the `INFORMATION_SCHEMA` database provides information about partitions and partitioned tables. See [Section 20.12, “The INFORMATION_SCHEMA PARTITIONS Table”](#), for more information; for some examples of queries against this table, see [Section 18.2.7, “How MySQL Partitioning Handles NULL”](#).

For known issues with partitioning in MySQL 5.5, see [Section 18.5, “Restrictions and Limitations on Partitioning”](#).

You may also find the following resources to be useful when working with partitioned tables.

Additional Resources. Other sources of information about user-defined partitioning in MySQL include the following:

- [MySQL Partitioning Forum](#)

This is the official discussion forum for those interested in or experimenting with MySQL Partitioning technology. It features announcements and updates from MySQL developers and others. It is monitored by members of the Partitioning Development and Documentation Teams.

- [Mikael Ronström's Blog](#)

MySQL Partitioning Architect and Lead Developer Mikael Ronström frequently posts articles here concerning his work with MySQL Partitioning and MySQL Cluster.

- [PlanetMySQL](#)

A MySQL news site featuring MySQL-related blogs, which should be of interest to anyone using my MySQL. We encourage you to check here for links to blogs kept by those working with MySQL Partitioning, or to have your own blog added to those covered.

MySQL 5.5 binaries are available from <http://dev.mysql.com/downloads/mysql/5.5.html>. However, for the latest partitioning bugfixes and feature additions, you can obtain the source from our Bazaar repository. To enable partitioning, the build must be configured with the `-DWITH_PARTITION_STORAGE_ENGINE` option. For more information about building MySQL, see [Section 2.10, “Installing MySQL from Source”](#). If you have problems compiling a partitioning-enabled MySQL 5.5 build, check the [MySQL Partitioning Forum](#) and ask for assistance there if you do not find a solution to your problem already posted.

18.1. Overview of Partitioning in MySQL

This section provides a conceptual overview of partitioning in MySQL 5.5.

For information on partitioning restrictions and feature limitations, see [Section 18.5, “Restrictions and Limitations on Partitioning”](#).

The SQL standard does not provide much in the way of guidance regarding the physical aspects of data storage. The SQL language itself is intended to work independently of any data structures or media underlying the schemas, tables, rows, or columns with which it works. Nonetheless, most advanced database management systems have evolved some means of determining the physical location to be used for storing specific pieces of data in terms of the file system, hardware or even both. In MySQL, the [InnoDB](#) storage engine has long supported the notion of a tablespace, and the MySQL Server, even prior to the introduction of partitioning, could be configured to employ different physical directories for storing different databases (see [Section 8.11.3.1, “Using Symbolic Links”](#), for an explanation of how this is done).

Partitioning takes this notion a step further, by enabling you to distribute portions of individual tables across a file system according to rules which you can set largely as needed. In effect, different portions of a table are stored as separate tables in different locations. The user-selected rule by which the division of data is accomplished is known as a *partitioning function*, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function. The function is selected according to the partitioning type specified by the user, and takes as its parameter the value of a user-supplied expression. This expression can be a column value, a function acting on one or more column values, or a set of one or more column values, depending on the type of partitioning that is used.

In the case of [RANGE](#), [LIST](#), and [\[LINEAR\] HASH](#) partitioning, the value of the partitioning column is passed to the partitioning function, which returns an integer value representing the number of the partition in which that particular record should be stored. This function must be nonconstant and nonrandom. It may not contain any queries, but may use an SQL expression that is valid in MySQL, as long as that expression returns either [NULL](#) or an integer *intval* such that

```
-MAXVALUE <= intval <= MAXVALUE
```

([MAXVALUE](#) is used to represent the least upper bound for the type of integer in question. [-MAXVALUE](#) represents the greatest lower bound.)

For [\[LINEAR\] KEY](#), [RANGE COLUMNS](#), and [LIST COLUMNS](#) partitioning, the partitioning expression consists of a list of one or more columns.

For [\[LINEAR\] KEY](#) partitioning, the partitioning function is supplied by MySQL.

For more information about permitted partitioning column types and partitioning functions, see [Section 18.2, “Partitioning Types”](#), as well as [Section 13.1.17, “CREATE TABLE Syntax”](#), which provides partitioning syntax descriptions and additional examples. For information about restrictions on partitioning functions, see [Section 18.5.3, “Partitioning Limitations Relating to Functions”](#).

This is known as *horizontal partitioning*—that is, different rows of a table may be assigned to different physical partitions. MySQL 5.5 does not support *vertical partitioning*, in which different columns of a table are assigned to different physical partitions. There are not at this time any plans to introduce vertical partitioning into MySQL 5.5.

For information about determining whether your MySQL Server binary supports user-defined partitioning, see [Chapter 18, Partitioning](#).

For creating partitioned tables, you can use most storage engines that are supported by your MySQL server; the MySQL partitioning engine runs in a separate layer and can interact with any of these. In MySQL 5.5, all partitions of the same partitioned table must use the same storage engine; for example, you cannot use [MyISAM](#) for one partition and [InnoDB](#) for another. However, there is nothing preventing you from using different storage engines for different partitioned tables on the same MySQL server or even in the same database.

MySQL partitioning cannot be used with the [MERGE](#), [CSV](#), or [FEDERATED](#) storage engines.

Partitioning by [KEY](#) or [LINEAR KEY](#) is possible with [NDBCLUSTER](#), but other types of user-defined partitioning are not supported for tables using this storage engine. In addition, an [NDBCLUSTER](#) table that employs user-defined partitioning must have an explicit primary key, and any columns referenced in the table's partitioning expression must be part of the primary key. However, if no columns are listed in the [PARTITION BY KEY](#) or [PARTITION BY LINEAR KEY](#) clause of the [CREATE TABLE](#) or [ALTER TABLE](#) statement used to create or modify a user-partitioned [NDBCLUSTER](#) table, then the table is not required to have an explicit primary key. For more information, see [Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”](#).

To employ a particular storage engine for a partitioned table, it is necessary only to use the [\[STORAGE\] ENGINE](#) option just as you would for a nonpartitioned table. However, you should keep in mind that [\[STORAGE\] ENGINE](#) (and other table options) need to be listed *before* any partitioning options are used in a [CREATE TABLE](#) statement. This example shows how to create a table that is partitioned by hash into 6 partitions and which uses the [InnoDB](#) storage engine:

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
ENGINE=INNODB
PARTITION BY HASH( MONTH(tr_date) )
PARTITIONS 6;
```

Each [PARTITION](#) clause can include a [\[STORAGE\] ENGINE](#) option, but in MySQL 5.5 this has no effect.



Important

Partitioning applies to all data and indexes of a table; you cannot partition only the data and not the indexes, or *vice versa*, nor can you partition only a portion of the table.

Data and indexes for each partition can be assigned to a specific directory using the [DATA DIRECTORY](#) and [INDEX DIRECTORY](#) options for the [PARTITION](#) clause of the [CREATE TABLE](#) statement used to create the partitioned table.

The [DATA DIRECTORY](#) and [INDEX DIRECTORY](#) options have no effect when defining partitions for tables using the [InnoDB](#) storage engine.

[DATA DIRECTORY](#) and [INDEX DIRECTORY](#) are not supported for individual partitions or subpartitions on Windows. These options are ignored on Windows, except that a warning is generated.

In addition, [MAX_ROWS](#) and [MIN_ROWS](#) can be used to determine the maximum and minimum numbers of rows, respectively, that can be stored in each partition. The [MAX_ROWS](#) option can be useful for causing MySQL Cluster tables to be created with extra partitions, thus allowing for greater storage of hash indexes.

See the documentation for the [DataMemory \[2043\]](#) data node configuration parameter, as well as [Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#), for more information.

Some advantages of partitioning are listed here:

- Partitioning makes it possible to store more data in one table than can be held on a single disk or file system partition.
- Data that loses its usefulness can often be easily removed from a partitioned table by dropping the partition (or partitions) containing only that data. Conversely, the process of adding new data can in some cases be greatly facilitated by adding one or more new partitions for storing specifically that data.
- Some queries can be greatly optimized in virtue of the fact that data satisfying a given `WHERE` clause can be stored only on one or more partitions, which automatically excluding any remaining partitions from the search. Because partitions can be altered after a partitioned table has been created, you can reorganize your data to enhance frequent queries that may not have been often used when the partitioning scheme was first set up. This ability to exclude non-matching partitions (and thus any rows they contain) is often referred to as *partition pruning*. For more information, see [Section 18.4, “Partition Pruning”](#).

Other benefits usually associated with partitioning include those in the following list. These features are not currently implemented in MySQL Partitioning, but are high on our list of priorities.

- Queries involving aggregate functions such as `SUM()` [\[1206\]](#) and `COUNT()` [\[1204\]](#) can easily be parallelized. A simple example of such a query might be `SELECT salesperson_id, COUNT(orders) as order_total FROM sales GROUP BY salesperson_id;` By “parallelized,” we mean that the query can be run simultaneously on each partition, and the final result obtained merely by summing the results obtained for all partitions.
- Achieving greater query throughput in virtue of spreading data seeks over multiple disks.

Be sure to check this section and chapter frequently for updates as MySQL Partitioning development continues.

18.2. Partitioning Types

This section discusses the types of partitioning which are available in MySQL 5.5. These include the types listed here:

- **RANGE partitioning.** This type of partitioning assigns rows to partitions based on column values falling within a given range. See [Section 18.2.1, “RANGE Partitioning”](#). MySQL 5.5 adds an extension, `RANGE COLUMNS`, to this type. See [Section 18.2.3.1, “RANGE COLUMNS partitioning”](#).
- **LIST partitioning.** Similar to partitioning by `RANGE`, except that the partition is selected based on columns matching one of a set of discrete values. See [Section 18.2.2, “LIST Partitioning”](#). MySQL 5.5 adds an extension, `LIST COLUMNS`, to this type. See [Section 18.2.3.2, “LIST COLUMNS partitioning”](#).
- **HASH partitioning.** With this type of partitioning, a partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table. The function may consist of any expression valid in MySQL that yields a nonnegative integer value. An extension to this type, `LINEAR HASH`, is also available. See [Section 18.2.4, “HASH Partitioning”](#).
- **KEY partitioning.** This type of partitioning is similar to partitioning by `HASH`, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function. These columns can contain other than integer values, since the hashing function supplied by MySQL guarantees an integer result regardless of the column data type. An extension to this type, `LINEAR KEY`, is also available. See [Section 18.2.5, “KEY Partitioning”](#).

A very common use of database partitioning is to segregate data by date. Some database systems support explicit date partitioning, which MySQL does not implement in 5.5. However, it is not difficult in MySQL to create partitioning schemes based on [DATE](#), [TIME](#), or [DATETIME](#) columns, or based on expressions making use of such columns.

When partitioning by [KEY](#) or [LINEAR KEY](#), you can use a [DATE](#), [TIME](#), or [DATETIME](#) column as the partitioning column without performing any modification of the column value. For example, this table creation statement is perfectly valid in MySQL:

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY KEY(joined)
PARTITIONS 6;
```

In MySQL 5.5, it is also possible to use a [DATE](#) or [DATETIME](#) column as the partitioning column using [RANGE COLUMNS](#) and [LIST COLUMNS](#) partitioning.

MySQL's other partitioning types, however, require a partitioning expression that yields an integer value or [NULL](#). If you wish to use date-based partitioning by [RANGE](#), [LIST](#), [HASH](#), or [LINEAR HASH](#), you can simply employ a function that operates on a [DATE](#), [TIME](#), or [DATETIME](#) column and returns such a value, as shown here:

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
  PARTITION p0 VALUES LESS THAN (1960),
  PARTITION p1 VALUES LESS THAN (1970),
  PARTITION p2 VALUES LESS THAN (1980),
  PARTITION p3 VALUES LESS THAN (1990),
  PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Additional examples of partitioning using dates may be found in the following sections of this chapter:

- [Section 18.2.1](#), “[RANGE Partitioning](#)”
- [Section 18.2.4](#), “[HASH Partitioning](#)”
- [Section 18.2.4.1](#), “[LINEAR HASH Partitioning](#)”

For more complex examples of date-based partitioning, see the following sections:

- [Section 18.4](#), “[Partition Pruning](#)”
- [Section 18.2.6](#), “[Subpartitioning](#)”

MySQL partitioning is optimized for use with the [TO_DAYS\(\)](#) [1148], [YEAR\(\)](#) [1152], and [TO_SECONDS\(\)](#) [1149] functions. However, you can use other date and time functions that return an integer or [NULL](#), such as [WEEKDAY\(\)](#) [1152], [DAYOFYEAR\(\)](#) [1140], or [MONTH\(\)](#) [1143]. See [Section 12.7](#), “[Date and Time Functions](#)”, for more information about such functions.

It is important to remember—regardless of the type of partitioning that you use—that partitions are always numbered automatically and in sequence when created, starting with 0. When a new row is inserted into a partitioned table, it is these partition numbers that are used in identifying the correct partition. For example, if your table uses 4 partitions, these partitions are numbered 0, 1, 2, and 3. For the [RANGE](#) and [LIST](#) partitioning types, it is necessary to ensure that there is a partition defined for each partition number. For [HASH](#) partitioning, the user function employed must return an integer value greater than 0. For [KEY](#) partitioning, this issue is taken care of automatically by the hashing function which the MySQL server employs internally.

Names of partitions generally follow the rules governing other MySQL identifiers, such as those for tables and databases. However, you should note that partition names are not case-sensitive. For example, the following [CREATE TABLE](#) statement fails as shown:

```
mysql> CREATE TABLE t2 (val INT)
-> PARTITION BY LIST(val)(
->     PARTITION mypart VALUES IN (1,3,5),
->     PARTITION MyPart VALUES IN (2,4,6)
-> );
ERROR 1488 (HY000): Duplicate partition name mypart
```

Failure occurs because MySQL sees no difference between the partition names `mypart` and `MyPart`.

When you specify the number of partitions for the table, this must be expressed as a positive, nonzero integer literal with no leading zeros, and may not be an expression such as `0.8E+01` or `6-2`, even if it evaluates to an integer value. Decimal fractions are not permitted.

In the sections that follow, we do not necessarily provide all possible forms for the syntax that can be used for creating each partition type; this information may be found in [Section 13.1.17](#), “[CREATE TABLE Syntax](#)”.

18.2.1. RANGE Partitioning

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range. Ranges should be contiguous but not overlapping, and are defined using the [VALUES LESS THAN](#) operator. For the next few examples, suppose that you are creating a table such as the following to hold personnel records for a chain of 20 video stores, numbered 1 through 20:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
);
```

This table can be partitioned by range in a number of ways, depending on your needs. One way would be to use the `store_id` column. For instance, you might decide to partition the table 4 ways by adding a [PARTITION BY RANGE](#) clause as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
```

```

PARTITION BY RANGE (store_id) (
  PARTITION p0 VALUES LESS THAN (6),
  PARTITION p1 VALUES LESS THAN (11),
  PARTITION p2 VALUES LESS THAN (16),
  PARTITION p3 VALUES LESS THAN (21)
);

```

In this partitioning scheme, all rows corresponding to employees working at stores 1 through 5 are stored in partition `p0`, to those employed at stores 6 through 10 are stored in partition `p1`, and so on. Note that each partition is defined in order, from lowest to highest. This is a requirement of the `PARTITION BY RANGE` syntax; you can think of it as being analogous to a series of `if ... elseif ...` statements in C or Java in this regard.

It is easy to determine that a new row containing the data `(72, 'Michael', 'Widenius', '1998-06-25', NULL, 13)` is inserted into partition `p2`, but what happens when your chain adds a 21st store? Under this scheme, there is no rule that covers a row whose `store_id` is greater than 20, so an error results because the server does not know where to place it. You can keep this from occurring by using a “catchall” `VALUES LESS THAN` clause in the `CREATE TABLE` statement that provides for all values greater than the highest value explicitly named:

```

CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
  PARTITION p0 VALUES LESS THAN (6),
  PARTITION p1 VALUES LESS THAN (11),
  PARTITION p2 VALUES LESS THAN (16),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);

```



Note

Another way to avoid an error when no matching value is found is to use the `IGNORE` keyword as part of the `INSERT` statement. For an example, see [Section 18.2.2, “LIST Partitioning”](#). Also see [Section 13.2.5, “INSERT Syntax”](#), for general information about `IGNORE`.

`MAXVALUE` represents an integer value that is always greater than the largest possible integer value (in mathematical language, it serves as a *least upper bound*). Now, any rows whose `store_id` column value is greater than or equal to 16 (the highest value defined) are stored in partition `p3`. At some point in the future—when the number of stores has increased to 25, 30, or more—you can use an `ALTER TABLE` statement to add new partitions for stores 21-25, 26-30, and so on (see [Section 18.3, “Partition Management”](#), for details of how to do this).

In much the same fashion, you could partition the table based on employee job codes—that is, based on ranges of `job_code` column values. For example—assuming that two-digit job codes are used for regular (in-store) workers, three-digit codes are used for office and support personnel, and four-digit codes are used for management positions—you could create the partitioned table using the following statement:

```

CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',

```

```

    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
    PARTITION p0 VALUES LESS THAN (100),
    PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (10000)
);

```

In this instance, all rows relating to in-store workers would be stored in partition `p0`, those relating to office and support staff in `p1`, and those relating to managers in partition `p2`.

It is also possible to use an expression in `VALUES LESS THAN` clauses. However, MySQL must be able to evaluate the expression's return value as part of a `LESS THAN (<)` comparison.

Rather than splitting up the table data according to store number, you can use an expression based on one of the two `DATE` columns instead. For example, let us suppose that you wish to partition based on the year that each employee left the company; that is, the value of `YEAR(separated)` [1152]. An example of a `CREATE TABLE` statement that implements such a partitioning scheme is shown here:

```

CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY RANGE ( YEAR(separated) ) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1996),
    PARTITION p2 VALUES LESS THAN (2001),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);

```

In this scheme, for all employees who left before 1991, the rows are stored in partition `p0`; for those who left in the years 1991 through 1995, in `p1`; for those who left in the years 1996 through 2000, in `p2`; and for any workers who left after the year 2000, in `p3`.

It is also possible to partition a table by `RANGE`, based on the value of a `TIMESTAMP` column, using the `UNIX_TIMESTAMP()` [1150] function, as shown in this example:

```

CREATE TABLE quarterly_report_status (
    report_id INT NOT NULL,
    report_status VARCHAR(20) NOT NULL,
    report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
    PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
    PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
    PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
    PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
    PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
    PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
    PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
    PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
    PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
    PARTITION p9 VALUES LESS THAN (MAXVALUE)
);

```

Any other expressions involving `TIMESTAMP` values are not permitted. (See Bug #42849.)

Range partitioning is particularly useful when one or more of the following conditions is true:

- You want or need to delete “old” data. If you are using the partitioning scheme shown immediately above, you can simply use `ALTER TABLE employees DROP PARTITION p0;` to delete all rows relating to employees who stopped working for the firm prior to 1991. (See [Section 13.1.7, “ALTER TABLE Syntax”](#), and [Section 18.3, “Partition Management”](#), for more information.) For a table with a great many rows, this can be much more efficient than running a `DELETE` query such as `DELETE FROM employees WHERE YEAR(separated) <= 1990;`.
- You want to use a column containing date or time values, or containing values arising from some other series.
- You frequently run queries that depend directly on the column used for partitioning the table. For example, when executing a query such as `EXPLAIN PARTITIONS SELECT COUNT(*) FROM employees WHERE separated BETWEEN '2000-01-01' AND '2000-12-31' GROUP BY store_id;`, MySQL can quickly determine that only partition `p2` needs to be scanned because the remaining partitions cannot contain any records satisfying the `WHERE` clause. See [Section 18.4, “Partition Pruning”](#), for more information about how this is accomplished.

A variant on this type of partitioning, [RANGE COLUMNS](#) partitioning, was introduced in MySQL 5.5.0. Partitioning by [RANGE COLUMNS](#) makes it possible to employ multiple columns for defining partitioning ranges that apply both to placement of rows in partitions and for determining the inclusion or exclusion of specific partitions when performing partition pruning. See [Section 18.2.3.1, “RANGE COLUMNS partitioning”](#), for more information.

Partitioning schemes based on time intervals. If you wish to implement a partitioning scheme based on ranges or intervals of time in MySQL 5.5, you have two options:

1. Partition the table by [RANGE](#), and for the partitioning expression, employ a function operating on a [DATE](#), [TIME](#), or [DATETIME](#) column and returning an integer value, as shown here:

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
  PARTITION p0 VALUES LESS THAN (1960),
  PARTITION p1 VALUES LESS THAN (1970),
  PARTITION p2 VALUES LESS THAN (1980),
  PARTITION p3 VALUES LESS THAN (1990),
  PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Beginning with MySQL 5.5.1, it is also possible to partition a table by [RANGE](#) based on the value of a [TIMESTAMP](#) column, using the `UNIX_TIMESTAMP () [1150]` function, as shown in this example:

```
CREATE TABLE quarterly_report_status (
  report_id INT NOT NULL,
  report_status VARCHAR(20) NOT NULL,
  report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
  PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
  PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
  PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
  PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
  PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
  PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
  PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
  PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
```

```

PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
PARTITION p9 VALUES LESS THAN (MAXVALUE)
);

```

Also beginning with MySQL 5.5.1, any other expressions involving `TIMESTAMP` values are not permitted. (See Bug #42849.)



Note

It is also possible in MySQL 5.5.1 and later to use `UNIX_TIMESTAMP(timestamp_column)` [1150] as a partitioning expression for tables that are partitioned by `LIST`. However, it is usually not practical to do so.

- Partition the table by `RANGE COLUMNS`, using a `DATE` or `DATETIME` column as the partitioning column. For example, the `members` table could be defined using the `joined` column directly, as shown here:

```

CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY RANGE COLUMNS(joined) (
  PARTITION p0 VALUES LESS THAN ('1960-01-01'),
  PARTITION p1 VALUES LESS THAN ('1970-01-01'),
  PARTITION p2 VALUES LESS THAN ('1980-01-01'),
  PARTITION p3 VALUES LESS THAN ('1990-01-01'),
  PARTITION p4 VALUES LESS THAN MAXVALUE
);

```



Note

The use of partitioning columns employing date or time types other than `DATE` or `DATETIME` is not supported with `RANGE COLUMNS`.

18.2.2. LIST Partitioning

List partitioning in MySQL is similar to range partitioning in many ways. As in partitioning by `RANGE`, each partition must be explicitly defined. The chief difference between the two types of partitioning is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values. This is done by using `PARTITION BY LIST(expr)` where `expr` is a column value or an expression based on a column value and returning an integer value, and then defining each partition by means of a `VALUES IN (value_list)`, where `value_list` is a comma-separated list of integers.



Note

In MySQL 5.5, it is possible to match against only a list of integers (and possibly `NULL`—see Section 18.2.7, “How MySQL Partitioning Handles `NULL`”) when partitioning by `LIST`.

However, beginning with MySQL 5.5.0, other column types may be used in value lists when employing `LIST COLUMN` partitioning, which is described later in this section.

Unlike the case with partitions defined by range, list partitions do not need to be declared in any particular order. For more detailed syntactical information, see Section 13.1.17, “`CREATE TABLE Syntax`”.

For the examples that follow, we assume that the basic definition of the table to be partitioned is provided by the `CREATE TABLE` statement shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
);
```

(This is the same table used as a basis for the examples in [Section 18.2.1, “RANGE Partitioning”](#).)

Suppose that there are 20 video stores distributed among 4 franchises as shown in the following table.

Region	Store ID Numbers
North	3, 5, 6, 9, 17
East	1, 2, 10, 11, 19, 20
West	4, 12, 13, 14, 18
Central	7, 8, 15, 16

To partition this table in such a way that rows for stores belonging to the same region are stored in the same partition, you could use the `CREATE TABLE` statement shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY LIST(store_id) (
  PARTITION pNorth VALUES IN (3,5,6,9,17),
  PARTITION pEast VALUES IN (1,2,10,11,19,20),
  PARTITION pWest VALUES IN (4,12,13,14,18),
  PARTITION pCentral VALUES IN (7,8,15,16)
);
```

This makes it easy to add or drop employee records relating to specific regions to or from the table. For instance, suppose that all stores in the West region are sold to another company. Beginning with MySQL 5.5.0, all rows relating to employees working at stores in that region can be deleted with the query `ALTER TABLE employees TRUNCATE PARTITION pWest`, which can be executed much more efficiently than the equivalent `DELETE` statement `DELETE FROM employees WHERE store_id IN (4,12,13,14,18);` (Using `ALTER TABLE employees DROP PARTITION pWest` would also delete all of these rows, but would also remove the partition `pWest` from the definition of the table; you would need to use an `ALTER TABLE ... ADD PARTITION` statement to restore the table's original partitioning scheme.)

As with [RANGE](#) partitioning, it is possible to combine [LIST](#) partitioning with partitioning by hash or key to produce a composite partitioning (subpartitioning). See [Section 18.2.6, “Subpartitioning”](#).

Unlike the case with [RANGE](#) partitioning, there is no “catch-all” such as `MAXVALUE`; all expected values for the partitioning expression should be covered in `PARTITION ... VALUES IN (...)` clauses. An `INSERT` statement containing an unmatched partitioning column value fails with an error, as shown in this example:

```
mysql> CREATE TABLE h2 (
->   c1 INT,
->   c2 INT
-> )
-> PARTITION BY LIST(c1) (
->   PARTITION p0 VALUES IN (1, 4, 7),
->   PARTITION p1 VALUES IN (2, 5, 8)
-> );
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO h2 VALUES (3, 5);
ERROR 1525 (HY000): Table has no partition for value 3
```

When inserting multiple rows using a single `INSERT` statement, any rows coming before the row containing the unmatched value are inserted, but any coming after it are not:

```
mysql> SELECT * FROM h2;
Empty set (0.00 sec)

mysql> INSERT INTO h2 VALUES (4, 7), (3, 5), (6, 0);
ERROR 1525 (HY000): Table has no partition for value 3
mysql> SELECT * FROM h2;
+-----+-----+
| c1   | c2   |
+-----+-----+
|    4 |    7 |
+-----+-----+
1 row in set (0.00 sec)
```

You can cause this type of error to be ignored by using the `IGNORE` keyword. If you do so, rows containing unmatched partitioning column values are not inserted, but any rows with matching values are inserted, and no errors are reported:

```
mysql> TRUNCATE h2;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM h2;
Empty set (0.00 sec)

mysql> INSERT IGNORE INTO h2 VALUES (2, 5), (6, 10), (7, 5), (3, 1), (1, 9);
Query OK, 3 rows affected (0.00 sec)
Records: 5  Duplicates: 2  Warnings: 0

mysql> SELECT * FROM h2;
+-----+-----+
| c1   | c2   |
+-----+-----+
|    7 |    5 |
|    1 |    9 |
|    2 |    5 |
+-----+-----+
3 rows in set (0.00 sec)
```

MySQL 5.5 provides support for `LIST COLUMNS` partitioning. This is a variant of `LIST` partitioning that enables you to use columns of types other than integer types for partitioning columns, as well as to use multiple columns as partitioning keys. For more information, see [Section 18.2.3.2, “LIST COLUMNS partitioning”](#).

18.2.3. COLUMNS Partitioning

The next two sections discuss *COLUMNS partitioning*, which are variants on `RANGE` and `LIST` partitioning that were introduced in MySQL 5.5.0. `COLUMNS` partitioning enables the use of multiple columns in partitioning keys. All of these columns are taken into account both for the purpose of placing rows in

partitions and for the determination of which partitions are to be checked for matching rows in partition pruning.

In addition, both [RANGE COLUMNS](#) partitioning and [LIST COLUMNS](#) partitioning support the use of non-integer columns for defining value ranges or list members. The permitted data types are shown in the following list:

- All integer types: [TINYINT](#), [SMALLINT](#), [MEDIUMINT](#), [INT \(INTEGER\)](#), and [BIGINT](#). (This is the same as with partitioning by [RANGE](#) and [LIST](#).)

Other numeric data types (such as [DECIMAL](#) or [FLOAT](#)) are not supported as partitioning columns.

- [DATE](#) and [DATETIME](#).

Columns using other data types relating to dates or times are not supported as partitioning columns.

- The following string types: [CHAR](#), [VARCHAR](#), [BINARY](#), and [VARBINARY](#).

[TEXT](#) and [BLOB](#) columns are not supported as partitioning columns.

The discussions of [RANGE COLUMNS](#) and [LIST COLUMNS](#) partitioning in the next two sections assume that you are already familiar with partitioning based on ranges and lists as supported in MySQL 5.1 and later; for more information about these, see [Section 18.2.1, “RANGE Partitioning”](#), and [Section 18.2.2, “LIST Partitioning”](#), respectively.

18.2.3.1. RANGE COLUMNS partitioning

Range columns partitioning is similar to range partitioning, but enables you to define partitions using ranges based on multiple column values. In addition, you can define the ranges using columns of types other than integer types.

[RANGE COLUMNS](#) partitioning differs significantly from [RANGE](#) partitioning in the following ways:

- [RANGE COLUMNS](#) does not accept expressions, only names of columns.
- [RANGE COLUMNS](#) accepts a list of one or more columns.

[RANGE COLUMNS](#) partitions are based on comparisons between *tuples* (lists of column values) rather than comparisons between scalar values. Placement of rows in [RANGE COLUMNS](#) partitions is also based on comparisons between tuples; this is discussed further later in this section.

- [RANGE COLUMNS](#) partitioning columns are not restricted to integer columns; string, [DATE](#) and [DATETIME](#) columns can also be used as partitioning columns. (See [Section 18.2.3, “COLUMNS Partitioning”](#), for details.)

The basic syntax for creating a table partitioned by [RANGE COLUMNS](#) is shown here:

```
CREATE TABLE table_name
PARTITIONED BY RANGE COLUMNS(column_list) (
  PARTITION partition_name VALUES LESS THAN (value_list)[,
  PARTITION partition_name VALUES LESS THAN (value_list)[,
  ...]
)

column_list:
  column_name[, column_name][, ...]

value_list:
  value[, value][, ...]
```

**Note**

Not all `CREATE TABLE` options that can be used when creating partitioned tables are shown here. For complete information, see [Section 13.1.17, “CREATE TABLE Syntax”](#).

In the syntax just shown, *column_list* is a list of one or more columns (sometimes called a *partitioning column list*), and *value_list* is a list of values (that is, it is a *partition definition value list*). A *value_list* must be supplied for each partition definition, and each *value_list* must have the same number of values as the *column_list* has columns. Generally speaking, if you use *N* columns in the `COLUMNS` clause, then each `VALUES LESS THAN` clause must also be supplied with a list of *N* values.

The elements in the partitioning column list and in the value list defining each partition must occur in the same order. In addition, each element in the value list must be of the same data type as the corresponding element in the column list. However, the order of the column names in the partitioning column list and the value lists does not have to be the same as the order of the table column definitions in the main part of the `CREATE TABLE` statement. As with table partitioned by `RANGE`, you can use `MAXVALUE` to represent a value such that any legal value inserted into a given column is always less than this value. Here is an example of a `CREATE TABLE` statement that helps to illustrate all of these points:

```
mysql> CREATE TABLE rcx (
->   a INT,
->   b INT,
->   c CHAR(3),
->   d INT
-> )
-> PARTITION BY RANGE COLUMNS(a,d,c) (
->   PARTITION p0 VALUES LESS THAN (5,10,'ggg'),
->   PARTITION p1 VALUES LESS THAN (10,20,'mmmm'),
->   PARTITION p2 VALUES LESS THAN (15,30,'sss'),
->   PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
-> );
Query OK, 0 rows affected (0.15 sec)
```

Table `rcx` contains the columns `a`, `b`, `c`, `d`. The partitioning column list supplied to the `COLUMNS` clause uses 3 of these columns, in the order `a`, `d`, `c`. Each value list used to define a partition contains 3 values in the same order; that is, each value list tuple has the form `(INT, INT, CHAR(3))`, which corresponds to the data types used by columns `a`, `d`, and `c` (in that order).

Placement of rows into partitions is determined by comparing the tuple from a row to be inserted that matches the column list in the `COLUMNS` clause with the tuples used in the `VALUES LESS THAN` clauses to define partitions of the table. Because we are comparing tuples (that is, lists or sets of values) rather than scalar values, the semantics of `VALUES LESS THAN` as used with `RANGE COLUMNS` partitions differs somewhat from the case with simple `RANGE` partitions. In `RANGE` partitioning, a row generating an expression value that is equal to a limiting value in a `VALUES LESS THAN` is never placed in the corresponding partition; however, when using `RANGE COLUMNS` partitioning, it is sometimes possible for a row whose partitioning column list's first element is equal in value to the that of the first element in a `VALUES LESS THAN` value list to be placed in the corresponding partition.

Consider the `RANGE` partitioned table created by this statement:

```
CREATE TABLE r1 (
  a INT,
  b INT
)
PARTITION BY RANGE (a) (
  PARTITION p0 VALUES LESS THAN (5),
  PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

If we insert 3 rows into this table such that the column value for `a` is 5 for each row, all 3 rows are stored in partition `p1` because the `a` column value is in each case not less than 5, as we can see by executing the proper query against the `INFORMATION_SCHEMA.PARTITIONS` table:

```
mysql> INSERT INTO r1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT PARTITION_NAME, TABLE_ROWS
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'r1';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0              |            0 |
| p1              |            3 |
+-----+-----+
2 rows in set (0.00 sec)
```

Now consider a similar table `rc1` that uses `RANGE COLUMNS` partitioning with both columns `a` and `b` referenced in the `COLUMNS` clause, created as shown here:

```
CREATE TABLE rc1 (
  a INT,
  b INT
)
PARTITION BY RANGE COLUMNS(a, b) (
  PARTITION p0 VALUES LESS THAN (5, 12),
  PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE)
);
```

If we insert exactly the same rows into `rc1` as we just inserted into `r1`, the distribution of the rows is quite different:

```
mysql> INSERT INTO rc1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT PARTITION_NAME, TABLE_ROWS
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'rc1';
+-----+-----+-----+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+-----+-----+-----+
| p              | p0              |            2 |
| p              | p1              |            1 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

This is because we are comparing rows rather than scalar values. We can compare the row values inserted with the limiting row value from the `VALUES THAN LESS THAN` clause used to define partition `p0` in table `rc1`, like this:

```
mysql> SELECT (5,10) < (5,12), (5,11) < (5,12), (5,12) < (5,12);
+-----+-----+-----+
| (5,10) < (5,12) | (5,11) < (5,12) | (5,12) < (5,12) |
+-----+-----+-----+
| 1                | 1                | 0                |
+-----+-----+-----+
1 row in set (0.00 sec)
```

The 2 tuples `(5,10)` and `(5,11)` evaluate as less than `(5,12)`, so they are stored in partition `p0`. Since 5 is not less than 5 and 12 is not less than 12, `(5,12)` is considered not less than `(5,12)`, and is stored in partition `p1`.

The `SELECT` statement in the preceding example could also have been written using explicit row constructors, like this:

```
SELECT ROW(5,10) < ROW(5,12), ROW(5,11) < ROW(5,12), ROW(5,12) < ROW(5,12);
```

For more information about the use of row constructors in MySQL, see [Section 13.2.10.5, “Row Subqueries”](#).

For a table partitioned by `RANGE COLUMNS` using only a single partitioning column, the storing of rows in partitions is the same as that of an equivalent table that is partitioned by `RANGE`. The following `CREATE TABLE` statement creates a table partitioned by `RANGE COLUMNS` using 1 partitioning column:

```
CREATE TABLE rx (
  a INT,
  b INT
)
PARTITION BY RANGE COLUMNS (a) (
  PARTITION p0 VALUES LESS THAN (5),
  PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

If we insert the rows `(5,10)`, `(5,11)`, and `(5,12)` into this table, we can see that their placement is the same as it is for the table `r` we created and populated earlier:

```
mysql> INSERT INTO rx VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'rx';
```

TABLE_SCHEMA	PARTITION_NAME	TABLE_ROWS
p	p0	0
p	p1	3

```
2 rows in set (0.00 sec)
```

It is also possible to create tables partitioned by `RANGE COLUMNS` where limiting values for one or more columns are repeated in successive partition definitions. You can do this as long as the tuples of column values used to define the partitions are strictly increasing. For example, each of the following `CREATE TABLE` statements is valid:

```
CREATE TABLE rc2 (
  a INT,
  b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
  PARTITION p0 VALUES LESS THAN (0,10),
  PARTITION p1 VALUES LESS THAN (10,20),
  PARTITION p2 VALUES LESS THAN (10,30),
  PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)
);
```

```
CREATE TABLE rc3 (
  a INT,
  b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
  PARTITION p0 VALUES LESS THAN (0,10),
  PARTITION p1 VALUES LESS THAN (10,20),
  PARTITION p2 VALUES LESS THAN (10,30),
  PARTITION p3 VALUES LESS THAN (10,35),
  PARTITION p4 VALUES LESS THAN (20,40),
```

```

PARTITION p5 VALUES LESS THAN (MAXVALUE,MAXVALUE)
);

```

The following statement also succeeds, even though it might appear at first glance that it would not, since the limiting value of column `b` is 25 for partition `p0` and 20 for partition `p1`, and the limiting value of column `c` is 100 for partition `p1` and 50 for partition `p2`:

```

CREATE TABLE rc4 (
  a INT,
  b INT,
  c INT
)
PARTITION BY RANGE COLUMNS(a,b,c) (
  PARTITION p0 VALUES LESS THAN (0,25,50),
  PARTITION p1 VALUES LESS THAN (10,20,100),
  PARTITION p2 VALUES LESS THAN (10,30,50)
  PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
);

```

When designing tables partitioned by `RANGE COLUMNS`, you can always test successive partition definitions by comparing the desired tuples using the `mysql` client, like this:

```

mysql> SELECT (0,25,50) < (10,20,100), (10,20,100) < (10,30,50);
+-----+-----+
| (0,25,50) < (10,20,100) | (10,20,100) < (10,30,50) |
+-----+-----+
| 1 | 1 |
+-----+-----+
1 row in set (0.00 sec)

```

If a `CREATE TABLE` statement contains partition definitions that are not in strictly increasing order, it fails with an error, as shown in this example:

```

mysql> CREATE TABLE rcf (
->   a INT,
->   b INT,
->   c INT
-> )
-> PARTITION BY RANGE COLUMNS(a,b,c) (
->   PARTITION p0 VALUES LESS THAN (0,25,50),
->   PARTITION p1 VALUES LESS THAN (20,20,100),
->   PARTITION p2 VALUES LESS THAN (10,30,50),
->   PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
-> );
ERROR 1493 (HY000): VALUES LESS THAN value must be strictly increasing for each partition

```

When you get such an error, you can deduce which partition definitions are invalid by making “less than” comparisons between their column lists. In this case, the problem is with the definition of partition `p2` because the tuple used to define it is not less than the tuple used to define partition `p3`, as shown here:

```

mysql> SELECT (0,25,50) < (20,20,100), (20,20,100) < (10,30,50);
+-----+-----+
| (0,25,50) < (20,20,100) | (20,20,100) < (10,30,50) |
+-----+-----+
| 1 | 0 |
+-----+-----+
1 row in set (0.00 sec)

```

It is also possible for `MAXVALUE` to appear for the same column in more than one `VALUES LESS THAN` clause when using `RANGE COLUMNS`. However, the limiting values for individual columns in successive partition definitions should otherwise be increasing, there should be no more than one partition defined where `MAXVALUE` is used as the upper limit for all column values, and this partition definition should appear last in the list of `PARTITION ... VALUES LESS THAN` clauses. In addition, you cannot use `MAXVALUE` as the limiting value for the first column in more than one partition definition.

As stated previously, it is also possible with `RANGE COLUMNS` partitioning to use non-integer columns as partitioning columns. (See [Section 18.2.3, “COLUMNS Partitioning”](#), for a complete listing of these.) Consider a table named `employees` (which is not partitioned), created using the following statement:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
);
```

Using `RANGE COLUMNS` partitioning, you can create a version of this table that stores each row in one of four partitions based on the employee's last name, like this:

```
CREATE TABLE employees_by_lname (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
PARTITION BY RANGE COLUMNS (lname) (
  PARTITION p0 VALUES LESS THAN ('g'),
  PARTITION p1 VALUES LESS THAN ('m'),
  PARTITION p2 VALUES LESS THAN ('t'),
  PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

Alternatively, you could cause the `employees` table as created previously to be partitioned using this scheme by executing the following `ALTER TABLE` statement:

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (lname) (
  PARTITION p0 VALUES LESS THAN ('g'),
  PARTITION p1 VALUES LESS THAN ('m'),
  PARTITION p2 VALUES LESS THAN ('t'),
  PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```



Note

Because different character sets and collations have different sort orders, the character sets and collations in use may effect which partition of a table partitioned by `RANGE COLUMNS` a given row is stored in when using string columns as partitioning columns. In addition, changing the character set or collation for a given database, table, or column after such a table is created may cause changes in how rows are distributed. For example, when using a case-sensitive collation, `'and'` sorts before `'Andersen'`, but when using a collation that is case insensitive, the reverse is true.

For information about how MySQL handles character sets and collations, see [Section 10.1, “Character Set Support”](#).

Similarly, you can cause the `employees` table to be partitioned in such a way that each row is stored in one of several partitions based on the decade in which the corresponding employee was hired using the `ALTER TABLE` statement shown here:

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (hired) (
```

```

PARTITION p0 VALUES LESS THAN ('1970-01-01'),
PARTITION p1 VALUES LESS THAN ('1980-01-01'),
PARTITION p2 VALUES LESS THAN ('1990-01-01'),
PARTITION p3 VALUES LESS THAN ('2000-01-01'),
PARTITION p4 VALUES LESS THAN ('2010-01-01'),
PARTITION p5 VALUES LESS THAN (MAXVALUE)
);

```

See [Section 13.1.17, “CREATE TABLE Syntax”](#), for additional information about `PARTITION BY RANGE COLUMNS` syntax.

18.2.3.2. LIST COLUMNS partitioning

MySQL 5.5 provides support for `LIST COLUMNS` partitioning. This is a variant of `LIST` partitioning that enables the use of multiple columns as partition keys, and for columns of data types other than integer types to be used as partitioning columns; you can use string types, `DATE`, and `DATETIME` columns. (For more information about permitted data types for `COLUMNS` partitioning columns, see [Section 18.2.3, “COLUMNS Partitioning”](#).)

Suppose that you have a business that has customers in 12 cities which, for sales and marketing purposes, you organize into 4 regions of 3 cities each as shown in the following table:

Region	Cities
1	Oskarshamn, Högsby, Mönsterås
2	Vimmerby, Hultsfred, Västervik
3	Nässjö, Eksjö, Vetlanda
4	Uppvidinge, Alvesta, Växjö

With `LIST COLUMNS` partitioning, you can create a table for customer data that assigns a row to any of 4 partitions corresponding to these regions based on the name of the city where a customer resides, as shown here:

```

CREATE TABLE customers_1 (
  first_name VARCHAR(25),
  last_name VARCHAR(25),
  street_1 VARCHAR(30),
  street_2 VARCHAR(30),
  city VARCHAR(15),
  renewal DATE
)
PARTITION BY LIST COLUMNS(city) (
  PARTITION pRegion_1 VALUES IN('Oskarshamn', 'Högsby', 'Mönsterås'),
  PARTITION pRegion_2 VALUES IN('Vimmerby', 'Hultsfred', 'Västervik'),
  PARTITION pRegion_3 VALUES IN('Nässjö', 'Eksjö', 'Vetlanda'),
  PARTITION pRegion_4 VALUES IN('Uppvidinge', 'Alvesta', 'Växjö')
);

```

As with partitioning by `RANGE COLUMNS`, you do not need to use expressions in the `COLUMNS()` clause to convert column values into integers. (In fact, the use of expressions other than column names is not permitted with `COLUMNS()`.)

It is also possible to use `DATE` and `DATETIME` columns, as shown in the following example that uses the same name and columns as the `customers_1` table shown previously, but employs `LIST COLUMNS` partitioning based on the `renewal` column to store rows in one of 4 partitions depending on the week in February 2010 the customer's account is scheduled to renew:

```

CREATE TABLE customers_2 (
  first_name VARCHAR(25),

```

```

    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY LIST COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES IN('2010-02-01', '2010-02-02', '2010-02-03',
    '2010-02-04', '2010-02-05', '2010-02-06', '2010-02-07'),
    PARTITION pWeek_2 VALUES IN('2010-02-08', '2010-02-09', '2010-02-10',
    '2010-02-11', '2010-02-12', '2010-02-13', '2010-02-14'),
    PARTITION pWeek_3 VALUES IN('2010-02-15', '2010-02-16', '2010-02-17',
    '2010-02-18', '2010-02-19', '2010-02-20', '2010-02-21'),
    PARTITION pWeek_4 VALUES IN('2010-02-22', '2010-02-23', '2010-02-24',
    '2010-02-25', '2010-02-26', '2010-02-27', '2010-02-28')
);

```

This works, but becomes cumbersome to define and maintain if the number of dates involved grows very large; in such cases, it is usually more practical to employ [RANGE](#) or [RANGE COLUMNS](#) partitioning instead. In this case, since the column we wish to use as the partitioning key is a [DATE](#) column, we use [RANGE COLUMNS](#) partitioning, as shown here:

```

CREATE TABLE customers_3 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY RANGE COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES LESS THAN('2010-02-09'),
    PARTITION pWeek_2 VALUES LESS THAN('2010-02-15'),
    PARTITION pWeek_3 VALUES LESS THAN('2010-02-22'),
    PARTITION pWeek_4 VALUES LESS THAN('2010-03-01')
);

```

See [Section 18.2.3.1, “RANGE COLUMNS partitioning”](#), for more information.

In addition (as with [RANGE COLUMNS](#) partitioning), you can use multiple columns in the [COLUMNS\(\)](#) clause.

See [Section 13.1.17, “CREATE TABLE Syntax”](#), for additional information about [PARTITION BY LIST COLUMNS\(\)](#) syntax.

18.2.4. HASH Partitioning

Partitioning by [HASH](#) is used primarily to ensure an even distribution of data among a predetermined number of partitions. With range or list partitioning, you must specify explicitly into which partition a given column value or set of column values is to be stored; with hash partitioning, MySQL takes care of this for you, and you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

To partition a table using [HASH](#) partitioning, it is necessary to append to the [CREATE TABLE](#) statement a [PARTITION BY HASH \(expr\)](#) clause, where *expr* is an expression that returns an integer. This can simply be the name of a column whose type is one of MySQL's integer types. In addition, you will most likely want to follow this with a [PARTITIONS num](#) clause, where *num* is a positive integer representing the number of partitions into which the table is to be divided.

For example, the following statement creates a table that uses hashing on the `store_id` column and is divided into 4 partitions:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

If you do not include a `PARTITIONS` clause, the number of partitions defaults to 1.

Using the `PARTITIONS` keyword without a number following it results in a syntax error.

You can also use an SQL expression that returns an integer for *expr*. For instance, you might want to partition based on the year in which an employee was hired. This can be done as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY HASH( YEAR(hired) )
PARTITIONS 4;
```

expr must return a nonconstant, nonrandom integer value (in other words, it should be varying but deterministic), and must not contain any prohibited constructs as described in [Section 18.5, “Restrictions and Limitations on Partitioning”](#). You should also keep in mind that this expression is evaluated each time a row is inserted or updated (or possibly deleted); this means that very complex expressions may give rise to performance issues, particularly when performing operations (such as batch inserts) that affect a great many rows at one time.

The most efficient hashing function is one which operates upon a single table column and whose value increases or decreases consistently with the column value, as this allows for “pruning” on ranges of partitions. That is, the more closely that the expression varies with the value of the column on which it is based, the more efficiently MySQL can use the expression for hash partitioning.

For example, where *date_col* is a column of type `DATE`, then the expression `TO_DAYS(date_col)` [1148] is said to vary directly with the value of *date_col*, because for every change in the value of *date_col*, the value of the expression changes in a consistent manner. The variance of the expression `YEAR(date_col)` [1152] with respect to *date_col* is not quite as direct as that of `TO_DAYS(date_col)` [1148], because not every possible change in *date_col* produces an equivalent change in `YEAR(date_col)` [1152]. Even so, `YEAR(date_col)` [1152] is a good candidate for a hashing function, because it varies directly with a portion of *date_col* and there is no possible change in *date_col* that produces a disproportionate change in `YEAR(date_col)` [1152].

By way of contrast, suppose that you have a column named *int_col* whose type is `INT`. Now consider the expression `POW(5-int_col,3) + 6` [1128]. This would be a poor choice for a hashing function because a change in the value of *int_col* is not guaranteed to produce a proportional change in the value of the expression. Changing the value of *int_col* by a given amount can produce by widely different changes in the value of the expression. For example, changing *int_col* from 5 to 6 produces a change of -1 in the value of the expression, but changing the value of *int_col* from 6 to 7 produces a change of -7 in the expression value.

In other words, the more closely the graph of the column value *versus* the value of the expression follows a straight line as traced by the equation $y=cx$ where c is some nonzero constant, the better the expression is suited to hashing. This has to do with the fact that the more nonlinear an expression is, the more uneven the distribution of data among the partitions it tends to produce.

In theory, pruning is also possible for expressions involving more than one column value, but determining which of such expressions are suitable can be quite difficult and time-consuming. For this reason, the use of hashing expressions involving multiple columns is not particularly recommended.

When `PARTITION BY HASH` is used, MySQL determines which partition of num partitions to use based on the modulus of the result of the user function. In other words, for an expression $expr$, the partition in which the record is stored is partition number N , where $N = \text{MOD}(expr, num)$. Suppose that table `t1` is defined as follows, so that it has 4 partitions:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY HASH( YEAR(col3) )
PARTITIONS 4;
```

If you insert a record into `t1` whose `col3` value is `'2005-09-15'`, then the partition in which it is stored is determined as follows:

```
MOD(YEAR('2005-09-01'),4)
= MOD(2005,4)
= 1
```

MySQL 5.5 also supports a variant of `HASH` partitioning known as *linear hashing* which employs a more complex algorithm for determining the placement of new rows inserted into the partitioned table. See [Section 18.2.4.1, “LINEAR HASH Partitioning”](#), for a description of this algorithm.

The user function is evaluated each time a record is inserted or updated. It may also—depending on the circumstances—be evaluated when records are deleted.



Note

If a table to be partitioned has a `UNIQUE` key, then any columns supplied as arguments to the `HASH` user function or to the `KEY`'s `column_list` must be part of that key.

18.2.4.1. LINEAR HASH Partitioning

MySQL also supports linear hashing, which differs from regular hashing in that linear hashing utilizes a linear powers-of-two algorithm whereas regular hashing employs the modulus of the hashing function's value.

Syntactically, the only difference between linear-hash partitioning and regular hashing is the addition of the `LINEAR` keyword in the `PARTITION BY` clause, as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY LINEAR HASH( YEAR(hired) )
PARTITIONS 4;
```

Given an expression *expr*, the partition in which the record is stored when linear hashing is used is partition number *N* from among *num* partitions, where *N* is derived according to the following algorithm:

1. Find the next power of 2 greater than *num*. We call this value *V*; it can be calculated as:

```
V = POWER(2, CEILING(LOG(2, num)))
```

(Suppose that *num* is 13. Then `LOG(2, 13)` [1126] is 3.7004397181411. `CEILING(3.7004397181411)` [1125] is 4, and `V = POWER(2, 4)` [1128], which is 16.)

2. Set $N = F(\text{column_list}) \& (V - 1)$.

3. While $N \geq \text{num}$:

- Set $V = \text{CEIL}(V / 2)$
- Set $N = N \& (V - 1)$

Suppose that the table `t1`, using linear hash partitioning and having 6 partitions, is created using this statement:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY LINEAR HASH( YEAR(col3) )
PARTITIONS 6;
```

Now assume that you want to insert two records into `t1` having the `col3` column values '2003-04-14' and '1998-10-19'. The partition number for the first of these is determined as follows:

```
V = POWER(2, CEILING( LOG(2,6) )) = 8
N = YEAR('2003-04-14') & (8 - 1)
  = 2003 & 7
  = 3
(3 >= 6 is FALSE: record stored in partition #3)
```

The number of the partition where the second record is stored is calculated as shown here:

```
V = 8
N = YEAR('1998-10-19') & (8-1)
  = 1998 & 7
  = 6
(6 >= 6 is TRUE: additional step required)
N = 6 & CEILING(8 / 2)
  = 6 & 3
  = 2
(2 >= 6 is FALSE: record stored in partition #2)
```

The advantage in partitioning by linear hash is that the adding, dropping, merging, and splitting of partitions is made much faster, which can be beneficial when dealing with tables containing extremely large amounts (terabytes) of data. The disadvantage is that data is less likely to be evenly distributed between partitions as compared with the distribution obtained using regular hash partitioning.

18.2.5. KEY Partitioning

Partitioning by key is similar to partitioning by hash, except that where hash partitioning employs a user-defined expression, the hashing function for key partitioning is supplied by the MySQL server. MySQL Cluster uses `MD5()` [1187] for this purpose; for tables using other storage engines, the server employs its own internal hashing function which is based on the same algorithm as `PASSWORD()` [1188].

The syntax rules for `CREATE TABLE ... PARTITION BY KEY` are similar to those for creating a table that is partitioned by hash. The major differences are listed here:

- `KEY` is used rather than `HASH`.
- `KEY` takes only a list of zero or more column names. Any columns used as the partitioning key must comprise part or all of the table's primary key, if the table has one. Where no column name is specified as the partitioning key, the table's primary key is used, if there is one. For example, the following `CREATE TABLE` statement is valid in MySQL 5.5:

```
CREATE TABLE k1 (
  id INT NOT NULL PRIMARY KEY,
  name VARCHAR(20)
)
PARTITION BY KEY()
PARTITIONS 2;
```

If there is no primary key but there is a unique key, then the unique key is used for the partitioning key:

```
CREATE TABLE k1 (
  id INT NOT NULL,
  name VARCHAR(20),
  UNIQUE KEY (id)
)
PARTITION BY KEY()
PARTITIONS 2;
```

However, if the unique key column were not defined as `NOT NULL`, then the previous statement would fail.

In both of these cases, the partitioning key is the `id` column, even though it is not shown in the output of `SHOW CREATE TABLE` or in the `PARTITION_EXPRESSION` column of the `INFORMATION_SCHEMA.PARTITIONS` table.

Unlike the case with other partitioning types, columns used for partitioning by `KEY` are not restricted to integer or `NULL` values. For example, the following `CREATE TABLE` statement is valid:

```
CREATE TABLE tm1 (
  s1 CHAR(32) PRIMARY KEY
)
PARTITION BY KEY(s1)
PARTITIONS 10;
```

The preceding statement would *not* be valid, were a different partitioning type to be specified. (In this case, simply using `PARTITION BY KEY()` would also be valid and have the same effect as `PARTITION BY KEY(s1)`, since `s1` is the table's primary key.)

For additional information about this issue, see [Section 18.5, “Restrictions and Limitations on Partitioning”](#).



Note

Tables using the `NDBCLUSTER` storage engine are implicitly partitioned by `KEY`, again using the table's primary key as the partitioning key. In the event that the MySQL Cluster table has no explicit primary key, the “hidden” primary key generated by the `NDBCLUSTER` storage engine for each MySQL Cluster table is used as the partitioning key.

If you define an explicit partitioning scheme for an `NDBCLUSTER` table, the table must have an explicit primary key, and any columns used in the partitioning

expression must be part of this key. However, if the table uses an “empty” partitioning expression—that is, `PARTITION BY KEY()` with no column references—then no explicit primary key is required.

You can observe this partitioning using the `ndb_desc` utility (with the `-p` option).



Important

For a key-partitioned table, you cannot execute an `ALTER TABLE DROP PRIMARY KEY`, as doing so generates the error `ERROR 1466 (HY000): Field in list of fields for partition function not found in table`. This is not an issue for MySQL Cluster tables which are partitioned by `KEY`; in such cases, the table is reorganized using the “hidden” primary key as the table's new partitioning key. See [Chapter 17, MySQL Cluster NDB 7.2](#).

It is also possible to partition a table by linear key. Here is a simple example:

```
CREATE TABLE tk (
  col1 INT NOT NULL,
  col2 CHAR(5),
  col3 DATE
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

Using `LINEAR` has the same effect on `KEY` partitioning as it does on `HASH` partitioning, with the partition number being derived using a powers-of-two algorithm rather than modulo arithmetic. See [Section 18.2.4.1, “LINEAR HASH Partitioning”](#), for a description of this algorithm and its implications.

18.2.6. Subpartitioning

Subpartitioning—also known as *composite partitioning*—is the further division of each partition in a partitioned table. Consider the following `CREATE TABLE` statement:

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) )
SUBPARTITIONS 2 (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

Table `ts` has 3 `RANGE` partitions. Each of these partitions—`p0`, `p1`, and `p2`—is further divided into 2 subpartitions. In effect, the entire table is divided into $3 * 2 = 6$ partitions. However, due to the action of the `PARTITION BY RANGE` clause, the first 2 of these store only those records with a value less than 1990 in the `purchased` column.

In MySQL 5.5, it is possible to subpartition tables that are partitioned by `RANGE` or `LIST`. Subpartitions may use either `HASH` or `KEY` partitioning. This is also known as *composite partitioning*.



Note

`SUBPARTITION BY HASH` and `SUBPARTITION BY KEY` generally follow the same syntax rules as `PARTITION BY HASH` and `PARTITION BY KEY`, respectively. An exception to this is that `SUBPARTITION BY KEY` (unlike `PARTITION BY KEY`) does not currently support a default column, so the column

used for this purpose must be specified, even if the table has an explicit primary key. This is a known issue which we are working to address; see [Issues with subpartitions](#), for more information and an example.

It is also possible to define subpartitions explicitly using `SUBPARTITION` clauses to specify options for individual subpartitions. For example, a more verbose fashion of creating the same table `ts` as shown in the previous example would be:

```
CREATE TABLE ts (id INT, purchased DATE)
  PARTITION BY RANGE( YEAR(purchased) )
  SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990) (
      SUBPARTITION s0,
      SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000) (
      SUBPARTITION s2,
      SUBPARTITION s3
    ),
    PARTITION p2 VALUES LESS THAN MAXVALUE (
      SUBPARTITION s4,
      SUBPARTITION s5
    )
  );
```

Some syntactical items of note are listed here:

- Each partition must have the same number of subpartitions.
- If you explicitly define any subpartitions using `SUBPARTITION` on any partition of a partitioned table, you must define them all. In other words, the following statement will fail:

```
CREATE TABLE ts (id INT, purchased DATE)
  PARTITION BY RANGE( YEAR(purchased) )
  SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990) (
      SUBPARTITION s0,
      SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE (
      SUBPARTITION s2,
      SUBPARTITION s3
    )
  );
```

This statement would still fail even if it included a `SUBPARTITIONS 2` clause.

- Each `SUBPARTITION` clause must include (at a minimum) a name for the subpartition. Otherwise, you may set any desired option for the subpartition or allow it to assume its default setting for that option.
- Subpartition names must be unique across the entire table. For example, the following `CREATE TABLE` statement is valid in MySQL 5.5:

```
CREATE TABLE ts (id INT, purchased DATE)
  PARTITION BY RANGE( YEAR(purchased) )
  SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990) (
      SUBPARTITION s0,
      SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000) (
```

Subpartitioning

```
        SUBPARTITION s2,  
        SUBPARTITION s3  
    ),  
    PARTITION p2 VALUES LESS THAN MAXVALUE (  
        SUBPARTITION s4,  
        SUBPARTITION s5  
    )  
);
```

Subpartitions can be used with especially large tables to distribute data and indexes across many disks. Suppose that you have 6 disks mounted as `/disk0`, `/disk1`, `/disk2`, and so on. Now consider the following example:

```
CREATE TABLE ts (id INT, purchased DATE)  
PARTITION BY RANGE( YEAR(purchased) )  
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (  
    PARTITION p0 VALUES LESS THAN (1990) (  
        SUBPARTITION s0  
            DATA DIRECTORY = '/disk0/data'  
            INDEX DIRECTORY = '/disk0/idx',  
        SUBPARTITION s1  
            DATA DIRECTORY = '/disk1/data'  
            INDEX DIRECTORY = '/disk1/idx'  
    ),  
    PARTITION p1 VALUES LESS THAN (2000) (  
        SUBPARTITION s2  
            DATA DIRECTORY = '/disk2/data'  
            INDEX DIRECTORY = '/disk2/idx',  
        SUBPARTITION s3  
            DATA DIRECTORY = '/disk3/data'  
            INDEX DIRECTORY = '/disk3/idx'  
    ),  
    PARTITION p2 VALUES LESS THAN MAXVALUE (  
        SUBPARTITION s4  
            DATA DIRECTORY = '/disk4/data'  
            INDEX DIRECTORY = '/disk4/idx',  
        SUBPARTITION s5  
            DATA DIRECTORY = '/disk5/data'  
            INDEX DIRECTORY = '/disk5/idx'  
    )  
);
```

In this case, a separate disk is used for the data and for the indexes of each [RANGE](#). Many other variations are possible; another example might be:

```
CREATE TABLE ts (id INT, purchased DATE)  
PARTITION BY RANGE(YEAR(purchased))  
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (  
    PARTITION p0 VALUES LESS THAN (1990) (  
        SUBPARTITION s0a  
            DATA DIRECTORY = '/disk0'  
            INDEX DIRECTORY = '/disk1',  
        SUBPARTITION s0b  
            DATA DIRECTORY = '/disk2'  
            INDEX DIRECTORY = '/disk3'  
    ),  
    PARTITION p1 VALUES LESS THAN (2000) (  
        SUBPARTITION s1a  
            DATA DIRECTORY = '/disk4/data'  
            INDEX DIRECTORY = '/disk4/idx',  
        SUBPARTITION s1b  
            DATA DIRECTORY = '/disk5/data'  
            INDEX DIRECTORY = '/disk5/idx'  
    ),  
    PARTITION p2 VALUES LESS THAN MAXVALUE (  
        SUBPARTITION s2a,
```

```

        SUBPARTITION s2b
    )
);

```

Here, the storage is as follows:

- Rows with `purchased` dates from before 1990 take up a vast amount of space, so are split up 4 ways, with a separate disk dedicated to the data and to the indexes for each of the two subpartitions (`s0a` and `s0b`) making up partition `p0`. In other words:
 - The data for subpartition `s0a` is stored on `/disk0`.
 - The indexes for subpartition `s0a` are stored on `/disk1`.
 - The data for subpartition `s0b` is stored on `/disk2`.
 - The indexes for subpartition `s0b` are stored on `/disk3`.
- Rows containing dates ranging from 1990 to 1999 (partition `p1`) do not require as much room as those from before 1990. These are split between 2 disks (`/disk4` and `/disk5`) rather than 4 disks as with the legacy records stored in `p0`:
 - Data and indexes belonging to `p1`'s first subpartition (`s1a`) are stored on `/disk4`—the data in `/disk4/data`, and the indexes in `/disk4/idx`.
 - Data and indexes belonging to `p1`'s second subpartition (`s1b`) are stored on `/disk5`—the data in `/disk5/data`, and the indexes in `/disk5/idx`.
- Rows reflecting dates from the year 2000 to the present (partition `p2`) do not take up as much space as required by either of the two previous ranges. Currently, it is sufficient to store all of these in the default location.

In future, when the number of purchases for the decade beginning with the year 2000 grows to a point where the default location no longer provides sufficient space, the corresponding rows can be moved using an `ALTER TABLE ... REORGANIZE PARTITION` statement. See [Section 18.3, “Partition Management”](#), for an explanation of how this can be done.

The `DATA DIRECTORY` and `INDEX DIRECTORY` options are not permitted in partition definitions when the `NO_DIR_IN_CREATE` [614] server SQL mode is in effect. Beginning with MySQL 5.5.5, these options are also not permitted when defining subpartitions (Bug #42954).

18.2.7. How MySQL Partitioning Handles `NULL`

Partitioning in MySQL does nothing to disallow `NULL` as the value of a partitioning expression, whether it is a column value or the value of a user-supplied expression. Even though it is permitted to use `NULL` as the value of an expression that must otherwise yield an integer, it is important to keep in mind that `NULL` is not a number. MySQL's partitioning implementation treats `NULL` as being less than any non-`NULL` value, just as `ORDER BY` does.

This means that treatment of `NULL` varies between partitioning of different types, and may produce behavior which you do not expect if you are not prepared for it. This being the case, we discuss in this section how each MySQL partitioning type handles `NULL` values when determining the partition in which a row should be stored, and provide examples for each.

Handling of `NULL` with `RANGE` partitioning. If you insert a row into a table partitioned by `RANGE` such that the column value used to determine the partition is `NULL`, the row is inserted into the lowest partition. Consider these two tables in a database named `p`, created as follows:

```
mysql> CREATE TABLE t1 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY RANGE(c1) (
->   PARTITION p0 VALUES LESS THAN (0),
->   PARTITION p1 VALUES LESS THAN (10),
->   PARTITION p2 VALUES LESS THAN MAXVALUE
-> );
```

Query OK, 0 rows affected (0.09 sec)

```
mysql> CREATE TABLE t2 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY RANGE(c1) (
->   PARTITION p0 VALUES LESS THAN (-5),
->   PARTITION p1 VALUES LESS THAN (0),
->   PARTITION p2 VALUES LESS THAN (10),
->   PARTITION p3 VALUES LESS THAN MAXVALUE
-> );
```

Query OK, 0 rows affected (0.09 sec)

You can see the partitions created by these two `CREATE TABLE` statements using the following query against the `PARTITIONS` table in the `INFORMATION_SCHEMA` database:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
```

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
t1	p0	0	0	0
t1	p1	0	0	0
t1	p2	0	0	0
t2	p0	0	0	0
t2	p1	0	0	0
t2	p2	0	0	0
t2	p3	0	0	0

7 rows in set (0.00 sec)

(For more information about this table, see [Section 20.12, “The `INFORMATION_SCHEMA PARTITIONS Table`”](#).) Now let us populate each of these tables with a single row containing a `NULL` in the column used as the partitioning key, and verify that the rows were inserted using a pair of `SELECT` statements:

```
mysql> INSERT INTO t1 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO t2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM t1;
+-----+-----+
| id  | name  |
+-----+-----+
| NULL | mothra |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM t2;
+-----+-----+
| id  | name  |
+-----+-----+
| NULL | mothra |
+-----+-----+
```

```
1 row in set (0.00 sec)
```

You can see which partitions are used to store the inserted rows by rerunning the previous query against `INFORMATION_SCHEMA.PARTITIONS` and inspecting the output:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
```

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
t1	p0	1	20	20
t1	p1	0	0	0
t1	p2	0	0	0
t2	p0	1	20	20
t2	p1	0	0	0
t2	p2	0	0	0
t2	p3	0	0	0

```
7 rows in set (0.01 sec)
```

You can also demonstrate that these rows were stored in the lowest partition of each table by dropping these partitions, and then re-running the `SELECT` statements:

```
mysql> ALTER TABLE t1 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE t2 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

(For more information on `ALTER TABLE ... DROP PARTITION`, see [Section 13.1.7](#), “`ALTER TABLE Syntax`”).

`NULL` is also treated in this way for partitioning expressions that use SQL functions. Suppose that we define a table using a `CREATE TABLE` statement such as this one:

```
CREATE TABLE tndate (
  id INT,
  dt DATE
)
PARTITION BY RANGE( YEAR(dt) )(
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

As with other MySQL functions, `YEAR(NULL)` [1152] returns `NULL`. A row with a `dt` column value of `NULL` is treated as though the partitioning expression evaluated to a value less than any other value, and so is inserted into partition `p0`.

Handling of `NULL` with `LIST` partitioning. A table that is partitioned by `LIST` admits `NULL` values if and only if one of its partitions is defined using that value-list that contains `NULL`. The converse of this is that a table partitioned by `LIST` which does not explicitly use `NULL` in a value list rejects rows resulting in a `NULL` value for the partitioning expression, as shown in this example:

```
mysql> CREATE TABLE ts1 (
-> c1 INT,
```

```

->     c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->     PARTITION p0 VALUES IN (0, 3, 6),
->     PARTITION p1 VALUES IN (1, 4, 7),
->     PARTITION p2 VALUES IN (2, 5, 8)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts1 VALUES (9, 'mothra');
ERROR 1504 (HY000): Table has no partition for value 9

mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');
ERROR 1504 (HY000): Table has no partition for value NULL

```

Only rows having a `c1` value between 0 and 8 inclusive can be inserted into `ts1`. `NULL` falls outside this range, just like the number 9. We can create tables `ts2` and `ts3` having value lists containing `NULL`, as shown here:

```

mysql> CREATE TABLE ts2 (
->     c1 INT,
->     c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->     PARTITION p0 VALUES IN (0, 3, 6),
->     PARTITION p1 VALUES IN (1, 4, 7),
->     PARTITION p2 VALUES IN (2, 5, 8),
->     PARTITION p3 VALUES IN (NULL)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE ts3 (
->     c1 INT,
->     c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->     PARTITION p0 VALUES IN (0, 3, 6),
->     PARTITION p1 VALUES IN (1, 4, 7, NULL),
->     PARTITION p2 VALUES IN (2, 5, 8)
-> );
Query OK, 0 rows affected (0.01 sec)

```

When defining value lists for partitioning, you can (and should) treat `NULL` just as you would any other value. For example, both `VALUES IN (NULL)` and `VALUES IN (1, 4, 7, NULL)` are valid, as are `VALUES IN (1, NULL, 4, 7)`, `VALUES IN (NULL, 1, 4, 7)`, and so on. You can insert a row having `NULL` for column `c1` into each of the tables `ts2` and `ts3`:

```

mysql> INSERT INTO ts2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ts3 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

```

By issuing the appropriate query against `INFORMATION_SCHEMA.PARTITIONS`, you can determine which partitions were used to store the rows just inserted (we assume, as in the previous examples, that the partitioned tables were created in the `p` database):

```

mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 'ts_';

```

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
ts2	p0	0	0	0
ts2	p1	0	0	0

```

| ts2      | p2      |          0 |          0 |          0 |
| / ts2    | / p3    | /          1 / |          20 / |          20 / |
| ts3      | p0      |          0 |          0 |          0 |
| / ts3    | / p1    | /          1 / |          20 / |          20 / |
| ts3      | p2      |          0 |          0 |          0 |
+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)
    
```

As shown earlier in this section, you can also verify which partitions were used for storing the rows by deleting these partitions and then performing a `SELECT`.

Handling of `NULL` with `HASH` and `KEY` partitioning. `NULL` is handled somewhat differently for tables partitioned by `HASH` or `KEY`. In these cases, any partition expression that yields a `NULL` value is treated as though its return value were zero. We can verify this behavior by examining the effects on the file system of creating a table partitioned by `HASH` and populating it with a record containing appropriate values. Suppose that you have a table `th` (also in the `p` database) created using the following statement:

```

mysql> CREATE TABLE th (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY HASH(c1)
-> PARTITIONS 2;
Query OK, 0 rows affected (0.00 sec)
    
```

The partitions belonging to this table can be viewed using the query shown here:

```

mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME = 'th';
+-----+-----+-----+-----+-----+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+-----+-----+-----+-----+-----+
| th          | p0             |          0 |          0     |          0   |
| th          | p1             |          0 |          0     |          0   |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
    
```

Note that `TABLE_ROWS` for each partition is 0. Now insert two rows into `th` whose `c1` column values are `NULL` and 0, and verify that these rows were inserted, as shown here:

```

mysql> INSERT INTO th VALUES (NULL, 'mothra'), (0, 'gigan');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM th;
+-----+-----+
| c1   | c2     |
+-----+-----+
| NULL | mothra |
+-----+-----+
| 0    | gigan  |
+-----+-----+
2 rows in set (0.01 sec)
    
```

Recall that for any integer `N`, the value of `NULL MOD N` is always `NULL`. For tables that are partitioned by `HASH` or `KEY`, this result is treated for determining the correct partition as 0. Checking the `INFORMATION_SCHEMA.PARTITIONS` table once again, we can see that both rows were inserted into partition `p0`:

```

mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME = 'th';
+-----+-----+-----+-----+-----+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+-----+-----+-----+-----+-----+
    
```

```

+-----+-----+-----+-----+-----+
| th      | p0      |      2 |      20 |      20 |
| th      | p1      |      0 |      0  |      0  |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

If you repeat this example using `PARTITION BY KEY` in place of `PARTITION BY HASH` in the definition of the table, you can verify easily that `NULL` is also treated like 0 for this type of partitioning.

18.3. Partition Management

MySQL 5.5 provides a number of ways to modify partitioned tables. It is possible to add, drop, redefine, merge, or split existing partitions. All of these actions can be carried out using the partitioning extensions to the `ALTER TABLE` statement. There are also ways to obtain information about partitioned tables and partitions. We discuss these topics in the sections that follow.

- For information about partition management in tables partitioned by `RANGE` or `LIST`, see [Section 18.3.1, “Management of RANGE and LIST Partitions”](#).
- For a discussion of managing `HASH` and `KEY` partitions, see [Section 18.3.2, “Management of HASH and KEY Partitions”](#).
- See [Section 18.3.4, “Obtaining Information About Partitions”](#), for a discussion of mechanisms provided in MySQL 5.5 for obtaining information about partitioned tables and partitions.
- For a discussion of performing maintenance operations on partitions, see [Section 18.3.3, “Maintenance of Partitions”](#).



Note

In MySQL 5.5, all partitions of a partitioned table must have the same number of subpartitions, and it is not possible to change the subpartitioning once the table has been created.

To change a table's partitioning scheme, it is necessary only to use the `ALTER TABLE` statement with a `partition_options` clause. This clause has the same syntax as that as used with `CREATE TABLE` for creating a partitioned table, and always begins with the keywords `PARTITION BY`. Suppose that you have a table partitioned by range using the following `CREATE TABLE` statement:

```

CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE( YEAR(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (2000),
    PARTITION p3 VALUES LESS THAN (2005)
);

```

To repartition this table so that it is partitioned by key into two partitions using the `id` column value as the basis for the key, you can use this statement:

```

ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;

```

This has the same effect on the structure of the table as dropping the table and re-creating it using `CREATE TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;`

`ALTER TABLE ... ENGINE = ...` changes only the storage engine used by the table, and leaves the table's partitioning scheme intact. Use `ALTER TABLE ... REMOVE PARTITIONING` to remove a table's partitioning. See [Section 13.1.7, “ALTER TABLE Syntax”](#).

**Important**

Only a single `PARTITION BY`, `ADD PARTITION`, `DROP PARTITION`, `REORGANIZE PARTITION`, or `COALESCE PARTITION` clause can be used in a given `ALTER TABLE` statement. If you (for example) wish to drop a partition and reorganize a table's remaining partitions, you must do so in two separate `ALTER TABLE` statements (one using `DROP PARTITION` and then a second one using `REORGANIZE PARTITIONS`).

Beginning with MySQL 5.5.0, it is possible to delete all rows from one or more selected partitions using `ALTER TABLE ... TRUNCATE PARTITION`.

18.3.1. Management of RANGE and LIST Partitions

Range and list partitions are very similar with regard to how the adding and dropping of partitions are handled. For this reason we discuss the management of both sorts of partitioning in this section. For information about working with tables that are partitioned by hash or key, see [Section 18.3.2, "Management of HASH and KEY Partitions"](#). Dropping a RANGE or LIST partition is more straightforward than adding one, so we discuss this first.

Dropping a partition from a table that is partitioned by either RANGE or by LIST can be accomplished using the `ALTER TABLE` statement with a `DROP PARTITION` clause. Here is a very basic example, which supposes that you have already created a table which is partitioned by range and then populated with 10 records using the following `CREATE TABLE` and `INSERT` statements:

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
->     PARTITION BY RANGE( YEAR(purchased) ) (
->         PARTITION p0 VALUES LESS THAN (1990),
->         PARTITION p1 VALUES LESS THAN (1995),
->         PARTITION p2 VALUES LESS THAN (2000),
->         PARTITION p3 VALUES LESS THAN (2005)
->     );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO tr VALUES
->     (1, 'desk organiser', '2003-10-15'),
->     (2, 'CD player', '1993-11-05'),
->     (3, 'TV set', '1996-03-10'),
->     (4, 'bookcase', '1982-01-10'),
->     (5, 'exercise bike', '2004-05-09'),
->     (6, 'sofa', '1987-06-05'),
->     (7, 'popcorn maker', '2001-11-22'),
->     (8, 'aquarium', '1992-08-04'),
->     (9, 'study desk', '1984-09-16'),
->     (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.01 sec)
```

You can see which items should have been inserted into partition `p2` as shown here:

```
mysql> SELECT * FROM tr
->     WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+-----+-----+-----+
| id   | name      | purchased |
+-----+-----+-----+
| 3    | TV set    | 1996-03-10 |
| 10   | lava lamp | 1998-12-25 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

To drop the partition named `p2`, execute the following command:

```
mysql> ALTER TABLE tr DROP PARTITION p2;
```

Query OK, 0 rows affected (0.03 sec)



Note

The `NDBCLUSTER` storage engine does not support `ALTER TABLE ... DROP PARTITION`. It does, however, support the other partitioning-related extensions to `ALTER TABLE` that are described in this chapter.

It is very important to remember that, *when you drop a partition, you also delete all the data that was stored in that partition*. You can see that this is the case by re-running the previous `SELECT` query:

```
mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

Because of this, you must have the `DROP` [\[696\]](#) privilege for a table before you can execute `ALTER TABLE ... DROP PARTITION` on that table.

If you wish to drop all data from all partitions while preserving the table definition and its partitioning scheme, use the `TRUNCATE TABLE` statement. (See [Section 13.1.33](#), “`TRUNCATE TABLE Syntax`”).

If you intend to change the partitioning of a table *without* losing data, use `ALTER TABLE ... REORGANIZE PARTITION` instead. See below or in [Section 13.1.7](#), “`ALTER TABLE Syntax`”, for information about `REORGANIZE PARTITION`.

If you now execute a `SHOW CREATE TABLE` statement, you can see how the partitioning makeup of the table has been changed:

```
mysql> SHOW CREATE TABLE tr\G
***** 1. row *****
      Table: tr
Create Table: CREATE TABLE `tr` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE ( YEAR(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.01 sec)
```

When you insert new rows into the changed table with `purchased` column values between `'1995-01-01'` and `'2004-12-31'` inclusive, those rows will be stored in partition `p3`. You can verify this as follows:

```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '2004-12-31';
+-----+-----+-----+
| id  | name          | purchased |
+-----+-----+-----+
| 11  | pencil holder | 1995-07-12 |
| 1   | desk organiser | 2003-10-15 |
| 5   | exercise bike | 2004-05-09 |
| 7   | popcorn maker | 2001-11-22 |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> ALTER TABLE tr DROP PARTITION p3;
```

```
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM tr WHERE purchased
      -> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

Note that the number of rows dropped from the table as a result of [ALTER TABLE ... DROP PARTITION](#) is not reported by the server as it would be by the equivalent [DELETE](#) query.

Dropping [LIST](#) partitions uses exactly the same [ALTER TABLE ... DROP PARTITION](#) syntax as used for dropping [RANGE](#) partitions. However, there is one important difference in the effect this has on your use of the table afterward: You can no longer insert into the table any rows having any of the values that were included in the value list defining the deleted partition. (See [Section 18.2.2, “LIST Partitioning”](#), for an example.)

To add a new range or list partition to a previously partitioned table, use the [ALTER TABLE ... ADD PARTITION](#) statement. For tables which are partitioned by [RANGE](#), this can be used to add a new range to the end of the list of existing partitions. Suppose that you have a partitioned table containing membership data for your organization, which is defined as follows:

```
CREATE TABLE members (
  id INT,
  fname VARCHAR(25),
  lname VARCHAR(25),
  dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
  PARTITION p0 VALUES LESS THAN (1970),
  PARTITION p1 VALUES LESS THAN (1980),
  PARTITION p2 VALUES LESS THAN (1990)
);
```

Suppose further that the minimum age for members is 16. As the calendar approaches the end of 2005, you realize that you will soon be admitting members who were born in 1990 (and later in years to come). You can modify the `members` table to accommodate new members born in the years 1990 to 1999 as shown here:

```
ALTER TABLE ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));
```



Important

With tables that are partitioned by range, you can use [ADD PARTITION](#) to add new partitions to the high end of the partitions list only. Trying to add a new partition in this manner between or before existing partitions will result in an error as shown here:

```
mysql> ALTER TABLE members
      > ADD PARTITION (
      > PARTITION p3 VALUES LESS THAN (1960));
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly »
      increasing for each partition
```

In a similar fashion, you can add new partitions to a table that is partitioned by [LIST](#). Suppose a table `tt` is defined using the following [CREATE TABLE](#) statement:

```
CREATE TABLE tt (
  id INT,
  data INT
)
PARTITION BY LIST(data) (
  PARTITION p0 VALUES IN (5, 10, 15),
  PARTITION p1 VALUES IN (6, 12, 18)
```

```
);
```

You can add a new partition in which to store rows having the `data` column values 7, 14, and 21 as shown:

```
ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));
```

Note that you *cannot* add a new [LIST](#) partition encompassing any values that are already included in the value list of an existing partition. If you attempt to do so, an error will result:

```
mysql> ALTER TABLE tt ADD PARTITION
> (PARTITION np VALUES IN (4, 8, 12));
ERROR 1465 (HY000): Multiple definition of same constant »
in list partitioning
```

Because any rows with the `data` column value 12 have already been assigned to partition `p1`, you cannot create a new partition on table `tt` that includes 12 in its value list. To accomplish this, you could drop `p1`, and add `np` and then a new `p1` with a modified definition. However, as discussed earlier, this would result in the loss of all data stored in `p1`—and it is often the case that this is not what you really want to do. Another solution might appear to be to make a copy of the table with the new partitioning and to copy the data into it using `CREATE TABLE ... SELECT ...`, then drop the old table and rename the new one, but this could be very time-consuming when dealing with a large amounts of data. This also might not be feasible in situations where high availability is a requirement.

You can add multiple partitions in a single `ALTER TABLE ... ADD PARTITION` statement as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
  PARTITION p1 VALUES LESS THAN (1991),
  PARTITION p2 VALUES LESS THAN (1996),
  PARTITION p3 VALUES LESS THAN (2001),
  PARTITION p4 VALUES LESS THAN (2005)
);

ALTER TABLE employees ADD PARTITION (
  PARTITION p5 VALUES LESS THAN (2010),
  PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

Fortunately, MySQL's partitioning implementation provides ways to redefine partitions without losing data. Let us look first at a couple of simple examples involving [RANGE](#) partitioning. Recall the `members` table which is now defined as shown here:

```
mysql> SHOW CREATE TABLE members\G
***** 1. row *****
      Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) default NULL,
  `fname` varchar(25) default NULL,
  `lname` varchar(25) default NULL,
  `dob` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE ( YEAR(dob) ) (
  PARTITION p0 VALUES LESS THAN (1970) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1980) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2000) ENGINE = MyISAM
```

)

Suppose that you would like to move all rows representing members born before 1960 into a separate partition. As we have already seen, this cannot be done using `ALTER TABLE ... ADD PARTITION`. However, you can use another partition-related extension to `ALTER TABLE` to accomplish this:

```
ALTER TABLE members REORGANIZE PARTITION p0 INTO (
    PARTITION s0 VALUES LESS THAN (1960),
    PARTITION s1 VALUES LESS THAN (1970)
);
```

In effect, this command splits partition `p0` into two new partitions `s0` and `s1`. It also moves the data that was stored in `p0` into the new partitions according to the rules embodied in the two `PARTITION ... VALUES ...` clauses, so that `s0` contains only those records for which `YEAR(dob)` [1152] is less than 1960 and `s1` contains those rows in which `YEAR(dob)` [1152] is greater than or equal to 1960 but less than 1970.

A `REORGANIZE PARTITION` clause may also be used for merging adjacent partitions. You can return the `members` table to its previous partitioning as shown here:

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
    PARTITION p0 VALUES LESS THAN (1970)
);
```

No data is lost in splitting or merging partitions using `REORGANIZE PARTITION`. In executing the above statement, MySQL moves all of the records that were stored in partitions `s0` and `s1` into partition `p0`.

The general syntax for `REORGANIZE PARTITION` is shown here:

```
ALTER TABLE tbl_name
    REORGANIZE PARTITION partition_list
    INTO (partition_definitions);
```

Here, `tbl_name` is the name of the partitioned table, and `partition_list` is a comma-separated list of names of one or more existing partitions to be changed. `partition_definitions` is a comma-separated list of new partition definitions, which follow the same rules as for the `partition_definitions` list used in `CREATE TABLE` (see [Section 13.1.17, “CREATE TABLE Syntax”](#)). It should be noted that you are not limited to merging several partitions into one, or to splitting one partition into many, when using `REORGANIZE PARTITION`. For example, you can reorganize all four partitions of the `members` table into two, as follows:

```
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
    PARTITION m0 VALUES LESS THAN (1980),
    PARTITION m1 VALUES LESS THAN (2000)
);
```

You can also use `REORGANIZE PARTITION` with tables that are partitioned by `LIST`. Let us return to the problem of adding a new partition to the list-partitioned `tt` table and failing because the new partition had a value that was already present in the value-list of one of the existing partitions. We can handle this by adding a partition that contains only nonconflicting values, and then reorganizing the new partition and the existing one so that the value which was stored in the existing one is now moved to the new one:

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
    PARTITION p1 VALUES IN (6, 18),
    PARTITION np VALUES in (4, 8, 12)
);
```

Here are some key points to keep in mind when using `ALTER TABLE ... REORGANIZE PARTITION` to repartition tables that are partitioned by `RANGE` or `LIST`:

- The [PARTITION](#) clauses used to determine the new partitioning scheme are subject to the same rules as those used with a [CREATE TABLE](#) statement.

Most importantly, you should remember that the new partitioning scheme cannot have any overlapping ranges (applies to tables partitioned by [RANGE](#)) or sets of values (when reorganizing tables partitioned by [LIST](#)).

- The combination of partitions in the *partition_definitions* list should account for the same range or set of values overall as the combined partitions named in the *partition_list*.

For instance, in the `members` table used as an example in this section, partitions `p1` and `p2` together cover the years 1980 through 1999. Therefore, any reorganization of these two partitions should cover the same range of years overall.

- For tables partitioned by [RANGE](#), you can reorganize only adjacent partitions; you cannot skip over range partitions.

For instance, you could not reorganize the `members` table used as an example in this section using a statement beginning with `ALTER TABLE members REORGANIZE PARTITION p0,p2 INTO ...` because `p0` covers the years prior to 1970 and `p2` the years from 1990 through 1999 inclusive, and thus the two are not adjacent partitions.

- You cannot use [REORGANIZE PARTITION](#) to change the table's partitioning type; that is, you cannot (for example) change [RANGE](#) partitions to [HASH](#) partitions or *vice versa*. You also cannot use this command to change the partitioning expression or column. To accomplish either of these tasks without dropping and re-creating the table, you can use `ALTER TABLE ... PARTITION BY ...`. For example:

```
ALTER TABLE members
PARTITION BY HASH( YEAR(dob) )
PARTITIONS 8;
```

18.3.2. Management of [HASH](#) and [KEY](#) Partitions

Tables which are partitioned by hash or by key are very similar to one another with regard to making changes in a partitioning setup, and both differ in a number of ways from tables which have been partitioned by range or list. For that reason, this section addresses the modification of tables partitioned by hash or by key only. For a discussion of adding and dropping of partitions of tables that are partitioned by range or list, see [Section 18.3.1, “Management of \[RANGE\]\(#\) and \[LIST\]\(#\) Partitions”](#).

You cannot drop partitions from tables that are partitioned by [HASH](#) or [KEY](#) in the same way that you can from tables that are partitioned by [RANGE](#) or [LIST](#). However, you can merge [HASH](#) or [KEY](#) partitions using the `ALTER TABLE ... COALESCE PARTITION` statement. Suppose that you have a table containing data about clients, which is divided into twelve partitions. The `clients` table is defined as shown here:

```
CREATE TABLE clients (
  id INT,
  fname VARCHAR(30),
  lname VARCHAR(30),
  signed DATE
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;
```

To reduce the number of partitions from twelve to eight, execute the following `ALTER TABLE` command:

```
mysql> ALTER TABLE clients COALESCE PARTITION 4;
Query OK, 0 rows affected (0.02 sec)
```

`COALESCE` works equally well with tables that are partitioned by `HASH`, `KEY`, `LINEAR HASH`, or `LINEAR KEY`. Here is an example similar to the previous one, differing only in that the table is partitioned by `LINEAR KEY`:

```
mysql> CREATE TABLE clients_lk (
->   id INT,
->   fname VARCHAR(30),
->   lname VARCHAR(30),
->   signed DATE
-> )
-> PARTITION BY LINEAR KEY(signed)
-> PARTITIONS 12;
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE clients_lk COALESCE PARTITION 4;
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Note that the number following `COALESCE PARTITION` is the number of partitions to merge into the remainder—in other words, it is the number of partitions to remove from the table.

If you attempt to remove more partitions than the table has, the result is an error like the one shown:

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
ERROR 1478 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

To increase the number of partitions for the `clients` table from 12 to 18, use `ALTER TABLE ... ADD PARTITION` as shown here:

```
ALTER TABLE clients ADD PARTITION PARTITIONS 6;
```

18.3.3. Maintenance of Partitions

A number of table and partition maintenance tasks can be carried out using SQL statements intended for such purposes on partitioned tables in MySQL 5.5.

Table maintenance of partitioned tables can be accomplished using the statements `CHECK TABLE`, `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `REPAIR TABLE`, which are supported for partitioned tables.

You can use a number of extensions to `ALTER TABLE` for performing operations of this type on one or more partitions directly, as described in the following list:

- **Rebuilding partitions.** Rebuilds the partition; this has the same effect as dropping all records stored in the partition, then reinserting them. This can be useful for purposes of defragmentation.

Example:

```
ALTER TABLE t1 REBUILD PARTITION p0, p1;
```

- **Optimizing partitions.** If you have deleted a large number of rows from a partition or if you have made many changes to a partitioned table with variable-length rows (that is, having `VARCHAR`, `BLOB`, or `TEXT` columns), you can use `ALTER TABLE ... OPTIMIZE PARTITION` to reclaim any unused space and to defragment the partition data file.

Example:

```
ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
```

Using `OPTIMIZE PARTITION` on a given partition is equivalent to running `CHECK PARTITION`, `ANALYZE PARTITION`, and `REPAIR PARTITION` on that partition.

- **Analyzing partitions.** This reads and stores the key distributions for partitions.

Example:

```
ALTER TABLE t1 ANALYZE PARTITION p3;
```

- **Repairing partitions.** This repairs corrupted partitions.

Example:

```
ALTER TABLE t1 REPAIR PARTITION p0,p1;
```

- **Checking partitions.** You can check partitions for errors in much the same way that you can use `CHECK TABLE` with nonpartitioned tables.

Example:

```
ALTER TABLE trb3 CHECK PARTITION p1;
```

This command will tell you if the data or indexes in partition `p1` of table `t1` are corrupted. If this is the case, use `ALTER TABLE ... REPAIR PARTITION` to repair the partition.

Each of the statements in the list just shown also supports the keyword `ALL` in place of the list of partition names. Using `ALL` causes the statement to act on all partitions in the table.

The use of `mysqlcheck` and `myisamchk` is not supported with partitioned tables.

Beginning with MySQL 5.5.0, you can also truncate partitions using `ALTER TABLE ... TRUNCATE PARTITION`. This statement can be used to delete all rows from one or more partitions in much the same way that `TRUNCATE TABLE` deletes all rows from a table.

`ALTER TABLE ... TRUNCATE PARTITION ALL` truncates all partitions in the table.

18.3.4. Obtaining Information About Partitions

This section discusses obtaining information about existing partitions, which can be done in a number of ways. Methods of obtaining such information include the following:

- Using the `SHOW CREATE TABLE` statement to view the partitioning clauses used in creating a partitioned table.
- Using the `SHOW TABLE STATUS` statement to determine whether a table is partitioned.
- Querying the `INFORMATION_SCHEMA.PARTITIONS` table.
- Using the statement `EXPLAIN PARTITIONS SELECT` to see which partitions are used by a given `SELECT`.

As discussed elsewhere in this chapter, `SHOW CREATE TABLE` includes in its output the `PARTITION BY` clause used to create a partitioned table. For example:

```
mysql> SHOW CREATE TABLE trb3\G
***** 1. row *****
      Table: trb3
Create Table: CREATE TABLE `trb3` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

```

PARTITION BY RANGE (YEAR(purchased)) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (2000) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.00 sec)
    
```

The output from `SHOW TABLE STATUS` for partitioned tables is the same as that for nonpartitioned tables, except that the `Create_options` column contains the string `partitioned`. The `Engine` column contains the name of the storage engine used by all partitions of the table. (See [Section 13.7.5.37](#), “`SHOW TABLE STATUS Syntax`”, for more information about this statement.)

You can also obtain information about partitions from `INFORMATION_SCHEMA`, which contains a `PARTITIONS` table. See [Section 20.12](#), “The `INFORMATION_SCHEMA PARTITIONS Table`”.

It is possible to determine which partitions of a partitioned table are involved in a given `SELECT` query using `EXPLAIN PARTITIONS`. The `PARTITIONS` keyword adds a `partitions` column to the output of `EXPLAIN` listing the partitions from which records would be matched by the query.

Suppose that you have a table `trb1` created and populated as follows:

```

CREATE TABLE trb1 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE(id)
(
  PARTITION p0 VALUES LESS THAN (3),
  PARTITION p1 VALUES LESS THAN (7),
  PARTITION p2 VALUES LESS THAN (9),
  PARTITION p3 VALUES LESS THAN (11)
);

INSERT INTO trb1 VALUES
(1, 'desk organiser', '2003-10-15'),
(2, 'CD player', '1993-11-05'),
(3, 'TV set', '1996-03-10'),
(4, 'bookcase', '1982-01-10'),
(5, 'exercise bike', '2004-05-09'),
(6, 'sofa', '1987-06-05'),
(7, 'popcorn maker', '2001-11-22'),
(8, 'aquarium', '1992-08-04'),
(9, 'study desk', '1984-09-16'),
(10, 'lava lamp', '1998-12-25');
    
```

You can see which partitions are used in a query such as `SELECT * FROM trb1;`, as shown here:

```

mysql> EXPLAIN PARTITIONS SELECT * FROM trb1\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1,p2,p3
         type: ALL
possible_keys: NULL
          key: NULL
        key_len: NULL
         ref: NULL
         rows: 10
      Extra: Using filesort
    
```

In this case, all four partitions are searched. However, when a limiting condition making use of the partitioning key is added to the query, you can see that only those partitions containing matching values are scanned, as shown here:

```

mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5\G
    
```

```
***** 1. row *****
      id: 1
      select_type: SIMPLE
      table: trb1
      partitions: p0,p1
      type: ALL
      possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 10
      Extra: Using where
```

`EXPLAIN PARTITIONS` provides information about keys used and possible keys, just as with the standard `EXPLAIN SELECT` statement:

```
mysql> ALTER TABLE trb1 ADD PRIMARY KEY (id);
Query OK, 10 rows affected (0.03 sec)
Records: 10 Duplicates: 0 Warnings: 0

mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5\G
***** 1. row *****
      id: 1
      select_type: SIMPLE
      table: trb1
      partitions: p0,p1
      type: range
      possible_keys: PRIMARY
      key: PRIMARY
      key_len: 4
      ref: NULL
      rows: 7
      Extra: Using where
```

You should take note of the following restrictions and limitations on `EXPLAIN PARTITIONS`:

- You cannot use the `PARTITIONS` and `EXTENDED` keywords together in the same `EXPLAIN ... SELECT` statement. Attempting to do so produces a syntax error.
- If `EXPLAIN PARTITIONS` is used to examine a query against a nonpartitioned table, no error is produced, but the value of the `partitions` column is always `NULL`.

The `rows` column of `EXPLAIN PARTITIONS` output displays the total number of rows in the table.

See also [Section 13.8.2, “EXPLAIN Syntax”](#).

18.4. Partition Pruning

This section discusses an optimization known as *partition pruning*. The core concept behind partition pruning is relatively simple, and can be described as “Do not scan partitions where there can be no matching values”. Suppose that you have a partitioned table `t1` defined by this statement:

```
CREATE TABLE t1 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY RANGE( region_code ) (
  PARTITION p0 VALUES LESS THAN (64),
  PARTITION p1 VALUES LESS THAN (128),
  PARTITION p2 VALUES LESS THAN (192),
  PARTITION p3 VALUES LESS THAN MAXVALUE
```

```
);
```

Consider the case where you wish to obtain results from a query such as this one:

```
SELECT fname, lname, region_code, dob
FROM t1
WHERE region_code > 125 AND region_code < 130;
```

It is easy to see that none of the rows which ought to be returned will be in either of the partitions `p0` or `p3`; that is, we need to search only in partitions `p1` and `p2` to find matching rows. By doing so, it is possible to expend much less time and effort in finding matching rows than would be required to scan all partitions in the table. This “cutting away” of unneeded partitions is known as *pruning*. When the optimizer can make use of partition pruning in performing a query, execution of the query can be an order of magnitude faster than the same query against a nonpartitioned table containing the same column definitions and data.

The query optimizer can perform pruning whenever a `WHERE` condition can be reduced to either one of the following two cases:

- `partition_column = constant`
- `partition_column IN (constant1, constant2, ..., constantN)`

In the first case, the optimizer simply evaluates the partitioning expression for the value given, determines which partition contains that value, and scans only this partition. In many cases, the equal sign can be replaced with another arithmetic comparison, including `<`, `>`, `<=`, `>=`, and `<>`. Some queries using `BETWEEN` in the `WHERE` clause can also take advantage of partition pruning. See the examples later in this section.

In the second case, the optimizer evaluates the partitioning expression for each value in the list, creates a list of matching partitions, and then scans only the partitions in this partition list.

Pruning can also be applied to short ranges, which the optimizer can convert into equivalent lists of values. For instance, in the previous example, the `WHERE` clause can be converted to `WHERE region_code IN (126, 127, 128, 129)`. Then the optimizer can determine that the first three values in the list are found in partition `p1`, the remaining three values in partition `p2`, and that the other partitions contain no relevant values and so do not need to be searched for matching rows.

Beginning with MySQL 5.5.0, the optimizer can also perform pruning for queries that involve comparisons of the preceding types on multiple columns for tables that use `RANGE COLUMNS` or `LIST COLUMNS` partitioning.

This type of optimization can be applied whenever the partitioning expression consists of an equality or a range which can be reduced to a set of equalities, or when the partitioning expression represents an increasing or decreasing relationship. Pruning can also be applied for tables partitioned on a `DATE` or `DATETIME` column when the partitioning expression uses the `YEAR()` [1152] or `TO_DAYS()` [1148] function. In addition, in MySQL 5.5, pruning can be applied for such tables when the partitioning expression uses the `TO_SECONDS()` [1149] function.

Suppose that table `t2`, defined as shown here, is partitioned on a `DATE` column:

```
CREATE TABLE t2 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY RANGE( YEAR(dob) ) (
  PARTITION d0 VALUES LESS THAN (1970),
  PARTITION d1 VALUES LESS THAN (1975),
```

```

PARTITION d2 VALUES LESS THAN (1980),
PARTITION d3 VALUES LESS THAN (1985),
PARTITION d4 VALUES LESS THAN (1990),
PARTITION d5 VALUES LESS THAN (2000),
PARTITION d6 VALUES LESS THAN (2005),
PARTITION d7 VALUES LESS THAN MAXVALUE
);

```

The following queries on `t2` can make use of partition pruning:

```

SELECT * FROM t2 WHERE dob = '1982-06-23';

SELECT * FROM t2 WHERE dob BETWEEN '1991-02-15' AND '1997-04-25';

SELECT * FROM t2 WHERE dob >= '1984-06-21' AND dob <= '1999-06-21'

```

In the case of the last query, the optimizer can also act as follows:

1. Find the partition containing the low end of the range.

`YEAR('1984-06-21')` [1152] yields the value 1984, which is found in partition `d3`.

2. Find the partition containing the high end of the range.

`YEAR('1999-06-21')` [1152] evaluates to 1999, which is found in partition `d5`.

3. Scan only these two partitions and any partitions that may lie between them.

In this case, this means that only partitions `d3`, `d4`, and `d5` are scanned. The remaining partitions may be safely ignored (and are ignored).



Important

Invalid `DATE` and `DATETIME` values referenced in the `WHERE` clause of a query on a partitioned table are treated as `NULL`. This means that a query such as `SELECT * FROM partitioned_table WHERE date_column < '2008-12-00'` does not return any values (see Bug #40972).

So far, we have looked only at examples using `RANGE` partitioning, but pruning can be applied with other partitioning types as well.

Consider a table that is partitioned by `LIST`, where the partitioning expression is increasing or decreasing, such as the table `t3` shown here. (In this example, we assume for the sake of brevity that the `region_code` column is limited to values between 1 and 10 inclusive.)

```

CREATE TABLE t3 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY LIST(region_code) (
  PARTITION r0 VALUES IN (1, 3),
  PARTITION r1 VALUES IN (2, 5, 8),
  PARTITION r2 VALUES IN (4, 9),
  PARTITION r3 VALUES IN (6, 7, 10)
);

```

For a query such as `SELECT * FROM t3 WHERE region_code BETWEEN 1 AND 3`, the optimizer determines in which partitions the values 1, 2, and 3 are found (`r0` and `r1`) and skips the remaining ones (`r2` and `r3`).

For tables that are partitioned by `HASH` or `KEY`, partition pruning is also possible in cases in which the `WHERE` clause uses a simple `=` relation against a column used in the partitioning expression. Consider a table created like this:

```
CREATE TABLE t4 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY KEY(region_code)
PARTITIONS 8;
```

Any query, such as this one, that compares a column value with a constant can be pruned:

```
SELECT * FROM t4 WHERE region_code = 7;
```

Pruning can also be employed for short ranges, because the optimizer can turn such conditions into `IN` relations. For example, using the same table `t4` as defined previously, queries such as these can be pruned:

```
SELECT * FROM t4 WHERE region_code > 2 AND region_code < 6;
SELECT * FROM t4 WHERE region_code BETWEEN 3 AND 5;
```

In both these cases, the `WHERE` clause is transformed by the optimizer into `WHERE region_code IN (3, 4, 5)`.



Important

This optimization is used only if the range size is smaller than the number of partitions. Consider this query:

```
SELECT * FROM t4 WHERE region_code BETWEEN 4 AND 12;
```

The range in the `WHERE` clause covers 9 values (4, 5, 6, 7, 8, 9, 10, 11, 12), but `t4` has only 8 partitions. This means that the previous query cannot be pruned.

Pruning can be used only on integer columns of tables partitioned by `HASH` or `KEY`. For example, this query on table `t4` cannot use pruning because `dob` is a `DATE` column:

```
SELECT * FROM t4 WHERE dob >= '2001-04-14' AND dob <= '2005-10-15';
```

However, if the table stores year values in an `INT` column, then a query having `WHERE year_col >= 2001 AND year_col <= 2005` can be pruned.



Note

In MySQL 5.1, a query against a table partitioned by `KEY` and having a composite partitioning key could be pruned only if the query's `WHERE` clause compared every column in the key to a constant. In MySQL 5.5, it is possible to prune queries against such tables even if the `WHERE` clause does not reference every column in the partitioning key.

18.5. Restrictions and Limitations on Partitioning

This section discusses current restrictions and limitations on MySQL partitioning support.

Prohibited constructs. The following constructs are not permitted in partitioning expressions:

- Stored procedures, stored functions, UDFs, or plugins.
- Declared variables or user variables.

For a list of SQL functions which are permitted in partitioning expressions, see [Section 18.5.3, “Partitioning Limitations Relating to Functions”](#).

Arithmetic and logical operators. Use of the arithmetic operators `+` [1121], `-` [1122], and `*` [1122] is permitted in partitioning expressions. However, the result must be an integer value or `NULL` (except in the case of `[LINEAR] KEY` partitioning, as discussed elsewhere in this chapter; see [Section 18.2, “Partitioning Types”](#), for more information).

The `DIV` [1122] operator is also supported, and the `/` [1122] operator is not permitted. (Bug #30188, Bug #33182)

The bit operators `|` [1182], `&` [1182], `^` [1182], `<<` [1183], `>>` [1183], and `~` [1183] are not permitted in partitioning expressions.

Server SQL mode. Tables employing user-defined partitioning do not preserve the SQL mode in effect at the time that they were created. As discussed in [Section 5.1.7, “Server SQL Modes”](#), the results of many MySQL functions and operators may change according to the server SQL mode. Therefore, a change in the SQL mode at any time after the creation of partitioned tables may lead to major changes in the behavior of such tables, and could easily lead to corruption or loss of data. For these reasons, *it is strongly recommended that you never change the server SQL mode after creating partitioned tables.*

Examples. The following examples illustrate some changes in behavior of partitioned tables due to a change in the server SQL mode:

1. **Error handling.** Suppose that you create a partitioned table whose partitioning expression is one such as `column DIV 0` or `column MOD 0`, as shown here:

```
mysql> CREATE TABLE tn (c1 INT)
->     PARTITION BY LIST(1 DIV c1) (
->         PARTITION p0 VALUES IN (NULL),
->         PARTITION p1 VALUES IN (1)
-> );
Query OK, 0 rows affected (0.05 sec)
```

The default behavior for MySQL is to return `NULL` for the result of a division by zero, without producing any errors:

```
mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
|             |
+-----+
1 row in set (0.00 sec)

mysql> INSERT INTO tn VALUES (NULL), (0), (1);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

However, changing the server SQL mode to treat division by zero as an error and to enforce strict error handling causes the same `INSERT` statement to fail, as shown here:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tn VALUES (NULL), (0), (1);
```

```
ERROR 1365 (22012): Division by 0
```

2. **Table accessibility.** Sometimes a change in the server SQL mode can make partitioned tables unusable. The following `CREATE TABLE` statement can be executed successfully only if the `NO_UNSIGNED_SUBTRACTION` [614] mode is in effect:

```
mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
|           |
+-----+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
-> PARTITION BY RANGE(c1 - 10) (
-> PARTITION p0 VALUES LESS THAN (-5),
-> PARTITION p1 VALUES LESS THAN (0),
-> PARTITION p2 VALUES LESS THAN (5),
-> PARTITION p3 VALUES LESS THAN (10),
-> PARTITION p4 VALUES LESS THAN (MAXVALUE)
-> );
ERROR 1563 (HY000): Partition constant is out of partition function domain

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
| NO_UNSIGNED_SUBTRACTION |
+-----+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
-> PARTITION BY RANGE(c1 - 10) (
-> PARTITION p0 VALUES LESS THAN (-5),
-> PARTITION p1 VALUES LESS THAN (0),
-> PARTITION p2 VALUES LESS THAN (5),
-> PARTITION p3 VALUES LESS THAN (10),
-> PARTITION p4 VALUES LESS THAN (MAXVALUE)
-> );
Query OK, 0 rows affected (0.05 sec)
```

If you remove the `NO_UNSIGNED_SUBTRACTION` [614] server SQL mode after creating `tu`, you may no longer be able to access this table:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tu;
ERROR 1563 (HY000): Partition constant is out of partition function domain
mysql> INSERT INTO tu VALUES (20);
ERROR 1563 (HY000): Partition constant is out of partition function domain
```

Server SQL modes also impact replication of partitioned tables. Differing SQL modes on master and slave can lead to partitioning expressions being evaluated differently; this can cause the distribution of data among partitions to be different in the master's and slave's copies of a given table, and may even cause inserts into partitioned tables that succeed on the master to fail on the slave. For best results, you should always use the same server SQL mode on the master and on the slave.

Performance considerations. Some affects of partitioning operations on performance are given in the following list:

- **File system operations.** Partitioning and repartitioning operations (such as `ALTER TABLE` with `PARTITION BY ...`, `REORGANIZE PARTITIONS`, or `REMOVE PARTITIONING`) depend on file system operations for their implementation. This means that the speed of these operations is affected by such factors as file system type and characteristics, disk speed, swap space, file handling efficiency of the operating system, and MySQL server options and variables that relate to file handling. In particular, you should make sure that `large_files_support` [511] is enabled and that `open_files_limit` [535] is set properly. For partitioned tables using the `MyISAM` storage engine, increasing `myisam_max_sort_file_size` [528] may improve performance; partitioning and repartitioning operations involving `InnoDB` tables may be made more efficient by enabling `innodb_file_per_table` [1555].

See also [Maximum number of partitions](#).
- **Table locks.** The process executing a partitioning operation on a table takes a write lock on the table. Reads from such tables are relatively unaffected; pending `INSERT` and `UPDATE` operations are performed as soon as the partitioning operation has completed.
- **Storage engine.** Partitioning operations, queries, and update operations generally tend to be faster with `MyISAM` tables than with `InnoDB` or `NDB` tables.
- **Use of indexes and partition pruning.** As with nonpartitioned tables, proper use of indexes can speed up queries on partitioned tables significantly. In addition, designing partitioned tables and queries on these tables to take advantage of *partition pruning* can improve performance dramatically. See [Section 18.4, “Partition Pruning”](#), for more information.
- **Performance with `LOAD DATA`.** In MySQL 5.5, `LOAD DATA` uses buffering to improve performance. You should be aware that the buffer uses 130 KB memory per partition to achieve this.

Maximum number of partitions.

The maximum possible number of partitions for a given table (that does not use the `NDB` storage engine) is 1024. This number includes subpartitions.

The maximum possible number of user-defined partitions for a table using the `NDBCLUSTER` storage engine is determined according to the version of the MySQL Cluster software being used, the number of data nodes, and other factors. See [NDB and user-defined partitioning](#), for more information.

If, when creating tables with a large number of partitions (but less than the maximum), you encounter an error message such as `Got error ... from storage engine: Out of resources when opening file`, you may be able to address the issue by increasing the value of the `open_files_limit` [535] system variable. However, this is dependent on the operating system, and may not be possible or advisable on all platforms; see [Section C.5.2.18, “'File' Not Found and Similar Errors”](#), for more information. In some cases, using large numbers (hundreds) of partitions may also not be advisable due to other concerns, so using more partitions does not automatically lead to better results.

See also [File system operations](#).

Query cache not supported.

The query cache is not supported for partitioned tables. Beginning with MySQL 5.5.23, the query cache is automatically disabled for queries involving partitioned tables, and cannot be enabled for such queries. (Bug #53775)

Per-partition key caches.

In MySQL 5.5, key caches are supported for partitioned `MyISAM` tables, using the `CACHE INDEX` and `LOAD INDEX INTO CACHE` statements. Key caches may be defined for one, several, or all partitions, and indexes for one, several, or all partitions may be preloaded into key caches.

Foreign keys not supported.

Partitioned tables do not support foreign keys. More specifically, this means that the following two statements are true:

1. Definitions of tables employing user-defined partitioning may not contain foreign key references to other tables.
2. No table definition may contain a foreign key reference to a partitioned table.

The scope of these restrictions includes tables that use the [InnoDB](#) storage engine.

ALTER TABLE ... ORDER BY. An `ALTER TABLE ... ORDER BY column` statement run against a partitioned table causes ordering of rows only within each partition.

FULLTEXT indexes.

Partitioned tables do not support [FULLTEXT](#) indexes or searches. This includes partitioned tables employing the [MyISAM](#) storage engine.

Spatial columns. Columns with spatial data types such as [POINT](#) or [GEOMETRY](#) cannot be used in partitioned tables.

Temporary tables.

Temporary tables cannot be partitioned. (Bug #17497)

Log tables. It is not possible to partition the log tables; an `ALTER TABLE ... PARTITION BY ...` statement on such a table fails with an error.

Data type of partitioning key.

A partitioning key must be either an integer column or an expression that resolves to an integer. The column or expression value may also be [NULL](#). (See [Section 18.2.7, “How MySQL Partitioning Handles NULL”](#).)

There are two exceptions to this restriction:

1. When partitioning by [\[LINEAR\] KEY](#), it is possible to use columns of other types as partitioning keys, because MySQL's internal key-hashing functions produce the correct data type from these types. For example, the following `CREATE TABLE` statement is valid:

```
CREATE TABLE tkc (c1 CHAR)
PARTITION BY KEY(c1)
PARTITIONS 4;
```

2. When partitioning by [RANGE COLUMNS](#) or [LIST COLUMNS](#), it is possible to use string, [DATE](#), and [DATETIME](#) columns. For example, each of the following `CREATE TABLE` statements is valid:

```
CREATE TABLE rc (c1 INT, c2 DATE)
PARTITION BY RANGE COLUMNS(c2) (
    PARTITION p0 VALUES LESS THAN('1990-01-01'),
    PARTITION p1 VALUES LESS THAN('1995-01-01'),
    PARTITION p2 VALUES LESS THAN('2000-01-01'),
    PARTITION p3 VALUES LESS THAN('2005-01-01'),
    PARTITION p4 VALUES LESS THAN(MAXVALUE)
);

CREATE TABLE lc (c1 INT, c2 CHAR(1))
PARTITION BY LIST COLUMNS(c2) (
    PARTITION p0 VALUES IN('a', 'd', 'g', 'j', 'm', 'p', 's', 'v', 'y'),
    PARTITION p1 VALUES IN('b', 'e', 'h', 'k', 'n', 'q', 't', 'w', 'z'),
    PARTITION p2 VALUES IN('c', 'f', 'i', 'l', 'o', 'r', 'u', 'x', NULL)
);
```

Neither of the preceding exceptions applies to [BLOB](#) or [TEXT](#) column types.

Subqueries.

A partitioning key may not be a subquery, even if that subquery resolves to an integer value or `NULL`.

Issues with subpartitions.

Subpartitions must use `HASH` or `KEY` partitioning. Only `RANGE` and `LIST` partitions may be subpartitioned; `HASH` and `KEY` partitions cannot be subpartitioned.

Currently, `SUBPARTITION BY KEY` requires that the subpartitioning column or columns be specified explicitly, unlike the case with `PARTITION BY KEY`, where it can be omitted (in which case the table's primary key column is used by default). Consider the table created by this statement:

```
CREATE TABLE ts (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(30)
);
```

You can create a table having the same columns, partitioned by `KEY`, using a statement such as this one:

```
CREATE TABLE ts (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(30)
)
PARTITION BY KEY()
PARTITIONS 4;
```

The previous statement is treated as though it had been written like this, with the table's primary key column used as the partitioning column:

```
CREATE TABLE ts (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(30)
)
PARTITION BY KEY(id)
PARTITIONS 4;
```

However, the following statement that attempts to create a subpartitioned table using the default column as the subpartitioning column fails, and the column must be specified for the statement to succeed, as shown here:

```
mysql> CREATE TABLE ts (
->   id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
->   name VARCHAR(30)
-> )
-> PARTITION BY RANGE(id)
-> SUBPARTITION BY KEY()
-> SUBPARTITIONS 4
-> (
->   PARTITION p0 VALUES LESS THAN (100),
->   PARTITION p1 VALUES LESS THAN (MAXVALUE)
-> );
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near ')'

mysql> CREATE TABLE ts (
->   id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
->   name VARCHAR(30)
-> )
-> PARTITION BY RANGE(id)
-> SUBPARTITION BY KEY(id)
-> SUBPARTITIONS 4
-> (
```

```

-> PARTITION p0 VALUES LESS THAN (100),
-> PARTITION p1 VALUES LESS THAN (MAXVALUE)
-> );
Query OK, 0 rows affected (0.07 sec)

```

This is a known issue (see Bug #51470).

DELAYED option not supported. Use of `INSERT DELAYED` to insert rows into a partitioned table is not supported. Attempting to do so fails with an error.

DATA DIRECTORY and INDEX DIRECTORY options. `DATA DIRECTORY` and `INDEX DIRECTORY` are subject to the following restrictions when used with partitioned tables:

- Table-level `DATA DIRECTORY` and `INDEX DIRECTORY` options are ignored (see Bug #32091).
- On Windows, the `DATA DIRECTORY` and `INDEX DIRECTORY` options are not supported for individual partitions or subpartitions (Bug #30459).

Repairing and rebuilding partitioned tables. The statements `CHECK TABLE`, `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `REPAIR TABLE` are supported for partitioned tables.

In addition, you can use `ALTER TABLE ... REBUILD PARTITION` to rebuild one or more partitions of a partitioned table; `ALTER TABLE ... REORGANIZE PARTITION` also causes partitions to be rebuilt. See [Section 13.1.7, “ALTER TABLE Syntax”](#), for more information about these two statements.

`mysqlcheck` and `myisamchk` are not supported with partitioned tables.

18.5.1. Partitioning Keys, Primary Keys, and Unique Keys

This section discusses the relationship of partitioning keys with primary keys and unique keys. The rule governing this relationship can be expressed as follows: All columns used in the partitioning expression for a partitioned table must be part of every unique key that the table may have.

In other words, *every unique key on the table must use every column in the table's partitioning expression.* (This also includes the table's primary key, since it is by definition a unique key. This particular case is discussed later in this section.) For example, each of the following table creation statements is invalid:

```

CREATE TABLE t1 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1),
  UNIQUE KEY (col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;

```

In each case, the proposed table would have at least one unique key that does not include all columns used in the partitioning expression.

Each of the following statements is valid, and represents one way in which the corresponding invalid table creation statement could be made to work:

```
CREATE TABLE t1 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1, col2, col3)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1, col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

This example shows the error produced in such cases:

```
mysql> CREATE TABLE t3 (
->   col1 INT NOT NULL,
->   col2 DATE NOT NULL,
->   col3 INT NOT NULL,
->   col4 INT NOT NULL,
->   UNIQUE KEY (col1, col2),
->   UNIQUE KEY (col3)
-> )
-> PARTITION BY HASH(col1 + col3)
-> PARTITIONS 4;
ERROR 1491 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

The `CREATE TABLE` statement fails because both `col1` and `col3` are included in the proposed partitioning key, but neither of these columns is part of both of unique keys on the table. This shows one possible fix for the invalid table definition:

```
mysql> CREATE TABLE t3 (
->   col1 INT NOT NULL,
->   col2 DATE NOT NULL,
->   col3 INT NOT NULL,
->   col4 INT NOT NULL,
->   UNIQUE KEY (col1, col2, col3),
->   UNIQUE KEY (col3)
-> )
-> PARTITION BY HASH(col3)
-> PARTITIONS 4;
Query OK, 0 rows affected (0.05 sec)
```

In this case, the proposed partitioning key `col3` is part of both unique keys, and the table creation statement succeeds.

The following table cannot be partitioned at all, because there is no way to include in a partitioning key any columns that belong to both unique keys:

```
CREATE TABLE t4 (
  col1 INT NOT NULL,
  col2 INT NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1, col3),
```

```

        UNIQUE KEY (col2, col4)
    );

```

Since every primary key is by definition a unique key, this restriction also includes the table's primary key, if it has one. For example, the next two statements are invalid:

```

CREATE TABLE t5 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t6 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col3),
    UNIQUE KEY(col2)
)
PARTITION BY HASH( YEAR(col2) )
PARTITIONS 4;

```

In both cases, the primary key does not include all columns referenced in the partitioning expression. However, both of the next two statements are valid:

```

CREATE TABLE t7 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;

CREATE TABLE t8 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2, col4),
    UNIQUE KEY(col2, col1)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;

```

If a table has no unique keys—this includes having no primary key—then this restriction does not apply, and you may use any column or columns in the partitioning expression as long as the column type is compatible with the partitioning type.

For the same reason, you cannot later add a unique key to a partitioned table unless the key includes all columns used by the table's partitioning expression. Consider the partitioned table created as shown here:

```

mysql> CREATE TABLE t_no_pk (c1 INT, c2 INT)
->     PARTITION BY RANGE(c1) (
->         PARTITION p0 VALUES LESS THAN (10),
->         PARTITION p1 VALUES LESS THAN (20),
->         PARTITION p2 VALUES LESS THAN (30),
->         PARTITION p3 VALUES LESS THAN (40)
->     );

```

```
Query OK, 0 rows affected (0.12 sec)
```

It is possible to add a primary key to `t_no_pk` using either of these `ALTER TABLE` statements:

```
# possible PK
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1);
Query OK, 0 rows affected (0.13 sec)
Records: 0 Duplicates: 0 Warnings: 0

# drop this PK
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;
Query OK, 0 rows affected (0.10 sec)
Records: 0 Duplicates: 0 Warnings: 0

# use another possible PK
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1, c2);
Query OK, 0 rows affected (0.12 sec)
Records: 0 Duplicates: 0 Warnings: 0

# drop this PK
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;
Query OK, 0 rows affected (0.09 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

However, the next statement fails, because `c1` is part of the partitioning key, but is not part of the proposed primary key:

```
# fails with error 1503
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c2);
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

Since `t_no_pk` has only `c1` in its partitioning expression, attempting to adding a unique key on `c2` alone fails. However, you can add a unique key that uses both `c1` and `c2`.

These rules also apply to existing nonpartitioned tables that you wish to partition using `ALTER TABLE ... PARTITION BY`. Consider a table `np_pk` created as shown here:

```
mysql> CREATE TABLE np_pk (
->   id INT NOT NULL AUTO_INCREMENT,
->   name VARCHAR(50),
->   added DATE,
->   PRIMARY KEY (id)
-> );
Query OK, 0 rows affected (0.08 sec)
```

The following `ALTER TABLE` statement fails with an error, because the `added` column is not part of any unique key in the table:

```
mysql> ALTER TABLE np_pk
->   PARTITION BY HASH( TO_DAYS(added) )
->   PARTITIONS 4;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

However, this statement using the `id` column for the partitioning column is valid, as shown here:

```
mysql> ALTER TABLE np_pk
->   PARTITION BY HASH(id)
->   PARTITIONS 4;
Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

In the case of `np_pk`, the only column that may be used as part of a partitioning expression is `id`; if you wish to partition this table using any other column or columns in the partitioning expression, you must first

modify the table, either by adding the desired column or columns to the primary key, or by dropping the primary key altogether.

18.5.2. Partitioning Limitations Relating to Storage Engines

The following limitations apply to the use of storage engines with user-defined partitioning of tables.

MERGE storage engine. User-defined partitioning and the `MERGE` storage engine are not compatible. Tables using the `MERGE` storage engine cannot be partitioned. Partitioned tables cannot be merged.

FEDERATED storage engine. Partitioning of `FEDERATED` tables is not supported; it is not possible to create partitioned `FEDERATED` tables.

CSV storage engine. Partitioned tables using the `CSV` storage engine are not supported; it is not possible to create partitioned `CSV` tables.

InnoDB storage engine. `ALTER TABLE ... OPTIMIZE PARTITION` does not work correctly with partitioned tables that use the `InnoDB` storage engine. Use `ALTER TABLE ... REBUILD PARTITION` and `ALTER TABLE ... ANALYZE PARTITION` instead. For more information, see [Section 13.1.7.1, “ALTER TABLE Partition Operations”](#).

NDBCLUSTER storage engine (MySQL Cluster). Partitioning by `KEY` (including `LINEAR KEY`) is the only type of partitioning supported for the `NDBCLUSTER` storage engine. It is not possible in MySQL Cluster NDB 7.2 to create a MySQL Cluster table using any partitioning type other than `[LINEAR] KEY`, and attempting to do so fails with an error.

In addition, the maximum number of partitions that can be defined for an `NDBCLUSTER` table depends on the number of data nodes and node groups in the cluster, the version of the MySQL Cluster software in use, and other factors. See [NDB and user-defined partitioning](#), for more information.

`CREATE TABLE` and `ALTER TABLE` statements that would cause a user-partitioned `NDBCLUSTER` table not to meet either or both of the following two requirements are not permitted, and fail with an error:

1. The table must have an explicit primary key.
2. All columns listed in the table's partitioning expression must be part of the primary key.

Exception. If a user-partitioned `NDBCLUSTER` table is created using an empty column-list (that is, using `PARTITION BY KEY()` or `PARTITION BY LINEAR KEY()`), then no explicit primary key is required.

Upgrading partitioned tables. When performing an upgrade, tables which are partitioned by `KEY` and which use any storage engine other than `NDBCLUSTER` must be dumped and reloaded.

Same storage engine for all partitions. All partitions of a partitioned table must use the same storage engine and it must be the same storage engine used by the table as a whole. In addition, if one does not specify an engine on the table level, then one must do either of the following when creating or altering a partitioned table:

- Do *not* specify any engine for *any* partition or subpartition
- Specify the engine for *all* partitions or subpartitions

18.5.3. Partitioning Limitations Relating to Functions

This section discusses limitations in MySQL Partitioning relating specifically to functions used in partitioning expressions.

Only the MySQL functions shown in the following table are supported in partitioning expressions.

ABS() [1123]	CEILING() [1125] (see CEILING() and FLOOR())	DAY() [1139]
DAYOFMONTH() [1140]	DAYOFWEEK() [1140]	DAYOFYEAR() [1140]
DATEDIFF() [1135]	EXTRACT() [1140] (see EXTRACT() function with WEEK specifier)	FLOOR() [1126] (see CEILING() and FLOOR())
HOUR() [1142]	MICROSECOND() [1142]	MINUTE() [1142]
MOD() [1127]	MONTH() [1143]	QUARTER() [1144]
SECOND() [1144]	TIME_TO_SEC() [1148]	TO_DAYS() [1148]
TO_SECONDS() [1149] (implemented in MySQL 5.5.0)	UNIX_TIMESTAMP() [1150] (permitted beginning with MySQL 5.5.1 and fully supported beginning with MySQL 5.5.15, with TIMESTAMP columns)	WEEKDAY() [1152]
YEAR() [1152]		YEARWEEK() [1152]

In MySQL 5.5, partition pruning is supported for the [TO_DAYS\(\) \[1148\]](#), [TO_SECONDS\(\) \[1149\]](#), and [YEAR\(\) \[1152\]](#) functions. In addition, beginning with MySQL 5.5.15, [UNIX_TIMESTAMP\(\) \[1150\]](#) is treated as monotonic in partitioning expressions. See [Section 18.4, “Partition Pruning”](#), for more information.

CEILING() and FLOOR(). Each of these functions returns an integer only if it is passed an argument of an exact numeric type, such as one of the [INT](#) types or [DECIMAL](#). This means, for example, that the following [CREATE TABLE](#) statement fails with an error, as shown here:

```
mysql> CREATE TABLE t (c FLOAT) PARTITION BY LIST( FLOOR(c) )(
->     PARTITION p0 VALUES IN (1,3,5),
->     PARTITION p1 VALUES IN (2,4,6)
-> );
ERROR 1490 (HY000): The PARTITION function returns the wrong type
```

EXTRACT() function with WEEK specifier. The value returned by the [EXTRACT\(\) \[1140\]](#) function, when used as [EXTRACT\(WEEK FROM col\) \[1140\]](#), depends on the value of the [default_week_format \[494\]](#) system variable. For this reason, beginning with MySQL 5.5.9, [EXTRACT\(\) \[1140\]](#) is longer permitted as a partitioning function when it specifies the unit as [WEEK](#). (Bug #54483)

See [Section 12.6.2, “Mathematical Functions”](#), for more information about the return types of these functions, as well as [Section 11.2, “Numeric Types”](#).

18.5.4. Partitioning and Table-Level Locking

For storage engines such as [MyISAM](#) that actually execute table-level locks when executing DML or DDL statements, such a statement affecting a partitioned table imposes a lock on the table as a whole; that is, all partitions are locked until the statement was finished. For example, a [SELECT](#) from a partitioned [MyISAM](#) table causes a lock on the entire table.

In practical terms, what this means is that the statements discussed later in this section tend to execute more slowly as the number of partitions increases. This limitation is greatly reduced in MySQL 5.6, with the introduction of *partition lock pruning* in MySQL 5.6.6.

This is not true for statements effecting partitioned tables using storage engines such as [InnoDB](#), that employ row-level locking and do not actually perform (or need to perform) the locks prior to partition pruning.

The next few paragraphs discuss the effects of MySQL statements on partitioned tables using storage engines that employ table-level locks.

DML statements

`SELECT` statements lock the entire table. `SELECT` statements containing unions or joins lock all tables named in the union or join.

`UPDATE` also locks the entire table.

`REPLACE` and `INSERT` (including `INSERT ... ON DUPLICATE KEY UPDATE`) lock the entire table.

`INSERT ... SELECT` locks both the source table and the target table.



Note

`INSERT DELAYED` is not supported for partitioned tables.

A `LOAD DATA` statement on a partitioned table locks the entire table.

A trigger on a partitioned table, once activated, locks the entire table.

DDL statements

`CREATE VIEW` causes a lock on any partitioned table from which it reads.

`ALTER TABLE` locks the affected partitioned table.

Other statements

`LOCK TABLES` locks all partitions of a partitioned table.

Evaluating the `expr` in a `CALL stored_procedure(expr)` statement locks all partitions of any partitioned table referenced by `expr`.

`ALTER TABLE` also takes a metadata lock on the table level.

Chapter 19. Stored Programs and Views

Table of Contents

19.1. Defining Stored Programs	2442
19.2. Using Stored Routines (Procedures and Functions)	2443
19.2.1. Stored Routine Syntax	2444
19.2.2. Stored Routines and MySQL Privileges	2444
19.2.3. Stored Routine Metadata	2445
19.2.4. Stored Procedures, Functions, Triggers, and <code>LAST_INSERT_ID()</code>	2445
19.3. Using Triggers	2445
19.3.1. Trigger Syntax	2446
19.3.2. Trigger Metadata	2448
19.4. Using the Event Scheduler	2448
19.4.1. Event Scheduler Overview	2449
19.4.2. Event Scheduler Configuration	2450
19.4.3. Event Syntax	2452
19.4.4. Event Metadata	2452
19.4.5. Event Scheduler Status	2453
19.4.6. The Event Scheduler and MySQL Privileges	2454
19.5. Using Views	2456
19.5.1. View Syntax	2457
19.5.2. View Processing Algorithms	2457
19.5.3. Updatable and Insertable Views	2459
19.5.4. View Metadata	2461
19.6. Access Control for Stored Programs and Views	2461
19.7. Binary Logging of Stored Programs	2462

This chapter discusses stored programs and views, which are database objects defined in terms of SQL code that is stored on the server for later execution.

Stored programs include these objects:

- Stored routines, that is, stored procedures and functions. A stored procedure is invoked using the `CALL` statement. A procedure does not have a return value but can modify its parameters for later inspection by the caller. It can also generate result sets to be returned to the client program. A stored function is used much like a built-in function. you invoke it in an expression and it returns a value during expression evaluation.
- Triggers. A trigger is a named database object that is associated with a table and that is activated when a particular event occurs for the table, such as an insert or update.
- Events. An event is a task that the server runs according to schedule.

Views are stored queries that when referenced produce a result set. A view acts as a virtual table.

This chapter describes how to use stored programs and views. The following sections provide additional information about SQL syntax for statements related to these objects:

- For each object type, there are `CREATE`, `ALTER`, and `DROP` statements that control which objects exist and how they are defined. See [Section 13.1, “Data Definition Statements”](#).

- The `CALL` statement is used to invoke stored procedures. See [Section 13.2.1, “CALL Syntax”](#).
- Stored program definitions include a body that may use compound statements, loops, conditionals, and declared variables. See [Section 13.6, “MySQL Compound-Statement Syntax”](#).

19.1. Defining Stored Programs

Each stored program contains a body that consists of an SQL statement. This statement may be a compound statement made up of several statements separated by semicolon (`;`) characters. For example, the following stored procedure has a body made up of a `BEGIN ... END` block that contains a `SET` statement and a `REPEAT` loop that itself contains another `SET` statement:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END;
```

If you use the `mysql` client program to define a stored program containing semicolon characters, a problem arises. By default, `mysql` itself recognizes the semicolon as a statement delimiter, so you must redefine the delimiter temporarily to cause `mysql` to pass the entire stored program definition to the server.

To redefine the `mysql` delimiter, use the `delimiter` command. The following example shows how to do this for the `dorepeat()` procedure just shown. The delimiter is changed to `//` to enable the entire definition to be passed to the server as a single statement, and then restored to `;` before invoking the procedure. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself.

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
->   SET @x = 0;
->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+-----+
| @x    |
+-----+
| 1001  |
+-----+
1 row in set (0.00 sec)
```

You can redefine the delimiter to a string other than `//`, and the delimiter can consist of a single character or multiple characters. You should avoid the use of the backslash (“`\`”) character because that is the escape character for MySQL.

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
```

```
-> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world! |
+-----+
1 row in set (0.00 sec)
```

19.2. Using Stored Routines (Procedures and Functions)

Stored routines (procedures and functions) are supported in MySQL 5.5. A stored routine is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored routine instead.

Stored routines require the `proc` table in the `mysql` database. This table is created during the MySQL 5.5 installation procedure. If you are upgrading to MySQL 5.5 from an earlier version, be sure to update your grant tables to make sure that the `proc` table exists. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

Stored routines can be particularly useful in certain situations:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures and functions for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Stored routines can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does increase the load on the database server because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored routines also enable you to have libraries of functions in the database server. This is a feature shared by modern application languages that enable such design internally (for example, by using classes). Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2. All syntax described here is supported and any limitations and extensions are documented where appropriate.

Additional Resources

- You may find the [Stored Procedures User Forum](#) of use when working with stored procedures and functions.
- For answers to some commonly asked questions regarding stored routines in MySQL, see [Section B.4, “MySQL 5.5 FAQ: Stored Procedures and Functions”](#).
- There are some restrictions on the use of stored routines. See [Section E.1, “Restrictions on Stored Programs”](#).
- Binary logging for stored routines takes place as described in [Section 19.7, “Binary Logging of Stored Programs”](#).

19.2.1. Stored Routine Syntax

A stored routine is either a procedure or a function. Stored routines are created with the `CREATE PROCEDURE` and `CREATE FUNCTION` statements (see [Section 13.1.15](#), “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”). A procedure is invoked using a `CALL` statement (see [Section 13.2.1](#), “`CALL` Syntax”), and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. The body of a stored routine can use compound statements (see [Section 13.6](#), “`MySQL Compound-Statement Syntax`”).

Stored routines can be dropped with the `DROP PROCEDURE` and `DROP FUNCTION` statements (see [Section 13.1.26](#), “`DROP PROCEDURE` and `DROP FUNCTION` Syntax”), and altered with the `ALTER PROCEDURE` and `ALTER FUNCTION` statements (see [Section 13.1.5](#), “`ALTER PROCEDURE` Syntax”).

A stored procedure or function is associated with a particular database. This has several implications:

- When the routine is invoked, an implicit `USE db_name` is performed (and undone when the routine terminates). `USE` statements within stored routines are not permitted.
- You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure `p` or function `f` that is associated with the `test` database, you can say `CALL test.p()` or `test.f()`.
- When a database is dropped, all stored routines associated with it are dropped as well.

Stored functions cannot be recursive.

Recursion in stored procedures is permitted but disabled by default. To enable recursion, set the `max_sp_recursion_depth` [525] server system variable to a value greater than zero. Stored procedure recursion increases the demand on thread stack space. If you increase the value of `max_sp_recursion_depth` [525], it may be necessary to increase thread stack size by increasing the value of `thread_stack` [566] at server startup. See [Section 5.1.4](#), “`Server System Variables`”, for more information.

MySQL supports a very useful extension that enables the use of regular `SELECT` statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple `SELECT` statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1. The client should also specify the `CLIENT_MULTI_RESULTS` option when it connects. For C programs, this can be done with the `mysql_real_connect()` C API function. See [Section 22.8.3.52](#), “`mysql_real_connect()`”, and [Section 22.8.13](#), “`C API Support for Multiple Statement Execution`”.

19.2.2. Stored Routines and MySQL Privileges

The MySQL grant system takes stored routines into account as follows:

- The `CREATE ROUTINE` [695] privilege is needed to create stored routines.
- The `ALTER ROUTINE` [695] privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine if necessary, and dropped from the creator when the routine is dropped.
- The `EXECUTE` [696] privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a routine if necessary (and dropped from the creator when the routine is

dropped). Also, the default `SQL SECURITY` characteristic for a routine is `DEFINER`, which enables users who have access to the database with which the routine is associated to execute the routine.

- If the `automatic_sp_privileges` [484] system variable is 0, the `EXECUTE` [696] and `ALTER ROUTINE` [695] privileges are not automatically granted to and dropped from the routine creator.
- The creator of a routine is the account used to execute the `CREATE` statement for it. This might not be the same as the account named as the `DEFINER` in the routine definition.

The server manipulates the `mysql.proc` table in response to statements that create, alter, or drop stored routines. It is not supported that the server will notice manual manipulation of this table.

19.2.3. Stored Routine Metadata

Metadata about stored routines can be obtained as follows:

- Query the `ROUTINES` table of the `INFORMATION_SCHEMA` database. See [Section 20.17, “The INFORMATION_SCHEMA ROUTINES Table”](#).
- Use the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements to see routine definitions. See [Section 13.7.5.11, “SHOW CREATE PROCEDURE Syntax”](#).
- Use the `SHOW PROCEDURE STATUS` and `SHOW FUNCTION STATUS` statements to see routine characteristics. See [Section 13.7.5.29, “SHOW PROCEDURE STATUS Syntax”](#).

19.2.4. Stored Procedures, Functions, Triggers, and `LAST_INSERT_ID()`

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` [1194] changes the same way as for statements executed outside the body of these kinds of objects (see [Section 12.14, “Information Functions”](#)). The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` [1194] that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()` [1194], the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements do not see a changed value.

19.3. Using Triggers

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is defined to activate when an `INSERT`, `DELETE`, or `UPDATE` statement executes for the associated table. A trigger can be set to activate either before or after the triggering statement. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.



Important

MySQL triggers are activated by SQL statements *only*. They are not activated by changes in views, nor by changes to tables made by APIs that do not transmit SQL statements to the MySQL Server. This means that:

- Triggers are not activated by changes in `INFORMATION_SCHEMA` tables, because these tables are actually views.
- Triggers are not activated by updates made using the `NDB` API.

To use triggers if you have upgraded to MySQL 5.5 from an older release that did not support triggers, you should upgrade your grant tables so that they contain the trigger-related privileges. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

The following discussion describes the syntax for creating and dropping triggers, and shows some examples of how to use them.

Additional Resources

- You may find the [Triggers User Forum](#) of use when working with views.
- For answers to some commonly asked questions regarding triggers in MySQL, see [Section B.5, “MySQL 5.5 FAQ: Triggers”](#).
- There are some restrictions on the use of triggers; see [Section E.1, “Restrictions on Stored Programs”](#).
- Binary logging for triggers takes place as described in [Section 19.7, “Binary Logging of Stored Programs”](#).

19.3.1. Trigger Syntax

To create a trigger or drop a trigger, use the `CREATE TRIGGER` or `DROP TRIGGER` statement. The syntax for these statements is described in [Section 13.1.19, “CREATE TRIGGER Syntax”](#), and [Section 13.1.30, “DROP TRIGGER Syntax”](#).

Here is a simple example that associates a trigger with a table for `INSERT` statements. The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

The `CREATE TRIGGER` statement creates a trigger named `ins_sum` that is associated with the `account` table. It also includes clauses that specify the trigger activation time, the triggering event, and what to do with the trigger activates:

- The keyword `BEFORE` indicates the trigger action time. In this case, the trigger should activate before each row inserted into the table. The other permissible keyword here is `AFTER`.
- The keyword `INSERT` indicates the event that activates the trigger. In the example, `INSERT` statements cause trigger activation. You can also create triggers for `DELETE` and `UPDATE` statements.
- The statement following `FOR EACH ROW` defines the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering statement. In the example, the triggered statement is a simple `SET` that accumulates the values inserted into the `amount` column. The statement refers to the column as `NEW.amount` which means “the value of the `amount` column to be inserted into the new row.”

To use the trigger, set the accumulator variable to zero, execute an `INSERT` statement, and then see what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----+
| Total amount inserted |
+-----+
| 1852.48                |
+-----+
```

In this case, the value of `@sum` after the `INSERT` statement has executed is `14.98 + 1937.50 - 100`, or `1852.48`.

To destroy the trigger, use a `DROP TRIGGER` statement. You must specify the schema name if the trigger is not in the default schema:

```
mysql> DROP TRIGGER test.ins_sum;
```

Triggers for a table are also dropped if you drop the table.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

In addition to the requirement that trigger names be unique for a schema, there are other limitations on the types of triggers you can create. In particular, you cannot have two triggers for a table that have the same activation time and activation event. For example, you cannot define two `BEFORE INSERT` triggers or two `AFTER UPDATE` triggers for a table. This should rarely be a significant limitation, because it is possible to define a trigger that executes multiple statements by using the `BEGIN ... END` compound statement construct after `FOR EACH ROW`. (An example appears later in this section.)

The `OLD` and `NEW` keywords enable you to access columns in the rows affected by a trigger. (`OLD` and `NEW` are not case sensitive.) In an `INSERT` trigger, only `NEW.col_name` can be used; there is no old row. In a `DELETE` trigger, only `OLD.col_name` can be used; there is no new row. In an `UPDATE` trigger, you can use `OLD.col_name` to refer to the columns of a row before it is updated and `NEW.col_name` to refer to the columns of the row after it is updated.

A column named with `OLD` is read only. You can refer to it (if you have the `SELECT` privilege), but not modify it. A column named with `NEW` can be referred to if you have the `SELECT` [697] privilege for it. In a `BEFORE` trigger, you can also change its value with `SET NEW.col_name = value` if you have the `UPDATE` [697] privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or that are used to update a row.

In a `BEFORE` trigger, the `NEW` value for an `AUTO_INCREMENT` column is 0, not the automatically generated sequence number that will be generated when the new record actually is inserted.

`OLD` and `NEW` are MySQL extensions to triggers.

By using the `BEGIN ... END` construct, you can define a trigger that executes multiple statements. Within the `BEGIN` block, you also can use other syntax that is permitted within stored routines such as conditionals and loops. However, just as for stored routines, if you use the `mysql` program to define a trigger that executes multiple statements, it is necessary to redefine the `mysql` statement delimiter so that you can use the `;` statement delimiter within the trigger definition. The following example illustrates these points. It defines an `UPDATE` trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a `BEFORE` trigger because the value needs to be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
```

```
-> FOR EACH ROW
-> BEGIN
->     IF NEW.amount < 0 THEN
->         SET NEW.amount = 0;
->     ELSEIF NEW.amount > 100 THEN
->         SET NEW.amount = 100;
->     END IF;
-> END; //
mysql> delimiter ;
```

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple [CALL](#) statement. This is also advantageous if you want to invoke the same routine from within several triggers.

There are some limitations on what can appear in statements that a trigger executes when activated:

- The trigger cannot use the [CALL](#) statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are permitted to return data to the trigger through [OUT](#) or [INOUT](#) parameters.)
- The trigger cannot use statements that explicitly or implicitly begin or end a transaction such as [START TRANSACTION](#), [COMMIT](#), or [ROLLBACK](#).

MySQL handles errors during trigger execution as follows:

- If a [BEFORE](#) trigger fails, the operation on the corresponding row is not performed.
- A [BEFORE](#) trigger is activated by the *attempt* to insert or modify the row, regardless of whether the attempt subsequently succeeds.
- An [AFTER](#) trigger is executed only if the [BEFORE](#) trigger (if any) and the row operation both execute successfully.
- An error during either a [BEFORE](#) or [AFTER](#) trigger results in failure of the entire statement that caused trigger invocation.
- For transactional tables, failure of a statement should cause rollback of all changes performed by the statement. Failure of a trigger causes the statement to fail, so trigger failure also causes rollback. For nontransactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

19.3.2. Trigger Metadata

Metadata about triggers can be obtained as follows:

- Query the [TRIGGERS](#) table of the [INFORMATION_SCHEMA](#) database. See [Section 20.25](#), “[The INFORMATION_SCHEMA TRIGGERS Table](#)”.
- Use the [SHOW TRIGGERS](#) statement. See [Section 13.7.5.39](#), “[SHOW TRIGGERS Syntax](#)”.

19.4. Using the Event Scheduler

The *MySQL Event Scheduler* manages the scheduling and execution of events: Tasks that run according to schedule. The following discussion covers the Event Scheduler and is divided into the following sections:

- [Section 19.4.1](#), “[Event Scheduler Overview](#)”, provides an introduction to and conceptual overview of MySQL Events.

- [Section 19.4.3, “Event Syntax”](#), discusses the SQL statements for creating, altering, and dropping MySQL Events.
- [Section 19.4.4, “Event Metadata”](#), shows how to obtain information about events and how this information is stored by the MySQL Server.
- [Section 19.4.6, “The Event Scheduler and MySQL Privileges”](#), discusses the privileges required to work with events and the ramifications that events have with regard to privileges when executing.

Stored routines require the `event` table in the `mysql` database. This table is created during the MySQL 5.5 installation procedure. If you are upgrading to MySQL 5.5 from an earlier version, be sure to update your grant tables to make sure that the `event` table exists. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

Additional Resources

- You may find the [MySQL Event Scheduler User Forum](#) of use when working with scheduled events.
- There are some restrictions on the use of events; see [Section E.1, “Restrictions on Stored Programs”](#).
- Binary logging for events takes place as described in [Section 19.7, “Binary Logging of Stored Programs”](#).

19.4.1. Event Scheduler Overview

MySQL Events are tasks that run according to a schedule. Therefore, we sometimes refer to them as *scheduled* events. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix `crontab` (also known as a “cron job”) or the Windows Task Scheduler.

Scheduled tasks of this type are also sometimes known as “temporal triggers”, implying that these are objects that are triggered by the passage of time. While this is essentially correct, we prefer to use the term *events* to avoid confusion with triggers of the type discussed in [Section 19.3, “Using Triggers”](#). Events should more specifically not be confused with “temporary triggers”. Whereas a trigger is a database object whose statements are executed in response to a specific type of event that occurs on a given table, a (scheduled) event is an object whose statements are executed in response to the passage of a specified time interval.

While there is no provision in the SQL Standard for event scheduling, there are precedents in other database systems, and you may notice some similarities between these implementations and that found in the MySQL Server.

MySQL Events have the following major features and properties:

- In MySQL 5.5, an event is uniquely identified by its name and the schema to which it is assigned.
- An event performs a specific action according to a schedule. This action consists of an SQL statement, which can be a compound statement in a `BEGIN . . . END` block if desired (see [Section 13.6, “MySQL Compound-Statement Syntax”](#)). An event's timing can be either *one-time* or *recurrent*. A one-time event executes one time only. A recurrent event repeats its action at a regular interval, and the schedule for a recurring event can be assigned a specific start day and time, end day and time, both, or neither. (By default, a recurring event's schedule begins as soon as it is created, and continues indefinitely, until it is disabled or dropped.)

If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent

simultaneous instances. For example, you could use the `GET_LOCK()` [1199] function, or row or table locking.

- Users can create, modify, and drop scheduled events using SQL statements intended for these purposes. Syntactically invalid event creation and modification statements fail with an appropriate error message. *A user may include statements in an event's action which require privileges that the user does not actually have.* The event creation or modification statement succeeds but the event's action fails. See [Section 19.4.6, “The Event Scheduler and MySQL Privileges”](#) for details.
- Many of the properties of an event can be set or modified using SQL statements. These properties include the event's name, timing, persistence (that is, whether it is preserved following the expiration of its schedule), status (enabled or disabled), action to be performed, and the schema to which it is assigned. See [Section 13.1.2, “ALTER EVENT Syntax”](#).

The default definer of an event is the user who created the event, unless the event has been altered, in which case the definer is the user who issued the last `ALTER EVENT` statement affecting that event. An event can be modified by any user having the `EVENT` [696] privilege on the database for which the event is defined. See [Section 19.4.6, “The Event Scheduler and MySQL Privileges”](#).

- An event's action statement may include most SQL statements permitted within stored routines. For restrictions, see [Section E.1, “Restrictions on Stored Programs”](#).

19.4.2. Event Scheduler Configuration

Events are executed by a special *event scheduler thread*; when we refer to the Event Scheduler, we actually refer to this thread. When running, the event scheduler thread and its current state can be seen by users having the `PROCESS` [696] privilege in the output of `SHOW PROCESSLIST`, as shown in the discussion that follows.

The global `event_scheduler` [498] system variable determines whether the Event Scheduler is enabled and running on the server. It has one of these 3 values, which affect event scheduling as described here:

- **OFF:** The Event Scheduler is stopped. The event scheduler thread does not run, is not shown in the output of `SHOW PROCESSLIST`, and no scheduled events are executed. **OFF** is the default value for `event_scheduler` [498].

When the Event Scheduler is stopped (`event_scheduler` [498] is **OFF**), it can be started by setting the value of `event_scheduler` [498] to **ON**. (See next item.)

- **ON:** The Event Scheduler is started; the event scheduler thread runs and executes all scheduled events.

When the Event Scheduler is **ON**, the event scheduler thread is listed in the output of `SHOW PROCESSLIST` as a daemon process, and its state is represented as shown here:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 1
  User: root
  Host: localhost
  db: NULL
Command: Query
  Time: 0
  State: NULL
  Info: show processlist
***** 2. row *****
  Id: 2
  User: event_scheduler
  Host: localhost
  db: NULL
```

```
Command: Daemon
Time: 3
State: Waiting for next activation
Info: NULL
2 rows in set (0.00 sec)
```

Event scheduling can be stopped by setting the value of `event_scheduler` [498] to `OFF`.

- **DISABLED**: This value renders the Event Scheduler nonoperational. When the Event Scheduler is **DISABLED**, the event scheduler thread does not run (and so does not appear in the output of `SHOW PROCESSLIST`). In addition, the Event Scheduler state cannot be changed at runtime.

If the Event Scheduler status has not been set to **DISABLED**, `event_scheduler` [498] can be toggled between **ON** and **OFF** (using `SET`). It is also possible to use `0` for **OFF**, and `1` for **ON** when setting this variable. Thus, any of the following 4 statements can be used in the `mysql` client to turn on the Event Scheduler:

```
SET GLOBAL event_scheduler = ON;
SET @@global.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@global.event_scheduler = 1;
```

Similarly, any of these 4 statements can be used to turn off the Event Scheduler:

```
SET GLOBAL event_scheduler = OFF;
SET @@global.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@global.event_scheduler = 0;
```

Although **ON** and **OFF** have numeric equivalents, the value displayed for `event_scheduler` [498] by `SELECT` or `SHOW VARIABLES` is always one of **OFF**, **ON**, or **DISABLED**. **DISABLED** has no numeric equivalent. For this reason, **ON** and **OFF** are usually preferred over `1` and `0` when setting this variable.

Note that attempting to set `event_scheduler` [498] without specifying it as a global variable causes an error:

```
mysql> SET @@event_scheduler = OFF;
ERROR 1229 (HY000): Variable 'event_scheduler' is a GLOBAL
variable and should be set with SET GLOBAL
```



Important

It is possible to set the Event Scheduler to **DISABLED** only at server startup. If `event_scheduler` [498] is **ON** or **OFF**, you cannot set it to **DISABLED** at runtime. Also, if the Event Scheduler is set to **DISABLED** at startup, you cannot change the value of `event_scheduler` [498] at runtime.

To disable the event scheduler, use one of the following two methods:

- As a command-line option when starting the server:

```
--event-scheduler=DISABLED
```

- In the server configuration file (`my.cnf`, or `my.ini` on Windows systems), include the line where it will be read by the server (for example, in a `[mysqld]` section):

```
event_scheduler=DISABLED
```

To enable the Event Scheduler, restart the server without the `--event-scheduler=DISABLED` [443] command-line option, or after removing or commenting out the line containing `event-`

`scheduler=DISABLED` [443] in the server configuration file, as appropriate. Alternatively, you can use `ON` (or `1`) or `OFF` (or `0`) in place of the `DISABLED` value when starting the server.

**Note**

You can issue event-manipulation statements when `event_scheduler` [498] is set to `DISABLED`. No warnings or errors are generated in such cases (provided that the statements are themselves valid). However, scheduled events cannot execute until this variable is set to `ON` (or `1`). Once this has been done, the event scheduler thread executes all events whose scheduling conditions are satisfied.

Starting the MySQL server with the `--skip-grant-tables` [461] option causes `event_scheduler` [498] to be set to `DISABLED`, overriding any other value set either on the command line or in the `my.cnf` or `my.ini` file (Bug #26807).

For SQL statements used to create, alter, and drop events, see [Section 19.4.3, “Event Syntax”](#).

MySQL 5.5 provides an `EVENTS` table in the `INFORMATION_SCHEMA` database. This table can be queried to obtain information about scheduled events which have been defined on the server. See [Section 19.4.4, “Event Metadata”](#), and [Section 20.7, “The INFORMATION_SCHEMA EVENTS Table”](#), for more information.

For information regarding event scheduling and the MySQL privilege system, see [Section 19.4.6, “The Event Scheduler and MySQL Privileges”](#).

19.4.3. Event Syntax

MySQL 5.5 provides several SQL statements for working with scheduled events:

- New events are defined using the `CREATE EVENT` statement. See [Section 13.1.11, “CREATE EVENT Syntax”](#).
- The definition of an existing event can be changed by means of the `ALTER EVENT` statement. See [Section 13.1.2, “ALTER EVENT Syntax”](#).
- When a scheduled event is no longer wanted or needed, it can be deleted from the server by its definer using the `DROP EVENT` statement. See [Section 13.1.22, “DROP EVENT Syntax”](#). Whether an event persists past the end of its schedule also depends on its `ON COMPLETION` clause, if it has one. See [Section 13.1.11, “CREATE EVENT Syntax”](#).

An event can be dropped by any user having the `EVENT` [696] privilege for the database on which the event is defined. See [Section 19.4.6, “The Event Scheduler and MySQL Privileges”](#).

19.4.4. Event Metadata

Metadata about events can be obtained as follows:

- Query the `event` table of the `mysql` database.
- Query the `EVENTS` table of the `INFORMATION_SCHEMA` database. See [Section 20.7, “The INFORMATION_SCHEMA EVENTS Table”](#).
- Use the `SHOW CREATE EVENT` statement. See [Section 13.7.5.9, “SHOW CREATE EVENT Syntax”](#).
- Use the `SHOW EVENTS` statement. See [Section 13.7.5.19, “SHOW EVENTS Syntax”](#).

Event Scheduler Time Representation

Each session in MySQL has a session time zone (STZ). This is the session `time_zone` [567] value that is initialized from the server's global `time_zone` [567] value when the session begins but may be changed during the session.

The session time zone that is current when a `CREATE EVENT` or `ALTER EVENT` statement executes is used to interpret times specified in the event definition. This becomes the event time zone (ETZ); that is, the time zone that is used for event scheduling and is in effect within the event as it executes.

For representation of event information in the `mysql.event` table, the `execute_at`, `starts`, and `ends` times are converted to UTC and stored along with the event time zone. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. The `last_executed` time is also stored in UTC.

If you select information from `mysql.event`, the times just mentioned are retrieved as UTC values. These times can also be obtained by selecting from the `INFORMATION_SCHEMA.EVENTS` table or from `SHOW EVENTS`, but they are reported as ETZ values. Other times available from these sources indicate when an event was created or last altered; these are displayed as STZ values. The following table summarizes representation of event times.

Value	<code>mysql.event</code>	<code>INFORMATION_SCHEMA.EVENTS</code>	<code>SHOW EVENTS</code>
Execute at	UTC	ETZ	ETZ
Starts	UTC	ETZ	ETZ
Ends	UTC	ETZ	ETZ
Last executed	UTC	ETZ	n/a
Created	STZ	STZ	n/a
Last altered	STZ	STZ	n/a

19.4.5. Event Scheduler Status

The Event Scheduler writes information about event execution that terminates with an error or warning to the MySQL Server's error log. See [Section 19.4.6, "The Event Scheduler and MySQL Privileges"](#) for an example.

Information about the state of the Event Scheduler for debugging and troubleshooting purposes can be obtained by running `mysqladmin debug` (see [Section 4.5.2, "mysqladmin — Client for Administering a MySQL Server"](#)); after running this command, the server's error log contains output relating to the Event Scheduler, similar to what is shown here:

```

Events status:
LLA = Last Locked At  LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State      : INITIALIZED
Thread id  : 0
LLA       : init_scheduler:313
LUA       : init_scheduler:318
WOC       : NO
Workers   : 0
Executed   : 0
Data locked: NO

Event queue status:
Element count : 1
    
```

```
Data locked      : NO
Attempting lock : NO
LLA             : init_queue:148
LUA            : init_queue:168
WOC            : NO
Next activation : 0000-00-00 00:00:00
```

In statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For frequently executed events, it is possible for this to result in many logged messages. For example, for `SELECT ... INTO var_list` statements, if the query returns no rows, a warning with error code 1329 occurs (`No data`), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (`Result consisted of more than one row`). For either condition, you can avoid having the warnings be logged by declaring a condition handler; see [Section 13.6.7.2, “DECLARE ... HANDLER Syntax”](#). For statements that may retrieve multiple rows, another strategy is to use `LIMIT 1` to limit the result set to a single row.

19.4.6. The Event Scheduler and MySQL Privileges

To enable or disable the execution of scheduled events, it is necessary to set the value of the global `event_scheduler` [\[498\]](#) system variable. This requires the `SUPER` [\[697\]](#) privilege.

The `EVENT` [\[696\]](#) privilege governs the creation, modification, and deletion of events. This privilege can be bestowed using `GRANT`. For example, this `GRANT` statement confers the `EVENT` [\[696\]](#) privilege for the schema named `myschema` on the user `jon@ghidora`:

```
GRANT EVENT ON myschema.* TO jon@ghidora;
```

(We assume that this user account already exists, and that we wish for it to remain unchanged otherwise.)

To grant this same user the `EVENT` [\[696\]](#) privilege on all schemas, use the following statement:

```
GRANT EVENT ON *.* TO jon@ghidora;
```

The `EVENT` [\[696\]](#) privilege has global or schema-level scope. Therefore, trying to grant it on a single table results in an error as shown:

```
mysql> GRANT EVENT ON myschema.mytable TO jon@ghidora;
ERROR 1144 (42000): Illegal GRANT/REVOKE command; please
consult the manual to see which privileges can be used
```

It is important to understand that an event is executed with the privileges of its definer, and that it cannot perform any actions for which its definer does not have the requisite privileges. For example, suppose that `jon@ghidora` has the `EVENT` [\[696\]](#) privilege for `myschema`. Suppose also that this user has the `SELECT` [\[697\]](#) privilege for `myschema`, but no other privileges for this schema. It is possible for `jon@ghidora` to create a new event such as this one:

```
CREATE EVENT e_store_ts
ON SCHEDULE
  EVERY 10 SECOND
DO
  INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
```

The user waits for a minute or so, and then performs a `SELECT * FROM mytable;` query, expecting to see several new rows in the table. Instead, the table is empty. Since the user does not have the `INSERT` [\[696\]](#) privilege for the table in question, the event has no effect.

If you inspect the MySQL error log (`hostname.err`), you can see that the event is executing, but the action it is attempting to perform fails, as indicated by `RetCode=0`:

```
060209 22:39:44 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:39:44 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0
060209 22:39:54 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:39:54 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0
060209 22:40:04 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:40:04 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0
```

Since this user very likely does not have access to the error log, it is possible to verify whether the event's action statement is valid by executing it directly:

```
mysql> INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
ERROR 1142 (42000): INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
```

Inspection of the `INFORMATION_SCHEMA.EVENTS` table shows that `e_store_ts` exists and is enabled, but its `LAST_EXECUTED` column is `NULL`:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME='e_store_ts'
> AND EVENT_SCHEMA='myschema'\G
***** 1. row *****
EVENT_CATALOG: NULL
EVENT_SCHEMA: myschema
EVENT_NAME: e_store_ts
DEFINER: jon@ghidora
EVENT_BODY: SQL
EVENT_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP())
EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 5
INTERVAL_FIELD: SECOND
SQL_MODE: NULL
STARTS: 0000-00-00 00:00:00
ENDS: 0000-00-00 00:00:00
STATUS: ENABLED
ON_COMPLETION: NOT PRESERVE
CREATED: 2006-02-09 22:36:06
LAST_ALTERED: 2006-02-09 22:36:06
LAST_EXECUTED: NULL
EVENT_COMMENT:
1 row in set (0.00 sec)
```

To rescind the `EVENT [696]` privilege, use the `REVOKE` statement. In this example, the `EVENT [696]` privilege on the schema `myschema` is removed from the `jon@ghidora` user account:

```
REVOKE EVENT ON myschema.* FROM jon@ghidora;
```



Important

Revoking the `EVENT [696]` privilege from a user does not delete or disable any events that may have been created by that user.

An event is not migrated or dropped as a result of renaming or dropping the user who created it.

Suppose that the user `jon@ghidora` has been granted the `EVENT [696]` and `INSERT [696]` privileges on the `myschema` schema. This user then creates the following event:

```
CREATE EVENT e_insert
ON SCHEDULE
EVERY 7 SECOND
DO
```

```
INSERT INTO myschema.mytable;
```

After this event has been created, `root` revokes the `EVENT [696]` privilege for `jon@ghidora`. However, `e_insert` continues to execute, inserting a new row into `mytable` each seven seconds. The same would be true if `root` had issued either of these statements:

- `DROP USER jon@ghidora;`
- `RENAME USER jon@ghidora TO someotherguy@ghidora;`

You can verify that this is true by examining the `mysql.event` table (discussed later in this section) or the `INFORMATION_SCHEMA.EVENTS` table (see [Section 20.7, “The INFORMATION_SCHEMA EVENTS Table”](#)) before and after issuing a `DROP USER` or `RENAME USER` statement.

Event definitions are stored in the `mysql.event` table. To drop an event created by another user account, the MySQL `root` user (or another user with the necessary privileges) can delete rows from this table. For example, to remove the event `e_insert` shown previously, `root` can use the following statement:

```
DELETE FROM mysql.event
WHERE db = 'myschema'
  AND definer = 'jon@ghidora'
  AND name = 'e_insert';
```

It is very important to match the event name, database schema name, and user account when deleting rows from the `mysql.event` table. This is because the same user can create different events of the same name in different schemas.

Users' `EVENT [696]` privileges are stored in the `Event_priv` columns of the `mysql.user` and `mysql.db` tables. In both cases, this column holds one of the values 'Y' or 'N'. 'N' is the default. `mysql.user.Event_priv` is set to 'Y' for a given user only if that user has the global `EVENT [696]` privilege (that is, if the privilege was bestowed using `GRANT EVENT ON *.*`). For a schema-level `EVENT [696]` privilege, `GRANT` creates a row in `mysql.db` and sets that row's `Db` column to the name of the schema, the `User` column to the name of the user, and the `Event_priv` column to 'Y'. There should never be any need to manipulate these tables directly, since the `GRANT EVENT` and `REVOKE EVENT` statements perform the required operations on them.

Five status variables provide counts of event-related operations (but *not* of statements executed by events; see [Section E.1, “Restrictions on Stored Programs”](#)). These are:

- `Com_create_event`: The number of `CREATE EVENT` statements executed since the last server restart.
- `Com_alter_event`: The number of `ALTER EVENT` statements executed since the last server restart.
- `Com_drop_event`: The number of `DROP EVENT` statements executed since the last server restart.
- `Com_show_create_event`: The number of `SHOW CREATE EVENT` statements executed since the last server restart.
- `Com_show_events`: The number of `SHOW EVENTS` statements executed since the last server restart.

You can view current values for all of these at one time by running the statement `SHOW STATUS LIKE '%event%';`.

19.5. Using Views

Views (including updatable views) are available in MySQL Server 5.5. Views are stored queries that when invoked produce a result set. A view acts as a virtual table.

To use views if you have upgraded to MySQL 5.5 from an older release that did not support views, you should upgrade your grant tables so that they contain the view-related privileges. See [Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”](#).

The following discussion describes the syntax for creating and dropping views, and shows some examples of how to use them.

Additional Resources

- You may find the [Views User Forum](#) of use when working with views.
- For answers to some commonly asked questions regarding views in MySQL, see [Section B.6, “MySQL 5.5 FAQ: Views”](#).
- There are some restrictions on the use of views; see [Section E.5, “Restrictions on Views”](#).

19.5.1. View Syntax

The `CREATE VIEW` statement creates a new view (see [Section 13.1.20, “CREATE VIEW Syntax”](#)). To alter the definition of a view or drop a view, use `ALTER VIEW` (see [Section 13.1.9, “ALTER VIEW Syntax”](#)), or `DROP VIEW` (see [Section 13.1.31, “DROP VIEW Syntax”](#)).

A view can be created from many kinds of `SELECT` statements. It can refer to base tables or other views. It can use joins, `UNION`, and subqueries. The `SELECT` need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50), (5, 60);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+-----+-----+-----+
| qty  | price | value |
+-----+-----+-----+
| 3    | 50    | 150   |
| 5    | 60    | 300   |
+-----+-----+-----+
mysql> SELECT * FROM v WHERE qty = 5;
+-----+-----+-----+
| qty  | price | value |
+-----+-----+-----+
| 5    | 60    | 300   |
+-----+-----+-----+
```

19.5.2. View Processing Algorithms

The optional `ALGORITHM` clause for `CREATE VIEW` or `ALTER VIEW` is a MySQL extension to standard SQL. It affects how MySQL processes the view. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`. The default algorithm is `UNDEFINED` if no `ALGORITHM` clause is present.

For `MERGE`, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.

For `TEMPTABLE`, the results from the view are retrieved into a temporary table, which then is used to execute the statement.

For `UNDEFINED`, MySQL chooses which algorithm to use. It prefers `MERGE` over `TEMPTABLE` if possible, because `MERGE` is usually more efficient and because a view cannot be updatable if a temporary table is used.

A reason to choose `TEMPTABLE` explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the `MERGE` algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be `UNDEFINED` for three reasons:

- No `ALGORITHM` clause is present in the `CREATE VIEW` statement.
- The `CREATE VIEW` statement has an explicit `ALGORITHM = UNDEFINED` clause.
- `ALGORITHM = MERGE` is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to `UNDEFINED`.

As mentioned earlier, `MERGE` is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the `MERGE` algorithm works. The examples assume that there is a view `v_merge` that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- `v_merge` becomes `t`
- `*` becomes `vc1, vc2`, which corresponds to `c1, c2`
- The view `WHERE` clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 2: Suppose that we issue this statement:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that `vc1 < 100` becomes `c1 < 100` and the view `WHERE` clause is added to the statement `WHERE` clause using an `AND` [1094] connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Effectively, the statement to be executed has a `WHERE` clause of this form:

```
WHERE (select WHERE) AND (view WHERE)
```

If the `MERGE` algorithm cannot be used, a temporary table must be used instead. `MERGE` cannot be used if the view contains any of the following constructs:

- Aggregate functions (`SUM()` [1206], `MIN()` [1206], `MAX()` [1205], `COUNT()` [1204], and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`

- `LIMIT`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Refers only to literal values (in this case, there is no underlying table)

19.5.3. Updatable and Insertable Views

Some views are updatable. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions (`SUM()` [1206], `MIN()` [1206], `MAX()` [1205], `COUNT()` [1204], and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Certain joins (see additional join discussion later in this section)
- Nonupdatable view in the `FROM` clause
- A subquery in the `WHERE` clause that refers to a table in the `FROM` clause
- Refers only to literal values (in this case, there is no underlying table to update)
- Uses `ALGORITHM = TEMPTABLE` (use of a temporary table always makes a view nonupdatable)
- Multiple references to any column of a base table.

With respect to insertability (being updatable with `INSERT` statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references and not derived columns. A derived column is one that is not a simple column reference but is derived from an expression. These are examples of derived columns:

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

A view that has a mix of simple column references and derived columns is not insertable, but it can be updatable if you update only those columns that are not derived. Consider this view:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because `col2` is derived from an expression. But it is updatable if the update does not try to update `col2`. This update is permissible:

```
UPDATE v SET col1 = 0;
```

This update is not permissible because it attempts to update a derived column:

```
UPDATE v SET col2 = 0;
```

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the `MERGE` algorithm. For this to work, the view must use an inner join (not an outer join or a `UNION`). Also, only a single table in the view definition can be updated, so the `SET` clause must name only columns from one of the tables in the view. Views that use `UNION ALL` are not permitted even though they might be theoretically updatable, because the implementation uses temporary tables to process them.

For a multiple-table updatable view, `INSERT` can work if it inserts into a single table. `DELETE` is not supported.

`INSERT DELAYED` is not supported for views.

If a table contains an `AUTO_INCREMENT` column, inserting into an insertable view on the table that does not include the `AUTO_INCREMENT` column does not change the value of `LAST_INSERT_ID()` [1194], because the side effects of inserting default values into columns not part of the view should not be visible.

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts or updates to rows except those for which the `WHERE` clause in the `select_statement` is true.

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. The `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is `CASCADED`. Consider the definitions for the following table and set of views:

```
mysql> CREATE TABLE t1 (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
-> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
-> WITH LOCAL CHECK OPTION;
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
-> WITH CASCADED CHECK OPTION;
```

Here the `v2` and `v3` views are defined in terms of another view, `v1`. `v2` has a `LOCAL` check option, so inserts are tested only against the `v2` check. `v3` has a `CASCADED` check option, so inserts are tested not only against its own check, but against those of underlying views. The following statements illustrate these differences:

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `INFORMATION_SCHEMA.VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable. If the view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it, as described elsewhere in this section.)

The updatability of views may be affected by the value of the `updatable_views_with_limit` [571] system variable. See [Section 5.1.4, “Server System Variables”](#).

19.5.4. View Metadata

Metadata about views can be obtained as follows:

- Query the `VIEWS` table of the `INFORMATION_SCHEMA` database. See [Section 20.27, “The INFORMATION_SCHEMA VIEWS Table”](#).
- Use the `SHOW CREATE VIEW` statement. See [Section 13.7.5.14, “SHOW CREATE VIEW Syntax”](#).

19.6. Access Control for Stored Programs and Views

Stored programs and views are defined prior to use and, when referenced, execute within a security context that determines their privileges. These privileges are controlled by their `DEFINER` attribute, and, if there is one, their `SQL SECURITY` characteristic.

All stored programs (procedures, functions, triggers, and events) and views can have a `DEFINER` attribute that names a MySQL account. If the `DEFINER` attribute is omitted from a stored program or view definition, the default account is the user who creates the object.

In addition, stored routines (procedures and functions) and views can have a `SQL SECURITY` characteristic with a value of `DEFINER` or `INVOKER` to specify whether the object executes in definer or invoker context. If the `SQL SECURITY` characteristic is omitted, the default is definer context.

Triggers and events have no `SQL SECURITY` characteristic and always execute in definer context. The server invokes these objects automatically as necessary, so there is no invoking user.

Definer and invoker security contexts differ as follows:

- A stored program or view that executes in definer security context executes with the privileges of the account named by its `DEFINER` attribute. These privileges may be entirely different from those of the invoking user. The invoker must have appropriate privileges to reference the object (for example, `EXECUTE` [696] to call a stored procedure or `SELECT` [697] to select from a view), but when the object executes, the invoker's privileges are ignored and only the `DEFINER` account privileges matter. If this account has few privileges, the object is correspondingly limited in the operations it can perform. If the `DEFINER` account is highly privileged (such as a `root` account), the object can perform powerful operations *no matter who invokes it*.
- A stored routine or view that executes in invoker security context can perform only operations for which the invoker has privileges. The `DEFINER` attribute can be specified but has no effect for objects that execute in invoker context.

Consider the following stored procedure:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()
SQL SECURITY DEFINER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

Any user who has the `EXECUTE` [696] privilege for `p1` can invoke it with a `CALL` statement. However, when `p1` executes, it does so in `DEFINER` security context and thus executes with the privileges of `'admin'@'localhost'`, the account named in the `DEFINER` attribute. This account must have the `EXECUTE` [696] privilege for `p1` as well as the `UPDATE` [697] privilege for the table `t1`. Otherwise, the procedure fails.

Now consider this stored procedure, which is identical to `p1` except that its `SQL SECURITY` characteristic is `INVOKER`:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p2()
SQL SECURITY INVOKER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

`p2`, unlike `p1`, executes in `INVOKER` security context. The `DEFINER` attribute is irrelevant and `p2` executes with the privileges of the invoking user. `p2` fails if the invoker lacks the `EXECUTE` [696] privilege for `p2` or the `UPDATE` [697] privilege for the table `t1`.

MySQL uses the following rules to control which accounts a user can specify in an object `DEFINER` attribute:

- You can specify a `DEFINER` value other than your own account only if you have the `SUPER` [697] privilege.
- If you do not have the `SUPER` [697] privilege, the only legal user value is your own account, either specified literally or by using `CURRENT_USER` [1192]. You cannot set the definer to some other account.

To minimize the risk potential for stored program and view creation and use, follow these guidelines:

- For a stored routine or view, use `SQL SECURITY INVOKER` in the object definition when possible so that it can be used only by users with permissions appropriate for the operations performed by the object.
- If you create definer-context stored programs or views while using an account that has the `SUPER` [697] privilege, specify an explicit `DEFINER` attribute that names an account possessing only the privileges required for the operations performed by the object. Specify a highly privileged `DEFINER` account only when absolutely necessary.
- Administrators can prevent users from specifying highly privileged `DEFINER` accounts by not granting them the `SUPER` [697] privilege.
- Definer-context objects should be written keeping in mind that they may be able to access data for which the invoking user has no privileges. In some cases, you can prevent reference to these objects by not granting unauthorized users particular privileges:
 - A stored procedure or function cannot be referenced by a user who does not have the `EXECUTE` [696] privilege for it.
 - A view cannot be referenced by a user who does not have the appropriate privilege for it (`SELECT` [697] to select from it, `INSERT` [696] to insert into it, and so forth).

However, no such control exists for triggers because users do not reference them directly. A trigger always executes in `DEFINER` context and is activated by access to the table with which it is associated, even ordinary table accesses by users with no special privileges. If the `DEFINER` account is highly privileged, the trigger can perform sensitive or dangerous operations. This remains true if the `SUPER` [697] and `TRIGGER` [697] privileges needed to create the trigger are revoked from the account of the user who created it. Administrators should be especially careful about granting users that combination of privileges.

19.7. Binary Logging of Stored Programs

The binary log contains information about SQL statements that modify database contents. This information is stored in the form of “events” that describe the modifications. The binary log has two important purposes:

- For replication, the binary log is used on master replication servers as a record of the statements to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See [Section 16.2, “Replication Implementation”](#).
- Certain data recovery operations require use of the binary log. After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See [Section 7.3.2, “Using Backups for Recovery”](#).

However, there are certain binary logging issues that apply with respect to stored programs (stored procedures and functions, triggers, and events), if logging occurs at the statement level:

- In some cases, it is possible that a statement will affect different sets of rows on a master and a slave.
- Replicated statements executed on a slave are processed by the slave SQL thread, which has full privileges. It is possible for a procedure to follow different execution paths on master and slave servers, so a user can write a routine containing a dangerous statement that will execute only on the slave where it is processed by a thread that has full privileges.
- If a stored program that modifies data is nondeterministic, it is not repeatable. This can result in different data on a master and slave, or cause restored data to differ from the original data.

This section describes how MySQL 5.5 handles binary logging for stored programs. It states the current conditions that the implementation places on the use of stored programs, and what you can do to avoid problems. It also provides additional information about the reasons for these conditions.

In general, the issues described here result when binary logging occurs at the SQL statement level. If you use row-based binary logging, the log contains changes made to individual rows as a result of executing SQL statements. When routines or triggers execute, row changes are logged, not the statements that make the changes. For stored procedures, this means that the `CALL` statement is not logged. For stored functions, row changes made within the function are logged, not the function invocation. For triggers, row changes made by the trigger are logged. On the slave side, only the row changes are seen, not the stored program invocation. For general information about row-based logging, see [Section 16.1.2, “Replication Formats”](#).

Unless noted otherwise, the remarks here assume that you have enabled binary logging by starting the server with the `--log-bin [1892]` option. (See [Section 5.2.4, “The Binary Log”](#).) If the binary log is not enabled, replication is not possible, nor is the binary log available for data recovery.

The current conditions on the use of stored functions in MySQL 5.5 can be summarized as follows. These conditions do not apply to stored procedures or Event Scheduler events and they do not apply unless binary logging is enabled.

- To create or alter a stored function, you must have the `SUPER [697]` privilege, in addition to the `CREATE ROUTINE [695]` or `ALTER ROUTINE [695]` privilege that is normally required. (Depending on the `DEFINER` value in the function definition, `SUPER [697]` might be required regardless of whether binary logging is enabled. See [Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).)
- When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for data recovery or replication.

By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
```

(you *might* want to use the less safe `log_bin_trust_function_creators` variable)

This function is deterministic (and does not modify data), so it is safe:

```
CREATE FUNCTION f1(i INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    RETURN i;
END;
```

This function uses `UUID()` [1201], which is not deterministic, so the function also is not deterministic and is not safe:

```
CREATE FUNCTION f2()
RETURNS CHAR(36) CHARACTER SET utf8
BEGIN
    RETURN UUID();
END;
```

This function modifies data, so it may not be safe:

```
CREATE FUNCTION f3(p_id INT)
RETURNS INT
BEGIN
    UPDATE t SET modtime = NOW() WHERE id = p_id;
    RETURN ROW_COUNT();
END;
```

Assessment of the nature of a function is based on the “honesty” of the creator: MySQL does not check that a function declared `DETERMINISTIC` is free of statements that produce nondeterministic results.

- Although it is possible to create a deterministic stored function without specifying `DETERMINISTIC`, you cannot execute this function using statement-based binary logging. To execute such a function, you must use row-based or mixed binary logging. Alternatively, if you explicitly specify `DETERMINISTIC` in the function definition, you can use any kind of logging, including statement-based logging.
- To relax the preceding conditions on function creation (that you must have the `SUPER` [697] privilege and that a function must be declared deterministic or to not modify data), set the global `log_bin_trust_function_creators` [514] system variable to 1. By default, this variable has a value of 0, but you can change it like this:

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

You can also set this variable by using the `--log-bin-trust-function-creators=1` [1892] option when starting the server.

If binary logging is not enabled, `log_bin_trust_function_creators` [514] does not apply. `SUPER` [697] is not required for function creation unless, as described previously, the `DEFINER` value in the function definition requires it.

- For information about built-in functions that may be unsafe for replication (and thus cause stored functions that use them to be unsafe as well), see [Section 16.4.1, “Replication Features and Issues”](#).

Triggers are similar to stored functions, so the preceding remarks regarding functions also apply to triggers with the following exception: `CREATE TRIGGER` does not have an optional `DETERMINISTIC` characteristic, so triggers are assumed to be always deterministic. However, this assumption might in some cases be invalid. For example, the `UUID()` [1201] function is nondeterministic (and does not replicate). You should be careful about using such functions in triggers.

Triggers can update tables, so error messages similar to those for stored functions occur with `CREATE TRIGGER` if you do not have the required privileges. On the slave side, the slave uses the trigger `DEFINER` attribute to determine which user is considered to be the creator of the trigger.

The rest of this section provides additional detail about the logging implementation and its implications. You need not read it unless you are interested in the background on the rationale for the current logging-related conditions on stored routine use. This discussion applies only for statement-based logging, and not for row-based logging, with the exception of the first item: `CREATE` and `DROP` statements are logged as statements regardless of the logging mode.

- The server writes `CREATE EVENT`, `CREATE PROCEDURE`, `CREATE FUNCTION`, `ALTER EVENT`, `ALTER PROCEDURE`, `ALTER FUNCTION`, `DROP EVENT`, `DROP PROCEDURE`, and `DROP FUNCTION` statements to the binary log.
- A stored function invocation is logged as a `SELECT` statement if the function changes data and occurs within a statement that would not otherwise be logged. This prevents nonreplication of data changes that result from use of stored functions in nonlogged statements. For example, `SELECT` statements are not written to the binary log, but a `SELECT` might invoke a stored function that makes changes. To handle this, a `SELECT func_name()` statement is written to the binary log when the given function makes a change. Suppose that the following statements are executed on the master:

```
CREATE FUNCTION f1(a INT) RETURNS INT
BEGIN
  IF (a < 3) THEN
    INSERT INTO t2 VALUES (a);
  END IF;
  RETURN 0;
END;

CREATE TABLE t1 (a INT);
INSERT INTO t1 VALUES (1),(2),(3);

SELECT f1(a) FROM t1;
```

When the `SELECT` statement executes, the function `f1()` is invoked three times. Two of those invocations insert a row, and MySQL logs a `SELECT` statement for each of them. That is, MySQL writes the following statements to the binary log:

```
SELECT f1(1);
SELECT f1(2);
```

The server also logs a `SELECT` statement for a stored function invocation when the function invokes a stored procedure that causes an error. In this case, the server writes the `SELECT` statement to the log along with the expected error code. On the slave, if the same error occurs, that is the expected result and replication continues. Otherwise, replication stops.

- Logging stored function invocations rather than the statements executed by a function has a security implication for replication, which arises from two factors:
 - It is possible for a function to follow different execution paths on master and slave servers.
 - Statements executed on a slave are processed by the slave SQL thread which has full privileges.

The implication is that although a user must have the `CREATE ROUTINE` [695] privilege to create a function, the user can write a function containing a dangerous statement that will execute only on the slave where it is processed by a thread that has full privileges. For example, if the master and slave servers have server ID values of 1 and 2, respectively, a user on the master server could create and invoke an unsafe function `unsafe_func()` as follows:

```
mysql> delimiter //
mysql> CREATE FUNCTION unsafe_func () RETURNS INT
-> BEGIN
->   IF @@server_id=2 THEN dangerous_statement; END IF;
->   RETURN 1;
-> END;
-> //
mysql> delimiter ;
mysql> INSERT INTO t VALUES(unsafe_func());
```

The `CREATE FUNCTION` and `INSERT` statements are written to the binary log, so the slave will execute them. Because the slave SQL thread has full privileges, it will execute the dangerous statement. Thus, the function invocation has different effects on the master and slave and is not replication-safe.

To guard against this danger for servers that have binary logging enabled, stored function creators must have the `SUPER` [697] privilege, in addition to the usual `CREATE ROUTINE` [695] privilege that is required. Similarly, to use `ALTER FUNCTION`, you must have the `SUPER` [697] privilege in addition to the `ALTER ROUTINE` [695] privilege. Without the `SUPER` [697] privilege, an error will occur:

```
ERROR 1419 (HY000): You do not have the SUPER privilege and
binary logging is enabled (you *might* want to use the less safe
log_bin_trust_function_creators variable)
```

If you do not want to require function creators to have the `SUPER` [697] privilege (for example, if all users with the `CREATE ROUTINE` [695] privilege on your system are experienced application developers), set the global `log_bin_trust_function_creators` [514] system variable to 1. You can also set this variable by using the `--log-bin-trust-function-creators=1` [1892] option when starting the server. If binary logging is not enabled, `log_bin_trust_function_creators` [514] does not apply. `SUPER` [697] is not required for function creation unless, as described previously, the `DEFINER` value in the function definition requires it.

- If a function that performs updates is nondeterministic, it is not repeatable. This can have two undesirable effects:
 - It will make a slave different from the master.
 - Restored data will be different from the original data.

To deal with these problems, MySQL enforces the following requirement: On a master server, creation and alteration of a function is refused unless you declare the function to be deterministic or to not modify data. Two sets of function characteristics apply here:

- The `DETERMINISTIC` and `NOT DETERMINISTIC` characteristics indicate whether a function always produces the same result for given inputs. The default is `NOT DETERMINISTIC` if neither characteristic is given. To declare that a function is deterministic, you must specify `DETERMINISTIC` explicitly.
- The `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` characteristics provide information about whether the function reads or writes data. Either `NO SQL` or `READS SQL DATA` indicates that a function does not change data, but you must specify one of these explicitly because the default is `CONTAINS SQL` if no characteristic is given.

By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
```

```
variable)
```

If you set `log_bin_trust_function_creators` [514] to 1, the requirement that functions be deterministic or not modify data is dropped.

- Stored procedure calls are logged at the statement level rather than at the `CALL` level. That is, the server does not log the `CALL` statement, it logs those statements within the procedure that actually execute. As a result, the same changes that occur on the master will be observed on slave servers. This prevents problems that could result from a procedure having different execution paths on different machines.

In general, statements executed within a stored procedure are written to the binary log using the same rules that would apply were the statements to be executed in standalone fashion. Some special care is taken when logging procedure statements because statement execution within procedures is not quite the same as in nonprocedure context:

- A statement to be logged might contain references to local procedure variables. These variables do not exist outside of stored procedure context, so a statement that refers to such a variable cannot be logged literally. Instead, each reference to a local variable is replaced by this construct for logging purposes:

```
NAME_CONST(var_name, var_value)
```

`var_name` is the local variable name, and `var_value` is a constant indicating the value that the variable has at the time the statement is logged. `NAME_CONST()` [1201] has a value of `var_value`, and a “name” of `var_name`. Thus, if you invoke this function directly, you get a result like this:

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|      14 |
+-----+
```

`NAME_CONST()` [1201] enables a logged standalone statement to be executed on a slave with the same effect as the original statement that was executed on the master within a stored procedure.

The use of `NAME_CONST()` [1201] can result in a problem for `CREATE TABLE ... SELECT` statements when the source column expressions refer to local variables. Converting these references to `NAME_CONST()` [1201] expressions can result in column names that are different on the master and slave servers, or names that are too long to be legal column identifiers. A workaround is to supply aliases for columns that refer to local variables. Consider this statement when `myvar` has a value of 1:

```
CREATE TABLE t1 SELECT myvar;
```

That will be rewritten as follows:

```
CREATE TABLE t1 SELECT NAME_CONST(myvar, 1);
```

To ensure that the master and slave tables have the same column names, write the statement like this:

```
CREATE TABLE t1 SELECT myvar AS myvar;
```

The rewritten statement becomes:

```
CREATE TABLE t1 SELECT NAME_CONST(myvar, 1) AS myvar;
```

- A statement to be logged might contain references to user-defined variables. To handle this, MySQL writes a `SET` statement to the binary log to make sure that the variable exists on the slave with

the same value as on the master. For example, if a statement refers to a variable `@my_var`, that statement will be preceded in the binary log by the following statement, where `value` is the value of `@my_var` on the master:

```
SET @my_var = value;
```

- Procedure calls can occur within a committed or rolled-back transaction. Transactional context is accounted for so that the transactional aspects of procedure execution are replicated correctly. That is, the server logs those statements within the procedure that actually execute and modify data, and also logs `BEGIN`, `COMMIT`, and `ROLLBACK` statements as necessary. For example, if a procedure updates only transactional tables and is executed within a transaction that is rolled back, those updates are not logged. If the procedure occurs within a committed transaction, `BEGIN` and `COMMIT` statements are logged with the updates. For a procedure that executes within a rolled-back transaction, its statements are logged using the same rules that would apply if the statements were executed in standalone fashion:
 - Updates to transactional tables are not logged.
 - Updates to nontransactional tables are logged because rollback does not cancel them.
 - Updates to a mix of transactional and nontransactional tables are logged surrounded by `BEGIN` and `ROLLBACK` so that slaves will make the same changes and rollbacks as on the master.
- A stored procedure call is *not* written to the binary log at the statement level if the procedure is invoked from within a stored function. In that case, the only thing logged is the statement that invokes the function (if it occurs within a statement that is logged) or a `DO` statement (if it occurs within a statement that is not logged). For this reason, care should be exercised in the use of stored functions that invoke a procedure, even if the procedure is otherwise safe in itself.

Chapter 20. INFORMATION_SCHEMA Tables

Table of Contents

20.1. The INFORMATION_SCHEMA CHARACTER_SETS Table	2472
20.2. The INFORMATION_SCHEMA COLLATIONS Table	2473
20.3. The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table	2473
20.4. The INFORMATION_SCHEMA COLUMNS Table	2473
20.5. The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table	2475
20.6. The INFORMATION_SCHEMA ENGINES Table	2475
20.7. The INFORMATION_SCHEMA EVENTS Table	2476
20.8. The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables	2479
20.9. The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables	2479
20.10. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table	2480
20.11. The INFORMATION_SCHEMA PARAMETERS Table	2481
20.12. The INFORMATION_SCHEMA PARTITIONS Table	2481
20.13. The INFORMATION_SCHEMA PLUGINS Table	2484
20.14. The INFORMATION_SCHEMA PROCESSLIST Table	2486
20.15. The INFORMATION_SCHEMA PROFILING Table	2486
20.16. The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table	2487
20.17. The INFORMATION_SCHEMA ROUTINES Table	2488
20.18. The INFORMATION_SCHEMA SCHEMATA Table	2489
20.19. The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table	2490
20.20. The INFORMATION_SCHEMA STATISTICS Table	2490
20.21. The INFORMATION_SCHEMA TABLES Table	2491
20.22. The INFORMATION_SCHEMA TABLESPACES Table	2492
20.23. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table	2493
20.24. The INFORMATION_SCHEMA TABLE_PRIVILEGES Table	2493
20.25. The INFORMATION_SCHEMA TRIGGERS Table	2494
20.26. The INFORMATION_SCHEMA USER_PRIVILEGES Table	2496
20.27. The INFORMATION_SCHEMA VIEWS Table	2496
20.28. INFORMATION_SCHEMA Tables for InnoDB	2497
20.28.1. The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables	2498
20.28.2. The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables	2498
20.28.3. The INFORMATION_SCHEMA INNODB_TRX Table	2499
20.28.4. The INFORMATION_SCHEMA INNODB_LOCKS Table	2500
20.28.5. The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table	2501
20.28.6. The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table	2502
20.28.7. The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table	2503
20.28.8. The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table	2503
20.29. INFORMATION_SCHEMA Tables for MySQL Cluster	2505
20.29.1. The INFORMATION_SCHEMA FILES Table	2505
20.29.2. The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table	2510
20.30. Thread Pool INFORMATION_SCHEMA Tables	2512
20.30.1. The INFORMATION_SCHEMA TP_THREAD_STATE Table	2512
20.30.2. The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table	2513
20.30.3. The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table	2514
20.31. Extensions to SHOW Statements	2516

`INFORMATION_SCHEMA` provides access to database *metadata*, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are *data dictionary* and *system catalog*.

Usage Notes for the `INFORMATION_SCHEMA` Database

`INFORMATION_SCHEMA` is a database within each MySQL instance, the place that stores information about all the other databases that the MySQL server maintains. The `INFORMATION_SCHEMA` database contains several read-only tables. They are actually views, not base tables, so there are no files associated with them, and you cannot set triggers on them. Also, there is no database directory with that name.

Although you can select `INFORMATION_SCHEMA` as the default database with a `USE` statement, you can only read the contents of tables, not perform `INSERT`, `UPDATE`, or `DELETE` operations on them.

Example

Here is an example of a statement that retrieves information from `INFORMATION_SCHEMA`:

```
mysql> SELECT table_name, table_type, engine
-> FROM information_schema.tables
-> WHERE table_schema = 'db5'
-> ORDER BY table_name;
```

table_name	table_type	engine
fk	BASE TABLE	InnoDB
fk2	BASE TABLE	InnoDB
goto	BASE TABLE	MyISAM
into	BASE TABLE	MyISAM
k	BASE TABLE	MyISAM
kurs	BASE TABLE	MyISAM
loop	BASE TABLE	MyISAM
pk	BASE TABLE	InnoDB
t	BASE TABLE	MyISAM
t2	BASE TABLE	MyISAM
t3	BASE TABLE	MyISAM
t7	BASE TABLE	MyISAM
tables	BASE TABLE	MyISAM
v	VIEW	NULL
v2	VIEW	NULL
v3	VIEW	NULL
v56	VIEW	NULL

17 rows in set (0.01 sec)

Explanation: The statement requests a list of all the tables in database `db5`, showing just three pieces of information: the name of the table, its type, and its storage engine.

The definition for character columns (for example, `TABLES.TABLE_NAME`) is generally `VARCHAR(N) CHARACTER SET utf8` where `N` is at least 64. MySQL uses the default collation for this character set (`utf8_general_ci`) for all searches, sorts, comparisons, and other string operations on such columns. However, searches in `INFORMATION_SCHEMA` string columns are also affected by file system case sensitivity. For more information, see [Section 10.1.7.9, "Collation and `INFORMATION_SCHEMA` Searches"](#).

Each MySQL user has the right to access these tables, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the `ROUTINE_DEFINITION` column in the `INFORMATION_SCHEMA.ROUTINES` table), users who have insufficient privileges will see `NULL`.

Character Set Considerations

The definition for character columns (for example, `TABLES.TABLE_NAME`) is generally `VARCHAR(N) CHARACTER SET utf8` where `N` is at least 64. MySQL uses the default collation for this character set (`utf8_general_ci`) for all searches, sorts, comparisons, and other string operations on such columns.

Because some MySQL objects are represented as files, searches in `INFORMATION_SCHEMA` string columns can be affected by file system case sensitivity. For more information, see [Section 10.1.7.9, “Collation and `INFORMATION_SCHEMA` Searches”](#).

`INFORMATION_SCHEMA` as Alternative to `SHOW` Statements

The `SELECT ... FROM INFORMATION_SCHEMA` statement is intended as a more consistent way to provide access to the information provided by the various `SHOW` statements that MySQL supports (`SHOW DATABASES`, `SHOW TABLES`, and so forth). Using `SELECT` has these advantages, compared to `SHOW`:

- It conforms to Codd's rules, because all access is done on tables.
- You can use the familiar syntax of the `SELECT` statement, and only need to learn some table and column names.
- The implementor need not worry about adding keywords.
- You can filter, sort, concatenate, and transform the results from `INFORMATION_SCHEMA` queries into whatever format your application needs, such as a data structure or a text representation to parse.
- This technique is more interoperable with other database systems. For example, Oracle Database users are familiar with querying tables in the Oracle data dictionary.

Because `SHOW` is familiar and widely used, the `SHOW` statements remain as an alternative. In fact, along with the implementation of `INFORMATION_SCHEMA`, there are enhancements to `SHOW` as described in [Section 20.31, “Extensions to `SHOW` Statements”](#).

Privileges

Each MySQL user has the right to access these tables, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the `ROUTINE_DEFINITION` column in the `INFORMATION_SCHEMA.ROUTINES` table), users who have insufficient privileges see `NULL`. These restrictions do not apply for `InnoDB` tables; you can see them with only the `PROCESS [696]` privilege.

The same privileges apply to selecting information from `INFORMATION_SCHEMA` and viewing the same information through `SHOW` statements. In either case, you must have some privilege on an object to see information about it.

Performance Considerations

`INFORMATION_SCHEMA` queries that search for information from more than one database might take a long time and impact performance. To check the efficiency of a query, you can use `EXPLAIN`. For information about using `EXPLAIN` output to tune `INFORMATION_SCHEMA` queries, see [Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”](#).

Standards Considerations

The implementation for the `INFORMATION_SCHEMA` table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such column is the `ENGINE` column in the `INFORMATION_SCHEMA.TABLES` table.

Although other DBMSs use a variety of names, like `syscat` or `system`, the standard name is `INFORMATION_SCHEMA`.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked “MySQL extension”. (For example, we changed `COLLATION` to `TABLE_COLLATION` in the `TABLES` table.) See the list of reserved words near the end of this article: http://web.archive.org/web/20070409075643rn_1/www.dbazine.com/db2/db2-disarticles/gulutzan5.

Conventions in the `INFORMATION_SCHEMA` Reference Sections

The following sections describe each of the tables and columns in `INFORMATION_SCHEMA`. For each column, there are three pieces of information:

- “`INFORMATION_SCHEMA` Name” indicates the name for the column in the `INFORMATION_SCHEMA` table. This corresponds to the standard SQL name unless the “Remarks” field says “MySQL extension.”
- “`SHOW` Name” indicates the equivalent field name in the closest `SHOW` statement, if there is one.
- “Remarks” provides additional information where applicable. If this field is `NULL`, it means that the value of the column is always `NULL`. If this field says “MySQL extension,” the column is a MySQL extension to standard SQL.

Many sections indicate what `SHOW` statement is equivalent to a `SELECT` that retrieves information from `INFORMATION_SCHEMA`. For `SHOW` statements that display information for the default database if you omit a `FROM db_name` clause, you can often select information for the default database by adding an `AND TABLE_SCHEMA = SCHEMA()` condition to the `WHERE` clause of a query that retrieves information from an `INFORMATION_SCHEMA` table.

For information about `INFORMATION_SCHEMA` tables specific to the `InnoDB` storage engine, see [Section 20.28, “`INFORMATION_SCHEMA` Tables for `InnoDB`”](#). For information about `INFORMATION_SCHEMA` tables specific to the `NDB` storage engine (MySQL Cluster), see [Section 20.29, “`INFORMATION_SCHEMA` Tables for MySQL Cluster”](#). For information about `INFORMATION_SCHEMA` tables specific to the thread pool plugin, see [Section 20.30, “Thread Pool `INFORMATION_SCHEMA` Tables”](#).

For answers to questions that are often asked concerning the `INFORMATION_SCHEMA` database, see [Section B.7, “MySQL 5.5 FAQ: `INFORMATION_SCHEMA`”](#).

20.1. The `INFORMATION_SCHEMA.CHARACTER_SETS` Table

The `CHARACTER_SETS` table provides information about available character sets.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>CHARACTER_SET_NAME</code>	<code>Charset</code>	
<code>DEFAULT_COLLATE_NAME</code>	<code>Default collation</code>	
<code>DESCRIPTION</code>	<code>Description</code>	MySQL extension
<code>MAXLEN</code>	<code>Maxlen</code>	MySQL extension

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE CHARACTER_SET_NAME LIKE 'wild']

SHOW CHARACTER SET
  [LIKE 'wild']
```

20.2. The INFORMATION_SCHEMA COLLATIONS Table

The `COLLATIONS` table provides information about collations for each character set.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
<code>COLLATION_NAME</code>	Collation	
<code>CHARACTER_SET_NAME</code>	Charset	MySQL extension
<code>ID</code>	Id	MySQL extension
<code>IS_DEFAULT</code>	Default	MySQL extension
<code>IS_COMPILED</code>	Compiled	MySQL extension
<code>SORTLEN</code>	Sortlen	MySQL extension

- `COLLATION_NAME` is the collation name.
- `CHARACTER_SET_NAME` is the name of the character set with which the collation is associated.
- `ID` is the collation ID.
- `IS_DEFAULT` indicates whether the collation is the default for its character set.
- `IS_COMPILED` indicates whether the character set is compiled into the server.
- `SORTLEN` is related to the amount of memory required to sort strings expressed in the character set.

Collation information is also available from the `SHOW COLLATION` statement. The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE COLLATION_NAME LIKE 'wild']

SHOW COLLATION
  [LIKE 'wild']
```

20.3. The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The `COLLATION_CHARACTER_SET_APPLICABILITY` table indicates what character set is applicable for what collation. The columns are equivalent to the first two display fields that we get from `SHOW COLLATION`.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
<code>COLLATION_NAME</code>	Collation	
<code>CHARACTER_SET_NAME</code>	Charset	

20.4. The INFORMATION_SCHEMA COLUMNS Table

The `COLUMNS` table provides information about columns in tables.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		see notes
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE	Type	MySQL extension
COLUMN_KEY	Key	MySQL extension
EXTRA	Extra	MySQL extension
PRIVILEGES	Privileges	MySQL extension
COLUMN_COMMENT	Comment	MySQL extension

Notes:

- In `SHOW`, the `Type` display includes values from several different `COLUMNS` columns.
- `ORDINAL_POSITION` is necessary because you might want to say `ORDER BY ORDINAL_POSITION`. Unlike `SHOW`, `SELECT` does not have automatic ordering.
- `CHARACTER_OCTET_LENGTH` should be the same as `CHARACTER_MAXIMUM_LENGTH`, except for multi-byte character sets.
- `CHARACTER_SET_NAME` can be derived from `Collation`. For example, if you say `SHOW FULL COLUMNS FROM t`, and you see in the `Collation` column a value of `latin1_swedish_ci`, the character set is what is before the first underscore: `latin1`.

The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'tbl_name'
[AND table_schema = 'db_name']
[AND column_name LIKE 'wild']

SHOW COLUMNS
FROM tbl_name
[FROM db_name]
[LIKE 'wild']
```

20.5. The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table

The COLUMN_PRIVILEGES table provides information about column privileges. This information comes from the `mysql.columns_priv` grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		'user_name'@'host_name' value
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- In the output from `SHOW FULL COLUMNS`, the privileges are all in one field and in lowercase, for example, `select,insert,update,references`. In `COLUMN_PRIVILEGES`, there is one privilege per row, in uppercase.
- `PRIVILEGE_TYPE` can contain one (and only one) of these values: `SELECT` [697], `INSERT` [696], `UPDATE` [697], `REFERENCES` [696].
- If the user has `GRANT OPTION` [696] privilege, `IS_GRANTABLE` should be `YES`. Otherwise, `IS_GRANTABLE` should be `NO`. The output does not list `GRANT OPTION` [696] as a separate privilege.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
SHOW GRANTS ...
```

20.6. The INFORMATION_SCHEMA ENGINES Table

The PLUGINS table provides information about storage engines.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
ENGINE	Engine	MySQL extension
SUPPORT	Support	MySQL extension
COMMENT	Comment	MySQL extension
TRANSACTIONS	Transactions	MySQL extension
XA	XA	MySQL extension
SAVEPOINTS	Savepoints	MySQL extension

Notes:

- The `ENGINES` table is a nonstandard table.

See also [Section 13.7.5.17](#), “SHOW ENGINES Syntax”.

20.7. The INFORMATION_SCHEMA EVENTS Table

The `EVENTS` table provides information about scheduled events, which are discussed in [Section 19.4](#), “Using the Event Scheduler”. The `SHOW Name` values correspond to column names of the `SHOW EVENTS` statement.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
EVENT_CATALOG		def, MySQL extension
EVENT_SCHEMA	Db	MySQL extension
EVENT_NAME	Name	MySQL extension
DEFINER	Definer	MySQL extension
TIME_ZONE	Time zone	MySQL extension
EVENT_BODY		MySQL extension
EVENT_DEFINITION		MySQL extension
EVENT_TYPE	Type	MySQL extension
EXECUTE_AT	Execute at	MySQL extension
INTERVAL_VALUE	Interval value	MySQL extension
INTERVAL_FIELD	Interval field	MySQL extension
SQL_MODE		MySQL extension
STARTS	Starts	MySQL extension
ENDS	Ends	MySQL extension
STATUS	Status	MySQL extension
ON_COMPLETION		MySQL extension
CREATED		MySQL extension
LAST_ALTERED		MySQL extension
LAST_EXECUTED		MySQL extension
EVENT_COMMENT		MySQL extension
ORIGINATOR	Originator	MySQL extension
CHARACTER_SET_CLIENT	character_set_client	MySQL extension
COLLATION_CONNECTION	collation_connection [489]	MySQL extension
DATABASE_COLLATION	Database Collation	MySQL extension

Notes:

- The `EVENTS` table is a nonstandard table.
- `EVENT_CATALOG`: The value of this column is always `def`.
- `EVENT_SCHEMA`: The name of the schema (database) to which this event belongs.
- `EVENT_NAME`: The name of the event.
- `DEFINER`: The account of the user who created the event, in '`user_name`'@'`host_name`' format.

- **TIME_ZONE**: The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is **SYSTEM**.
- **EVENT_BODY**: The language used for the statements in the event's **DO** clause; in MySQL 5.5, this is always **SQL**.

This column is not to be confused with the column of the same name (now named **EVENT_DEFINITION**) that existed in earlier MySQL versions.

- **EVENT_DEFINITION**: The text of the SQL statement making up the event's **DO** clause; in other words, the statement executed by this event.
- **EVENT_TYPE**: The event repetition type, either **ONE TIME** (transient) or **RECURRING** (repeating).
- **EXECUTE_AT**: For a one-time event, this is the **DATETIME** value specified in the **AT** clause of the **CREATE EVENT** statement used to create the event, or of the last **ALTER EVENT** statement that modified the event. The value shown in this column reflects the addition or subtraction of any **INTERVAL** value included in the event's **AT** clause. For example, if an event is created using **ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR**, and the event was created at 2006-02-09 14:05:30, the value shown in this column would be `'2006-02-10 20:05:30'`.

If the event's timing is determined by an **EVERY** clause instead of an **AT** clause (that is, if the event is recurring), the value of this column is **NULL**.

- **INTERVAL_VALUE**: For recurring events, this column contains the numeric portion of the event's **EVERY** clause.

For a one-time event (that is, an event whose timing is determined by an **AT** clause), this column is **NULL**.

- **INTERVAL_FIELD**: For recurring events, this column contains the units portion of the **EVERY** clause governing the timing of the event. Thus, this column contains a value such as **'YEAR'**, **'QUARTER'**, **'DAY'**, and so on.

For a one-time event (that is, an event whose timing is determined by an **AT** clause), this column is **NULL**.

- **SQL_MODE**: The SQL mode in effect at the time the event was created or altered.
- **STARTS**: For a recurring event whose definition includes a **STARTS** clause, this column contains the corresponding **DATETIME** value. As with the **EXECUTE_AT** column, this value resolves any expressions used.

If there is no **STARTS** clause affecting the timing of the event, this column is **NULL**.

- **ENDS**: For a recurring event whose definition includes a **ENDS** clause, this column contains the corresponding **DATETIME** value. As with the **EXECUTE_AT** column, this value resolves any expressions used.

If there is no **ENDS** clause affecting the timing of the event, this column is **NULL**.

- **STATUS**: One of the three values **ENABLED**, **DISABLED**, or **SLAVESIDE_DISABLED**.

SLAVESIDE_DISABLED indicates that the creation of the event occurred on another MySQL server acting as a replication master and was replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave. See [Section 16.4.1.10, "Replication of Invoked Features"](#), for more information.

- **ON_COMPLETION**: One of the two values `PRESERVE` or `NOT PRESERVE`.
- **CREATED**: The date and time when the event was created. This is a `DATETIME` value.
- **LAST_ALTERED**: The date and time when the event was last modified. This is a `DATETIME` value. If the event has not been modified since its creation, this column holds the same value as the `CREATED` column.
- **LAST_EXECUTED**: The date and time when the event last executed. A `DATETIME` value. If the event has never executed, this column is `NULL`.

`LAST_EXECUTED` indicates when the event started. As a result, the `ENDS` column is never less than `LAST_EXECUTED`.
- **EVENT_COMMENT**: The text of a comment, if the event has one. If not, the value of this column is an empty string.
- **ORIGINATOR**: The server ID of the MySQL server on which the event was created; used in replication. The default value is 0.
- **CHARACTER_SET_CLIENT** is the session value of the `character_set_client` [486] system variable when the event was created. **COLLATION_CONNECTION** is the session value of the `collation_connection` [489] system variable when the event was created. **DATABASE_COLLATION** is the collation of the database with which the event is associated.

Example: Suppose that the user `jon@ghidora` creates an event named `e_daily`, and then modifies it a few minutes later using an `ALTER EVENT` statement, as shown here:

```
DELIMITER |
CREATE EVENT e_daily
ON SCHEDULE
EVERY 1 DAY
COMMENT 'Saves total number of sessions then clears the table each day'
DO
BEGIN
INSERT INTO site_activity.totals (time, total)
SELECT CURRENT_TIMESTAMP, COUNT(*)
FROM site_activity.sessions;
DELETE FROM site_activity.sessions;
END |
DELIMITER ;
ALTER EVENT e_daily
ENABLED;
```

(Note that comments can span multiple lines.)

This user can then run the following `SELECT` statement, and obtain the output shown:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME = 'e_daily'
> AND EVENT_SCHEMA = 'myschema'\G
***** 1. row *****
EVENT_CATALOG: def
EVENT_SCHEMA: test
EVENT_NAME: e_daily
DEFINER: paul@localhost
TIME_ZONE: SYSTEM
EVENT_BODY: SQL
EVENT_DEFINITION: BEGIN
```

```

INSERT INTO site_activity.totals (time, total)
  SELECT CURRENT_TIMESTAMP, COUNT(*)
    FROM site_activity.sessions;
DELETE FROM site_activity.sessions;
END
EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 1
INTERVAL_FIELD: DAY
SQL_MODE:
  STARTS: 2008-09-03 12:13:39
  ENDS: NULL
  STATUS: ENABLED
ON_COMPLETION: NOT PRESERVE
  CREATED: 2008-09-03 12:13:39
  LAST_ALTERED: 2008-09-03 12:13:39
  LAST_EXECUTED: NULL
EVENT_COMMENT: Saves total number of sessions then clears the
                table each day
ORIGINATOR: 1
CHARACTER_SET_CLIENT: latin1
COLLATION_CONNECTION: latin1_swedish_ci
DATABASE_COLLATION: latin1_swedish_ci

```

Times in the `EVENTS` table are displayed using the event time zone or the current session time zone, as described in [Section 19.4.4, “Event Metadata”](#).

See also [Section 13.7.5.19, “SHOW EVENTS Syntax”](#).

20.8. The `INFORMATION_SCHEMA GLOBAL_STATUS` and `SESSION_STATUS` Tables

The `GLOBAL_STATUS` and `SESSION_STATUS` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL STATUS` and `SHOW SESSION STATUS` statements (see [Section 13.7.5.36, “SHOW STATUS Syntax”](#)).

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>VARIABLE_NAME</code>	Variable_name	
<code>VARIABLE_VALUE</code>	Value	

Notes:

- The `VARIABLE_VALUE` column for each of these tables is defined as `VARCHAR(20480)`.

20.9. The `INFORMATION_SCHEMA GLOBAL_VARIABLES` and `SESSION_VARIABLES` Tables

The `GLOBAL_VARIABLES` and `SESSION_VARIABLES` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL VARIABLES` and `SHOW SESSION VARIABLES` statements (see [Section 13.7.5.40, “SHOW VARIABLES Syntax”](#)).

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>VARIABLE_NAME</code>	Variable_name	
<code>VARIABLE_VALUE</code>	Value	

Notes:

- The VARIABLE_VALUE column for each of these tables is defined as VARCHAR(20480).

20.10. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The KEY_COLUMN_USAGE table describes which key columns have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		def
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSITION_IN_UNIQUE_CONSTRAINT		
REFERENCED_TABLE_SCHEMA		
REFERENCED_TABLE_NAME		
REFERENCED_COLUMN_NAME		

Notes:

- If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
- The value of ORDINAL_POSITION is the column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.
- The value of POSITION_IN_UNIQUE_CONSTRAINT is NULL for unique and primary-key constraints. For foreign-key constraints, it is the ordinal position in key of the table that is being referenced.

Suppose that there are two tables name t1 and t3 that have the following definitions:

```
CREATE TABLE t1
(
  s1 INT,
  s2 INT,
  s3 INT,
  PRIMARY KEY(s3)
) ENGINE=InnoDB;

CREATE TABLE t3
(
  s1 INT,
  s2 INT,
  s3 INT,
  KEY(s1),
  CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

For those two tables, the KEY_COLUMN_USAGE table has two rows:

- One row with CONSTRAINT_NAME = 'PRIMARY', TABLE_NAME = 't1', COLUMN_NAME = 's3', ORDINAL_POSITION = 1, POSITION_IN_UNIQUE_CONSTRAINT = NULL.

- One row with `CONSTRAINT_NAME = 'CO'`, `TABLE_NAME = 't3'`, `COLUMN_NAME = 's2'`, `ORDINAL_POSITION = 1`, `POSITION_IN_UNIQUE_CONSTRAINT = 1`.

20.11. The INFORMATION_SCHEMA PARAMETERS Table

The `PARAMETERS` table provides information about stored procedure and function parameters, and about return values for stored functions. Parameter information is similar to the contents of the `param_list` column in the `mysql.proc` table.

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
<code>SPECIFIC_CATALOG</code>		<code>def</code>
<code>SPECIFIC_SCHEMA</code>	<code>db</code>	routine database
<code>SPECIFIC_NAME</code>	<code>name</code>	routine name
<code>ORDINAL_POSITION</code>		1, 2, 3, ... for parameters, 0 for function <code>RETURNS</code> clause
<code>PARAMETER_MODE</code>		<code>IN</code> , <code>OUT</code> , <code>INOUT</code> (<code>NULL</code> for <code>RETURNS</code>)
<code>PARAMETER_NAME</code>		parameter name (<code>NULL</code> for <code>RETURNS</code>)
<code>DATA_TYPE</code>		same as for <code>COLUMNS</code> table
<code>CHARACTER_MAXIMUM_LENGTH</code>		same as for <code>COLUMNS</code> table
<code>CHARACTER_OCTET_LENGTH</code>		same as for <code>COLUMNS</code> table
<code>NUMERIC_PRECISION</code>		same as for <code>COLUMNS</code> table
<code>NUMERIC_SCALE</code>		same as for <code>COLUMNS</code> table
<code>CHARACTER_SET_NAME</code>		same as for <code>COLUMNS</code> table
<code>COLLATION_NAME</code>		same as for <code>COLUMNS</code> table
<code>DTD_IDENTIFIER</code>		same as for <code>COLUMNS</code> table
<code>ROUTINE_TYPE</code>	<code>type</code>	same as for <code>ROUTINES</code> table

Notes:

- The `PARAMETERS` table was added in MySQL 5.5.3.
- For successive parameters of a stored procedure or function, the `ORDINAL_POSITION` values are 1, 2, 3, and so forth. For a stored function, there is also a row that describes the data type for the `RETURNS` clause. The return value is not a true parameter, so the row that describes it has these unique characteristics:
 - The `ORDINAL_POSITION` value is 0.
 - The `PARAMETER_NAME` and `PARAMETER_MODE` values are `NULL` because the return value has no name and the mode does not apply.

20.12. The INFORMATION_SCHEMA PARTITIONS Table

The `PARTITIONS` table provides information about table partitions. See [Chapter 18, Partitioning](#), for more information about partitioning tables.

The INFORMATION_SCHEMA PARTITIONS Table

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		MySQL extension
TABLE_SCHEMA		MySQL extension
TABLE_NAME		MySQL extension
PARTITION_NAME		MySQL extension
SUBPARTITION_NAME		MySQL extension
PARTITION_ORDINAL_POSITION		MySQL extension
SUBPARTITION_ORDINAL_POSITION		MySQL extension
PARTITION_METHOD		MySQL extension
SUBPARTITION_METHOD		MySQL extension
PARTITION_EXPRESSION		MySQL extension
SUBPARTITION_EXPRESSION		MySQL extension
PARTITION_DESCRIPTION		MySQL extension
TABLE_ROWS		MySQL extension
AVG_ROW_LENGTH		MySQL extension
DATA_LENGTH		MySQL extension
MAX_DATA_LENGTH		MySQL extension
INDEX_LENGTH		MySQL extension
DATA_FREE		MySQL extension
CREATE_TIME		MySQL extension
UPDATE_TIME		MySQL extension
CHECK_TIME		MySQL extension
CHECKSUM		MySQL extension
PARTITION_COMMENT		MySQL extension
NODEGROUP		MySQL extension
TABLESPACE_NAME		MySQL extension

Notes:

- The `PARTITIONS` table is a nonstandard table.
Each record in this table corresponds to an individual partition or subpartition of a partitioned table.
- `TABLE_CATALOG`: This column is always `def`.
- `TABLE_SCHEMA`: This column contains the name of the database to which the table belongs.
- `TABLE_NAME`: This column contains the name of the table containing the partition.
- `PARTITION_NAME`: The name of the partition.
- `SUBPARTITION_NAME`: If the `PARTITIONS` table record represents a subpartition, then this column contains the name of subpartition; otherwise it is `NULL`.
- `PARTITION_ORDINAL_POSITION`: All partitions are indexed in the same order as they are defined, with `1` being the number assigned to the first partition. The indexing can change as partitions are added,

dropped, and reorganized; the number shown in this column reflects the current order, taking into account any indexing changes.

- `SUBPARTITION_ORDINAL_POSITION`: Subpartitions within a given partition are also indexed and reindexed in the same manner as partitions are indexed within a table.
- `PARTITION_METHOD`: One of the values `RANGE`, `LIST`, `HASH`, `LINEAR HASH`, `KEY`, or `LINEAR KEY`; that is, one of the available partitioning types as discussed in [Section 18.2, “Partitioning Types”](#).
- `SUBPARTITION_METHOD`: One of the values `HASH`, `LINEAR HASH`, `KEY`, or `LINEAR KEY`; that is, one of the available subpartitioning types as discussed in [Section 18.2.6, “Subpartitioning”](#).
- `PARTITION_EXPRESSION`: This is the expression for the partitioning function used in the `CREATE TABLE` or `ALTER TABLE` statement that created the table's current partitioning scheme.

For example, consider a partitioned table created in the `test` database using this statement:

```
CREATE TABLE tp (
  c1 INT,
  c2 INT,
  c3 VARCHAR(25)
)
PARTITION BY HASH(c1 + c2)
PARTITIONS 4;
```

The `PARTITION_EXPRESSION` column in a `PARTITIONS` table record for a partition from this table displays `c1 + c2`, as shown here:

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';
+-----+
| PARTITION_EXPRESSION |
+-----+
| c1 + c2              |
+-----+
1 row in set (0.09 sec)
```

- `SUBPARTITION_EXPRESSION`: This works in the same fashion for the subpartitioning expression that defines the subpartitioning for a table as `PARTITION_EXPRESSION` does for the partitioning expression used to define a table's partitioning.

If the table has no subpartitions, then this column is `NULL`.

- `PARTITION_DESCRIPTION`: This column is used for `RANGE` and `LIST` partitions. For a `RANGE` partition, it contains the value set in the partition's `VALUES LESS THAN` clause, which can be either an integer or `MAXVALUE`. For a `LIST` partition, this column contains the values defined in the partition's `VALUES IN` clause, which is a comma-separated list of integer values.

For partitions whose `PARTITION_METHOD` is other than `RANGE` or `LIST`, this column is always `NULL`.

- `TABLE_ROWS`: The number of table rows in the partition.

For partitioned `InnoDB` tables, the row count given in the `TABLE_ROWS` column is only an estimated value used in SQL optimization, and may not always be exact.

For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `AVG_ROW_LENGTH`: The average length of the rows stored in this partition or subpartition, in bytes.

This is the same as `DATA_LENGTH` divided by `TABLE_ROWS`.

You can also obtain equivalent information using the `ndb_desc` utility.

- `DATA_LENGTH`: The total length of all rows stored in this partition or subpartition, in bytes—that is, the total number of bytes stored in the partition or subpartition.

For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `MAX_DATA_LENGTH`: The maximum number of bytes that can be stored in this partition or subpartition.

For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `INDEX_LENGTH`: The length of the index file for this partition or subpartition, in bytes.

For partitions of `NDB` tables, whether the tables use implicit or explicit partitioning, the `INDEX_LENGTH` column value is always 0. However, you can obtain equivalent information using the `ndb_desc` utility.

- `DATA_FREE`: The number of bytes allocated to the partition or subpartition but not used.

For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `CREATE_TIME`: The time of the partition's or subpartition's creation.
- `UPDATE_TIME`: The time that the partition or subpartition was last modified.
- `CHECK_TIME`: The last time that the table to which this partition or subpartition belongs was checked.



Note

Some storage engines do not update this time; for tables using these storage engines, this value is always `NULL`.

- `CHECKSUM`: The checksum value, if any; otherwise, this column is `NULL`.
- `PARTITION_COMMENT`: This column contains the text of any comment made for the partition.

In MySQL 5.5, the display width of this column is 80 characters, and partition comments which exceed this length are truncated to fit. This issue is fixed in MySQL 5.6. (Bug #11748924, Bug #37728)

The default value for this column is an empty string.

- `NODEGROUP`: This is the nodegroup to which the partition belongs. This is relevant only to MySQL Cluster tables; otherwise the value of this column is always 0.
- `TABLESPACE_NAME`: This column contains the name of the tablespace to which the partition belongs. Currently, the value of this column is always `DEFAULT`.
- A nonpartitioned table has one record in `INFORMATION_SCHEMA.PARTITIONS`; however, the values of the `PARTITION_NAME`, `SUBPARTITION_NAME`, `PARTITION_ORDINAL_POSITION`, `SUBPARTITION_ORDINAL_POSITION`, `PARTITION_METHOD`, `SUBPARTITION_METHOD`, `PARTITION_EXPRESSION`, `SUBPARTITION_EXPRESSION`, and `PARTITION_DESCRIPTION` columns are all `NULL`. (The `PARTITION_COMMENT` column in this case is blank.)

In MySQL 5.5, there is also only one record in the `PARTITIONS` table for a table using the `NDBCLUSTER` storage engine. The same columns are also `NULL` (or empty) as for a nonpartitioned table.

20.13. The `INFORMATION_SCHEMA PLUGINS` Table

The `PLUGINS` table provides information about server plugins.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>PLUGIN_NAME</code>	Name	MySQL extension
<code>PLUGIN_VERSION</code>		MySQL extension
<code>PLUGIN_STATUS</code>	Status	MySQL extension
<code>PLUGIN_TYPE</code>	Type	MySQL extension
<code>PLUGIN_TYPE_VERSION</code>		MySQL extension
<code>PLUGIN_LIBRARY</code>	Library	MySQL extension
<code>PLUGIN_LIBRARY_VERSION</code>		MySQL extension
<code>PLUGIN_AUTHOR</code>		MySQL extension
<code>PLUGIN_DESCRIPTION</code>		MySQL extension
<code>PLUGIN_LICENSE</code>		MySQL extension
<code>LOAD_OPTION</code>		MySQL extension

Notes:

- The `PLUGINS` table is a nonstandard table.
- `PLUGIN_NAME` is the name used to refer to the plugin in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`.
- `PLUGIN_VERSION` is the version from the plugin's general type descriptor.
- `PLUGIN_STATUS` indicates the plugin status, one of `ACTIVE`, `INACTIVE`, `DISABLED`, or `DELETED`.
- `PLUGIN_TYPE` indicates the type of plugin, such as `STORAGE ENGINE`, `INFORMATION_SCHEMA`, or `AUTHENTICATION`.
- `PLUGIN_TYPE_VERSION` is the version from the plugin's type-specific descriptor.
- `PLUGIN_LIBRARY` is the name of the plugin shared object file. This is the name used to refer to the plugin file in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`. This file is located in the directory named by the `plugin_dir` [537] system variable. If the library name is `NULL`, the plugin is compiled in and cannot be uninstalled with `UNINSTALL PLUGIN`.
- `PLUGIN_LIBRARY_VERSION` indicates the plugin API interface version.
- `PLUGIN_AUTHOR` names the plugin author.
- `PLUGIN_DESCRIPTION` provides a short description of the plugin.
- `PLUGIN_LICENSE` indicates how the plugin is licensed; for example, `GPL`.
- `LOAD_OPTION` indicates how the plugin was loaded. The value is `OFF`, `ON`, `FORCE`, or `FORCE_PLUS_PERMANENT`. See Section 5.1.8.1, "Installing and Uninstalling Plugins". This column was added in MySQL 5.5.7.

For plugins installed with `INSTALL PLUGIN`, the `PLUGIN_NAME` and `PLUGIN_LIBRARY` values are also registered in the `mysql.plugin` table.

These statements are equivalent:

```
SELECT
  PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE,
```

```

    PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;

SHOW PLUGINS;

```

For information about plugin data structures that form the basis of the information in the `PLUGINS` table, see [Section 23.2, “The MySQL Plugin API”](#).

Plugin information is also available using the `SHOW PLUGINS` statement. See [Section 13.7.5.26, “SHOW PLUGINS Syntax”](#).

20.14. The `INFORMATION_SCHEMA.PROCESSLIST` Table

The `PROCESSLIST` table provides information about which threads are running.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
ID	Id	MySQL extension
USER	User	MySQL extension
HOST	Host	MySQL extension
DB	db	MySQL extension
COMMAND	Command	MySQL extension
TIME	Time	MySQL extension
STATE	State	MySQL extension
INFO	Info	MySQL extension

For an extensive description of the table columns, see [Section 13.7.5.30, “SHOW PROCESSLIST Syntax”](#).

Notes:

- The `PROCESSLIST` table is a nonstandard table.
- Like the output from the corresponding `SHOW` statement, the `PROCESSLIST` table will only show information about your own threads, unless you have the `PROCESS [696]` privilege, in which case you will see information about other threads, too. As an anonymous user, you cannot see any rows at all.
- If an SQL statement refers to `INFORMATION_SCHEMA.PROCESSLIST`, then MySQL will populate the entire table once, when statement execution begins, so there is read consistency during the statement. There is no read consistency for a multi-statement transaction, though.

The following statements are equivalent:

```

SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST

SHOW FULL PROCESSLIST

```

20.15. The `INFORMATION_SCHEMA.PROFILING` Table

The `PROFILING` table provides statement profiling information. Its contents correspond to the information produced by the `SHOW PROFILES` and `SHOW PROFILE` statements (see [Section 13.7.5.32, “SHOW PROFILES Syntax”](#)). The table is empty unless the `profiling [539]` session variable is set to 1.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
QUERY_ID	Query_ID	

INFORMATION_SCHEMA Name	SHOW Name	Remarks
SEQ		
STATE	Status	
DURATION	Duration	
CPU_USER	CPU_user	
CPU_SYSTEM	CPU_system	
CONTEXT_VOLUNTARY	Context_voluntary	
CONTEXT_INVOLUNTARY	Context_involuntary	
BLOCK_OPS_IN	Block_ops_in	
BLOCK_OPS_OUT	Block_ops_out	
MESSAGES_SENT	Messages_sent	
MESSAGES_RECEIVED	Messages_received	
PAGE_FAULTS_MAJOR	Page_faults_major	
PAGE_FAULTS_MINOR	Page_faults_minor	
SWAPS	Swaps	
SOURCE_FUNCTION	Source_function	
SOURCE_FILE	Source_file	
SOURCE_LINE	Source_line	

Notes:

- `QUERY_ID` is a numeric statement identifier.
- `SEQ` is a sequence number indicating the display order for rows with the same `QUERY_ID` value.
- `STATE` is the profiling state to which the row measurements apply.
- `DURATION` indicates how long statement execution remained in the given state, in seconds.
- `CPU_USER` and `CPU_SYSTEM` indicate user and system CPU use, in seconds.
- `CONTEXT_VOLUNTARY` and `CONTEXT_INVOLUNTARY` indicate how many voluntary and involuntary context switches occurred.
- `BLOCK_OPS_IN` and `BLOCK_OPS_OUT` indicate the number of block input and output operations.
- `MESSAGES_SENT` and `MESSAGES_RECEIVED` indicate the number of communication messages sent and received.
- `PAGE_FAULTS_MAJOR` and `PAGE_FAULTS_MINOR` indicate the number of major and minor page faults.
- `SWAPS` indicates how many swaps occurred.
- `SOURCE_FUNCTION`, `SOURCE_FILE`, and `SOURCE_LINE` provide information indicating where in the source code the profiled state executes.

20.16. The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table

The REFERENTIAL_CONSTRAINTS table provides information about foreign keys.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		def
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
UNIQUE_CONSTRAINT_CATALOG		def
UNIQUE_CONSTRAINT_SCHEMA		
UNIQUE_CONSTRAINT_NAME		
MATCH_OPTION		
UPDATE_RULE		
DELETE_RULE		
TABLE_NAME		
REFERENCED_TABLE_NAME		

Notes:

- TABLE_NAME has the same value as TABLE_NAME in INFORMATION_SCHEMA.TABLE_CONSTRAINTS.
- CONSTRAINT_SCHEMA and CONSTRAINT_NAME identify the foreign key.
- UNIQUE_CONSTRAINT_SCHEMA, UNIQUE_CONSTRAINT_NAME, and REFERENCED_TABLE_NAME identify the referenced key.
- The only valid value at this time for MATCH_OPTION is NONE.
- The possible values for UPDATE_RULE or DELETE_RULE are CASCADE, SET NULL, SET DEFAULT, RESTRICT, NO ACTION.

20.17. The INFORMATION_SCHEMA ROUTINES Table

The ROUTINES table provides information about stored routines (both procedures and functions). The ROUTINES table does not include user-defined functions (UDFs).

The column named “mysql.proc name” indicates the mysql.proc table column that corresponds to the INFORMATION_SCHEMA.ROUTINES table column, if any.

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
SPECIFIC_NAME	specific_name	
ROUTINE_CATALOG		def
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	{PROCEDURE FUNCTION}
DATA_TYPE		same as for COLUMNS table
CHARACTER_MAXIMUM_LENGTH		same as for COLUMNS table
CHARACTER_OCTET_LENGTH		same as for COLUMNS table
NUMERIC_PRECISION		same as for COLUMNS table

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
NUMERIC_SCALE		same as for COLUMNS table
CHARACTER_SET_NAME		same as for COLUMNS table
COLLATION_NAME		same as for COLUMNS table
DTD_IDENTIFIER		data type descriptor
ROUTINE_BODY		SQL
ROUTINE_DEFINITION	body_utf8	
EXTERNAL_NAME		NULL
EXTERNAL_LANGUAGE	language	NULL
PARAMETER_STYLE		SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
SQL_PATH		NULL
SECURITY_TYPE	security_type	
CREATED	created	
LAST_ALTERED	modified	
SQL_MODE	sql_mode [556]	MySQL extension
ROUTINE_COMMENT	comment	MySQL extension
DEFINER	definer	MySQL extension
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension
DATABASE_COLLATION		MySQL extension

Notes:

- MySQL calculates EXTERNAL_LANGUAGE thus:
 - If `mysql.proc.language='SQL'`, EXTERNAL_LANGUAGE is NULL
 - Otherwise, EXTERNAL_LANGUAGE is what is in `mysql.proc.language`. However, we do not have external languages yet, so it is always NULL.
- CHARACTER_SET_CLIENT is the session value of the `character_set_client` [486] system variable when the routine was created. COLLATION_CONNECTION is the session value of the `collation_connection` [489] system variable when the routine was created. DATABASE_COLLATION is the collation of the database with which the routine is associated.
- The DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH, NUMERIC_PRECISION, NUMERIC_SCALE, CHARACTER_SET_NAME, and COLLATION_NAME columns provide information about the data type for the RETURNS clause of stored functions. If a stored routine is a stored procedure, these columns all are NULL. These columns were added in MySQL 5.5.3.
- Information about stored function RETURNS data types is also available in the PARAMETERS table. The return value data type row for a function can be identified as the row that has an ORDINAL_POSITION value of 0.

20.18. The INFORMATION_SCHEMA SCHEMATA Table

A schema is a database, so the `SCHEMATA` table provides information about databases.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>CATALOG_NAME</code>		def
<code>SCHEMA_NAME</code>		Database
<code>DEFAULT_CHARACTER_SET_NAME</code>		
<code>DEFAULT_COLLATION_NAME</code>		
<code>SQL_PATH</code>		NULL

The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`
FROM INFORMATION_SCHEMA.SCHEMATA
[WHERE SCHEMA_NAME LIKE 'wild']

SHOW DATABASES
[LIKE 'wild']
```

20.19. The `INFORMATION_SCHEMA` `SCHEMA_PRIVILEGES` Table

The `SCHEMA_PRIVILEGES` table provides information about schema (database) privileges. This information comes from the `mysql.db` grant table.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>GRANTEE</code>		' <i>user_name</i> '@' <i>host_name</i> ' value, MySQL extension
<code>TABLE_CATALOG</code>		def, MySQL extension
<code>TABLE_SCHEMA</code>		MySQL extension
<code>PRIVILEGE_TYPE</code>		MySQL extension
<code>IS_GRANTABLE</code>		MySQL extension

Notes:

- This is a nonstandard table. It takes its values from the `mysql.db` table.

20.20. The `INFORMATION_SCHEMA` `STATISTICS` Table

The `STATISTICS` table provides information about table indexes.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>TABLE_CATALOG</code>		def
<code>TABLE_SCHEMA</code>		= Database
<code>TABLE_NAME</code>	Table	
<code>NON_UNIQUE</code>	Non_unique	
<code>INDEX_SCHEMA</code>		= Database
<code>INDEX_NAME</code>	Key_name	
<code>SEQ_IN_INDEX</code>	Seq_in_index	
<code>COLUMN_NAME</code>	Column_name	
<code>COLLATION</code>	Collation	

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension

Notes:

- There is no standard table for indexes. The preceding list is similar to what SQL Server 2000 returns for `sp_statistics`, except that we replaced the name `QUALIFIER` with `CATALOG` and we replaced the name `OWNER` with `SCHEMA`.

Clearly, the preceding table and the output from `SHOW INDEX` are derived from the same parent. So the correlation is already close.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
     AND table_schema = 'db_name'

SHOW INDEX
  FROM tbl_name
  FROM db_name
```

20.21. The INFORMATION_SCHEMA TABLES Table

The `TABLES` table provides information about tables in databases.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		def
TABLE_SCHEMA	Table_...	
TABLE_NAME	Table_...	
TABLE_TYPE		
ENGINE	Engine	MySQL extension
VERSION	Version	The version number of the table's <code>.frm</code> file, MySQL extension
ROW_FORMAT	Row_format	MySQL extension
TABLE_ROWS	Rows	MySQL extension
AVG_ROW_LENGTH	Avg_row_length	MySQL extension
DATA_LENGTH	Data_length	MySQL extension
MAX_DATA_LENGTH	Max_data_length	MySQL extension
INDEX_LENGTH	Index_length	MySQL extension
DATA_FREE	Data_free	MySQL extension
AUTO_INCREMENT	Auto_increment	MySQL extension

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>CREATE_TIME</code>	<code>Create_time</code>	MySQL extension
<code>UPDATE_TIME</code>	<code>Update_time</code>	MySQL extension
<code>CHECK_TIME</code>	<code>Check_time</code>	MySQL extension
<code>TABLE_COLLATION</code>	<code>Collation</code>	MySQL extension
<code>CHECKSUM</code>	<code>Checksum</code>	MySQL extension
<code>CREATE_OPTIONS</code>	<code>Create_options</code>	MySQL extension
<code>TABLE_COMMENT</code>	<code>Comment</code>	MySQL extension

Notes:

- `TABLE_SCHEMA` and `TABLE_NAME` are a single field in a `SHOW` display, for example `Table_in_db1`.
- `TABLE_TYPE` should be `BASE TABLE` or `VIEW`. Currently, the `TABLES` table does not list `TEMPORARY` tables.
- For partitioned tables, the `ENGINE` column shows the name of the storage engine used by all partitions. (Previously, this column showed `PARTITION` for such tables.)
- The `TABLE_ROWS` column is `NULL` if the table is in the `INFORMATION_SCHEMA` database.

For `InnoDB` tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the `InnoDB` table is partitioned.)

- For `NDB` tables, `DATA_LENGTH` includes data stored in main memory only; the `MAX_DATA_LENGTH` and `DATA_FREE` columns apply to Disk Data.
- For MySQL Cluster Disk Data tables, `MAX_DATA_LENGTH` shows the space allocated for the disk part of a Disk Data table or fragment. (In-memory data resource usage is reported by the `DATA_LENGTH` column.)
- The `DATA_FREE` column shows the free space in bytes for `InnoDB` tables.

For MySQL Cluster, `DATA_FREE` shows the space allocated on disk for, but not used by, a Disk Data table or fragment on disk. (In-memory data resource usage is reported by the `DATA_LENGTH` column.)

- We have nothing for the table's default character set. `TABLE_COLLATION` is close, because collation names begin with a character set name.
- The `CREATE_OPTIONS` column shows `partitioned` if the table is partitioned.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
WHERE table_schema = 'db_name'
[AND table_name LIKE 'wild']

SHOW TABLES
FROM db_name
[LIKE 'wild']
```

20.22. The `INFORMATION_SCHEMA TABLESPACES` Table

The `TABLESPACES` table provides information about active tablespaces. The table was added in MySQL 5.5.3.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLESPACE_NAME		MySQL extension
ENGINE		MySQL extension
TABLESPACE_TYPE		MySQL extension
LOGFILE_GROUP_NAME		MySQL extension
EXTENT_SIZE		MySQL extension
AUTOEXTEND_SIZE		MySQL extension
MAXIMUM_SIZE		MySQL extension
NODEGROUP_ID		MySQL extension
TABLESPACE_COMMENT		MySQL extension

20.23. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The TABLE_CONSTRAINTS table describes which tables have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		def
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

Notes:

- The CONSTRAINT_TYPE value can be UNIQUE, PRIMARY KEY, or FOREIGN KEY.
- The UNIQUE and PRIMARY KEY information is about the same as what you get from the Key_name field in the output from SHOW INDEX when the Non_unique field is 0.
- The CONSTRAINT_TYPE column can contain one of these values: UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. This is a CHAR (not ENUM) column. The CHECK value is not available until we support CHECK.

20.24. The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The TABLE_PRIVILEGES table provides information about table privileges. This information comes from the mysql.tables_priv grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		'user_name'@'host_name' value
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		

INFORMATION_SCHEMA Name	SHOW Name	Remarks
IS_GRANTABLE		

Notes:

- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT [697], INSERT [696], UPDATE [697], REFERENCES [696], ALTER [695], INDEX [696], DROP [696], CREATE VIEW [696].

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
SHOW GRANTS ...
```

20.25. The INFORMATION_SCHEMA TRIGGERS Table

The TRIGGERS table provides information about triggers. You can see results only for databases and tables for which you have the TRIGGER [697] privilege.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TRIGGER_CATALOG		def
TRIGGER_SCHEMA		
TRIGGER_NAME	Trigger	
EVENT_MANIPULATION	Event	
EVENT_OBJECT_CATALOG		def
EVENT_OBJECT_SCHEMA		
EVENT_OBJECT_TABLE	Table	
ACTION_ORDER		0
ACTION_CONDITION		NULL
ACTION_STATEMENT	Statement	
ACTION_ORIENTATION		ROW
ACTION_TIMING	Timing	
ACTION_REFERENCE_OLD_TABLE		NULL
ACTION_REFERENCE_NEW_TABLE		NULL
ACTION_REFERENCE_OLD_ROW		OLD
ACTION_REFERENCE_NEW_ROW		NEW
CREATED		NULL (0)
SQL_MODE		MySQL extension
DEFINER		MySQL extension
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension
DATABASE_COLLATION		MySQL extension

Notes:

- The TRIGGER_SCHEMA and TRIGGER_NAME columns contain the name of the database in which the trigger occurs and the trigger name, respectively.

- The `EVENT_MANIPULATION` column contains one of the values 'INSERT', 'DELETE', or 'UPDATE'.
- As noted in [Section 19.3, “Using Triggers”](#), every trigger is associated with exactly one table. The `EVENT_OBJECT_SCHEMA` and `EVENT_OBJECT_TABLE` columns contain the database in which this table occurs, and the table's name.
- The `ACTION_ORDER` column contains the ordinal position of the trigger's action within the list of all similar triggers on the same table. Currently, this value is always 0, because it is not possible to have more than one trigger with the same `EVENT_MANIPULATION` and `ACTION_TIMING` on the same table.
- The `ACTION_STATEMENT` column contains the statement to be executed when the trigger is invoked. This is the same as the text displayed in the `Statement` column of the output from `SHOW TRIGGERS`. Note that this text uses UTF-8 encoding.
- The `ACTION_ORIENTATION` column always contains the value 'ROW'.
- The `ACTION_TIMING` column contains one of the two values 'BEFORE' or 'AFTER'.
- The columns `ACTION_REFERENCE_OLD_ROW` and `ACTION_REFERENCE_NEW_ROW` contain the old and new column identifiers, respectively. This means that `ACTION_REFERENCE_OLD_ROW` always contains the value 'OLD' and `ACTION_REFERENCE_NEW_ROW` always contains the value 'NEW'.
- The `SQL_MODE` column shows the server SQL mode that was in effect at the time when the trigger was created (and thus which remains in effect for this trigger whenever it is invoked, *regardless of the current server SQL mode*). The possible range of values for this column is the same as that of the `sql_mode` [556] system variable. See [Section 5.1.7, “Server SQL Modes”](#).
- The `DEFINER` column indicates who defined the trigger.
- `CHARACTER_SET_CLIENT` is the session value of the `character_set_client` [486] system variable when the trigger was created. `COLLATION_CONNECTION` is the session value of the `collation_connection` [489] system variable when the trigger was created. `DATABASE_COLLATION` is the collation of the database with which the trigger is associated.
- The following columns currently always contain NULL: `ACTION_CONDITION`, `ACTION_REFERENCE_OLD_TABLE`, `ACTION_REFERENCE_NEW_TABLE`, and `CREATED`.

Example, using the `ins_sum` trigger defined in [Section 19.3, “Using Triggers”](#):

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS\G
***** 1. row *****
      TRIGGER_CATALOG: def
      TRIGGER_SCHEMA: test
      TRIGGER_NAME: ins_sum
      EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
      EVENT_OBJECT_SCHEMA: test
      EVENT_OBJECT_TABLE: account
      ACTION_ORDER: 0
      ACTION_CONDITION: NULL
      ACTION_STATEMENT: SET @sum = @sum + NEW.amount
      ACTION_ORIENTATION: ROW
      ACTION_TIMING: BEFORE
      ACTION_REFERENCE_OLD_TABLE: NULL
      ACTION_REFERENCE_NEW_TABLE: NULL
      ACTION_REFERENCE_OLD_ROW: OLD
      ACTION_REFERENCE_NEW_ROW: NEW
      CREATED: NULL
      SQL_MODE:
      DEFINER: me@localhost
```

See also [Section 13.7.5.39](#), “`SHOW TRIGGERS Syntax`”.

20.26. The INFORMATION_SCHEMA USER_PRIVILEGES Table

The `USER_PRIVILEGES` table provides information about global privileges. This information comes from the `mysql.user` grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		' <i>user_name</i> '@' <i>host_name</i> ' value, MySQL extension
TABLE_CATALOG		<i>def</i> , MySQL extension
PRIVILEGE_TYPE		MySQL extension
IS_GRANTABLE		MySQL extension

Notes:

- This is a nonstandard table. It takes its values from the `mysql.user` table.

20.27. The INFORMATION_SCHEMA VIEWS Table

The `VIEWS` table provides information about views in databases. You must have the `SHOW VIEW` [\[697\]](#) privilege to access this table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		<i>def</i>
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
DEFINER		
SECURITY_TYPE		
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension

Notes:

- The `VIEW_DEFINITION` column has most of what you see in the `Create Table` field that `SHOW CREATE VIEW` produces. Skip the words before `SELECT` and skip the words `WITH CHECK OPTION`. Suppose that the original statement was:

```
CREATE VIEW v AS
  SELECT s2,s1 FROM t
  WHERE s1 > 5
  ORDER BY s1
  WITH CHECK OPTION;
```

Then the view definition looks like this:

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- The `CHECK_OPTION` column has a value of `NONE`, `CASCADE`, or `LOCAL`.
- MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable. If the view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it; for details, refer to [Section 13.1.20](#), “`CREATE VIEW Syntax`”.)
- The `DEFINER` column indicates who defined the view. `SECURITY_TYPE` has a value of `DEFINER` or `INVOKER`.
- `CHARACTER_SET_CLIENT` is the session value of the `character_set_client` [486] system variable when the view was created. `COLLATION_CONNECTION` is the session value of the `collation_connection` [489] system variable when the view was created.

MySQL lets you use different `sql_mode` [556] settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` [617] SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (`||`), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` [556] setting to a value different from `ANSI` [617] could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` [1102] function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as coll;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
       -> WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v';
+-----+
| VIEW_DEFINITION |
+-----+
| select concat('a','b') AS `coll` |
+-----+
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` [556] will not affect the results from the view. However an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

20.28. INFORMATION_SCHEMA Tables for InnoDB

The `InnoDB` tables related to the `InnoDB` storage engine help you to monitor ongoing `InnoDB` activity, to detect inefficiencies before they turn into issues, or to troubleshoot performance and capacity issues that do occur. As your database becomes bigger and busier, running up against the limits of your hardware capacity, you monitor and tune these aspects to keep the database running smoothly. The monitoring information deals with:

- `InnoDB` table compression, a feature whose use depends on a balance between I/O reduction, CPU usage, buffer pool management, and how much compression is possible for your data.
- Transactions and locks, features that balance high performance for a single operation, against the ability to run multiple operations concurrently. (Transactions are the high-level, user-visible aspect of concurrency. Locks are the low-level mechanism that transactions use to avoid reading or writing unreliable data.)

20.28.1. The `INFORMATION_SCHEMA.INNODB_CMP` and `INNODB_CMP_RESET` Tables

The `INNODB_CMP` and `INNODB_CMP_RESET` tables contain status information on operations related to compressed `InnoDB` tables.

Table 20.1. Columns of `INNODB_CMP` and `INNODB_CMP_RESET`

Column name	Description
<code>PAGE_SIZE</code>	Compressed page size in bytes.
<code>COMPRESS_OPS</code>	Number of times a B-tree page of the size <code>PAGE_SIZE</code> has been compressed. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.
<code>COMPRESS_OPS_OK</code>	Number of times a B-tree page of the size <code>PAGE_SIZE</code> has been successfully compressed. This count should never exceed <code>COMPRESS_OPS</code> .
<code>COMPRESS_TIME</code>	Total time in seconds spent in attempts to compress B-tree pages of the size <code>PAGE_SIZE</code> .
<code>UNCOMPRESS_OPS</code>	Number of times a B-tree page of the size <code>PAGE_SIZE</code> has been uncompressed. B-tree pages are uncompressed whenever compression fails or at first access when the uncompressed page does not exist in the buffer pool.
<code>UNCOMPRESS_TIME</code>	Total time in seconds spent in uncompressing B-tree pages of the size <code>PAGE_SIZE</code> .

Notes:

- Use these tables to measure the effectiveness of `InnoDB` table compression in your database.
- You must have the `PROCESS` privilege to query this table.
- For usage information, see [Section 14.4.6.1.3, “Using the Compression Information Schema Tables”](#).

20.28.2. The `INFORMATION_SCHEMA.INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` Tables

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables contain status information on compressed pages within the `InnoDB` buffer pool.

Table 20.2. Columns of `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET`

Column name	Description
<code>PAGE_SIZE</code>	Block size in bytes. Each record of this table describes blocks of this size.
<code>PAGES_USED</code>	Number of blocks of the size <code>PAGE_SIZE</code> that are currently in use.
<code>PAGES_FREE</code>	Number of blocks of the size <code>PAGE_SIZE</code> that are currently available for allocation. This column shows the external fragmentation in the memory pool. Ideally, these numbers should be at most 1.
<code>RELOCATION_OPS</code>	Number of times a block of the size <code>PAGE_SIZE</code> has been relocated. The buddy system can relocate the allocated “buddy neighbor” of a freed block when it tries to form a bigger freed block. Reading from the table <code>INNODB_CMPMEM_RESET</code> resets this count.

Column name	Description
<code>RELOCATION_TIME</code>	Total time in microseconds spent in relocating blocks of the size <code>PAGE_SIZE</code> . Reading from the table <code>INNODB_CMPMEM_RESET</code> resets this count.

Notes:

- Use these tables to measure the effectiveness of `InnoDB` table compression in your database.
- You must have the `PROCESS` privilege to query this table.
- For usage information, see [Section 14.4.6.1.3, “Using the Compression Information Schema Tables”](#).

20.28.3. The `INFORMATION_SCHEMA INNODB_TRX` Table

The `INNODB_TRX` table contains information about every transaction currently executing inside `InnoDB`, including whether the transaction is waiting for a lock, when the transaction started, and the SQL statement the transaction is executing.

Table 20.3. `INNODB_TRX` Columns

Column name	Description
<code>TRX_ID</code>	Unique transaction ID number, internal to <code>InnoDB</code> . (Starting in MySQL 5.6, these IDs are not created for transactions that are read-only and non-locking. See Optimizations for Read-Only Transactions for details.)
<code>TRX_WEIGHT</code>	The weight of a transaction, reflecting (but not necessarily the exact count of) the number of rows altered and the number of rows locked by the transaction. To resolve a deadlock, <code>InnoDB</code> selects the transaction with the smallest weight as the “victim” to rollback. Transactions that have changed non-transactional tables are considered heavier than others, regardless of the number of altered and locked rows.
<code>TRX_STATE</code>	Transaction execution state. One of <code>RUNNING</code> , <code>LOCK WAIT</code> , <code>ROLLING BACK</code> or <code>COMMITTING</code> .
<code>TRX_STARTED</code>	Transaction start time.
<code>TRX_REQUESTED_LOCK_ID</code>	ID of the lock the transaction is currently waiting for (if <code>TRX_STATE</code> is <code>LOCK WAIT</code> , otherwise <code>NULL</code>). Details about the lock can be found by joining with <code>INNODB_LOCKS</code> on <code>LOCK_ID</code> .
<code>TRX_WAIT_STARTED</code>	Time when the transaction started waiting on the lock (if <code>TRX_STATE</code> is <code>LOCK WAIT</code> , otherwise <code>NULL</code>).
<code>TRX_MYSQL_THREAD_ID</code>	MySQL thread ID. Can be used for joining with <code>PROCESSLIST</code> on <code>ID</code> . See Section 14.4.6.3.3, “Possible Inconsistency with <code>PROCESSLIST</code>” .
<code>TRX_QUERY</code>	The SQL query that is being executed by the transaction.
<code>TRX_OPERATION_STATE</code>	The transaction's current operation, or <code>NULL</code> .
<code>TRX_TABLES_IN_USE</code>	The number of <code>InnoDB</code> tables used while processing the current SQL statement of this transaction.
<code>TRX_TABLES_LOCKED</code>	Number of <code>InnoDB</code> tables that currently have any locks. (Because these are row locks, not table locks, the tables can usually still be read from and written to by multiple transactions, despite some rows being locked.)

Column name	Description
TRX_LOCK_STRUCTS	The number of locks reserved by the transaction.
TRX_LOCK_MEMORY_BYTES	Total size taken up by the lock structures of this transaction in memory.
TRX_ROWS_LOCKED	Approximate number of rows locked by this transaction. The value might include delete-marked rows that are physically present but not visible to the transaction.
TRX_ROWS_MODIFIED	The number of modified and inserted rows in this transaction.
TRX_CONCURRENCY_TICKETS	A value indicating how much work the current transaction can do before being swapped out, as specified by the <code>innodb_concurrency_tickets</code> option.
TRX_ISOLATION_LEVEL	The isolation level of the current transaction.
TRX_UNIQUE_CHECKS	Whether unique checks are turned on or off for the current transaction. (They might be turned off during a bulk data load, for example.)
TRX_FOREIGN_KEY_CHECKS	Whether foreign key checks are turned on or off for the current transaction. (They might be turned off during a bulk data load, for example.)
TRX_LAST_FOREIGN_KEY_ERROR	Detailed error message for last FK error, or <code>NULL</code> .
TRX_ADAPTIVE_HASH_LATCHED	Whether or not the adaptive hash index is locked by the current transaction. (Only a single transaction at a time can modify the adaptive hash index.)
TRX_ADAPTIVE_HASH_TIMEOUT	Whether to relinquish the search latch immediately for the adaptive hash index, or reserve it across calls from MySQL. When there is no AHI contention, this value remains zero and statements reserve the latch until they finish. During times of contention, it counts down to zero, and statements release the latch immediately after each row lookup.
TRX_IS_READ_ONLY	A value of 1 indicates the transaction is read-only. (5.6.4 and up.)
TRX_AUTOCOMMIT_NON_LOCKING	A value of 1 indicates the transaction is a <code>SELECT</code> statement that does not use the <code>FOR UPDATE</code> or <code>LOCK IN SHARED MODE</code> clauses, and is executing with the <code>autocommit</code> setting turned on so that the transaction will only contain this one statement. (5.6.4 and up.) When this column and <code>TRX_IS_READ_ONLY</code> are both 1, <code>InnoDB</code> optimizes the transaction to reduce the overhead associated with transactions that change table data.

Notes:

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load.
- You must have the `PROCESS` privilege to query this table.
- For usage information, see [Section 14.4.6.2.4, “Using the Transaction Information Schema Tables”](#).

20.28.4. The INFORMATION_SCHEMA INNODB_LOCKS Table

The `INNODB_LOCKS` table contains information about each lock that an `InnoDB` transaction has requested but not yet acquired, and each lock that a transaction holds that is blocking another transaction.

Table 20.4. `INNODB_LOCKS` Columns

Column name	Description
<code>LOCK_ID</code>	Unique lock ID number, internal to <code>InnoDB</code> . Treat it as an opaque string. Although <code>LOCK_ID</code> currently contains <code>TRX_ID</code> , the format of the data in <code>LOCK_ID</code> is not guaranteed to remain the same in future releases. Do not write programs that parse the <code>LOCK_ID</code> value.
<code>LOCK_TRX_ID</code>	ID of the transaction holding this lock. Details about the transaction can be found by joining with <code>INNODB_TRX</code> on <code>TRX_ID</code> .
<code>LOCK_MODE</code>	Mode of the lock. One of <code>S</code> , <code>X</code> , <code>IS</code> , <code>IX</code> , <code>S_GAP</code> , <code>X_GAP</code> , <code>IS_GAP</code> , <code>IX_GAP</code> , or <code>AUTO_INC</code> for shared, exclusive, intention shared, intention exclusive row locks, shared and exclusive gap locks, intention shared and intention exclusive gap locks, and auto-increment table level lock, respectively. Refer to the sections Section 14.3.9.1, “InnoDB Lock Modes” and Section 14.3.9, “The InnoDB Transaction Model and Locking” for information on <code>InnoDB</code> locking.
<code>LOCK_TYPE</code>	Type of the lock. One of <code>RECORD</code> or <code>TABLE</code> for record (row) level or table level locks, respectively.
<code>LOCK_TABLE</code>	Name of the table that has been locked or contains locked records.
<code>LOCK_INDEX</code>	Name of the index if <code>LOCK_TYPE= 'RECORD'</code> , otherwise <code>NULL</code> .
<code>LOCK_SPACE</code>	Tablespace ID of the locked record if <code>LOCK_TYPE= 'RECORD'</code> , otherwise <code>NULL</code> .
<code>LOCK_PAGE</code>	Page number of the locked record if <code>LOCK_TYPE= 'RECORD'</code> , otherwise <code>NULL</code> .
<code>LOCK_REC</code>	Heap number of the locked record within the page if <code>LOCK_TYPE= 'RECORD'</code> , otherwise <code>NULL</code> .
<code>LOCK_DATA</code>	Primary key of the locked record if <code>LOCK_TYPE= 'RECORD'</code> , otherwise <code>NULL</code> . This column contains the value(s) of the primary key column(s) in the locked row, formatted as a valid SQL string (ready to be copied to SQL commands). If there is no primary key then the <code>InnoDB</code> internal unique row ID number is used. When the page containing the locked record is not in the buffer pool (in the case that it was paged out to disk while the lock was held), <code>InnoDB</code> does not fetch the page from disk, to avoid unnecessary disk operations. Instead, <code>LOCK_DATA</code> is set to <code>NULL</code> .

Notes:

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load.
- You must have the `PROCESS` privilege to query this table.
- For usage information, see [Section 14.4.6.2.4, “Using the Transaction Information Schema Tables”](#).

20.28.5. The `INFORMATION_SCHEMA.INNODB_LOCK_WAITS` Table

The `INNODB_LOCK_WAITS` table contains one or more rows for each blocked `InnoDB` transaction, indicating the lock it has requested and any locks that are blocking that request.

Table 20.5. `INNODB_LOCK_WAITS` Columns

Column name	Description
<code>REQUESTING_TRX_ID</code>	ID of the requesting transaction.
<code>REQUESTED_LOCK_ID</code>	ID of the lock for which a transaction is waiting. Details about the lock can be found by joining with <code>INNODB_LOCKS</code> on <code>LOCK_ID</code> .
<code>BLOCKING_TRX_ID</code>	ID of the blocking transaction.

Column name	Description
<code>BLOCKING_LOCK_ID</code>	ID of a lock held by a transaction blocking another transaction from proceeding. Details about the lock can be found by joining with <code>INNODB_LOCKS</code> on <code>LOCK_ID</code> .

Notes:

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load.
- You must have the `PROCESS` privilege to query this table.
- For usage information, see [Section 14.4.6.2.4, “Using the Transaction Information Schema Tables”](#).

20.28.6. The `INFORMATION_SCHEMA INNODB_BUFFER_PAGE` Table

The `INNODB_BUFFER_PAGE` table holds information about each `page` in the `InnoDB` buffer pool.

Table 20.6. `INNODB_BUFFER_PAGE` Columns

Column name	Description
<code>POOL_ID</code>	Buffer Pool ID. An identifier to distinguish between multiple buffer pool instances.
<code>BLOCK_ID</code>	Buffer Pool Block ID.
<code>SPACE</code>	Tablespace ID. Uses the same value as in <code>INNODB_SYS_TABLES.SPACE</code> .
<code>PAGE_NUMBER</code>	Page number.
<code>PAGE_TYPE</code>	Page type string. One of <code>allocated</code> (Freshly allocated page), <code>index</code> (B-tree node), <code>undo_log</code> (Undo log page), <code>inode</code> (Index node), <code>ibuf_free_list</code> (Insert buffer free list), <code>ibuf_bitmap</code> (Insert buffer bitmap), <code>system</code> (System page), <code>trx_system</code> (Transaction system data), <code>file_space_header</code> (File space header), <code>extent_descriptor</code> (Extent descriptor page), <code>blob</code> (Uncompressed BLOB page), <code>compressed_blob</code> (First compressed BLOB page), <code>compressed_blob2</code> (Subsequent comp BLOB page), <code>unknown</code> (unknown).
<code>FLUSH_TYPE</code>	Flush type.
<code>FIX_COUNT</code>	Number of threads using this block within the buffer pool. When zero, the block is eligible to be evicted.
<code>IS_HASHED</code>	Whether hash index has been built on this page.
<code>NEWEST_MODIFICATION</code>	Log Sequence Number of the youngest modification.
<code>OLDEST_MODIFICATION</code>	Log Sequence Number of the oldest modification.
<code>ACCESS_TIME</code>	Time of first access.
<code>TABLE_NAME</code>	Name of the table the page belongs to.
<code>INDEX_NAME</code>	Name of the index the page belongs to. It can be the name of a clustered index or a secondary index.
<code>NUMBER_RECORDS</code>	Number of records within the page.
<code>DATA_SIZE</code>	Sum of the sizes of the records.
<code>COMPRESSED_SIZE</code>	Compressed page size. Null for pages that are not compressed.
<code>PAGE_STATE</code>	Page state. A page with valid data has one of the following states: <code>FILE_PAGE</code> (buffers a page of data from a file), <code>MEMORY</code> (buffers a page

Column name	Description
	from an in-memory object), Other possible states (managed by <code>InnoDB</code>) are: null, <code>READY_FOR_USE</code> , <code>NOT_USED</code> , <code>REMOVE_HASH</code> .
<code>IO_FIX</code>	Specifies whether any I/O is pending for this page: <code>IO_NONE</code> = no pending I/O, <code>IO_READ</code> = read pending, <code>IO_WRITE</code> = write pending.
<code>IS_OLD</code>	<code>bpage->old</code> .
<code>FREE_PAGE_CLOCK</code>	<code>bpage->freed_page_clock</code> .

Notes:

- This table is primarily useful for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- Since the `INFORMATION_SCHEMA` is a general-purpose way to monitor the MySQL server, use this table rather than the corresponding `InnoDB` system table for any new monitoring application you develop.
- You must have the `PROCESS` privilege to query this table.

20.28.7. The `INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU` Table

The `INNODB_BUFFER_PAGE_LRU` table holds information about the pages in the `InnoDB` buffer pool, in particular how they are ordered that determines which pages to `evict` from the buffer pool when it becomes full.

The definition for this page is the same as for `INNODB_BUFFER_PAGE`, except this table has an `LRU_POSITION` column instead of `BLOCK_ID`.

Notes:

- This table is primarily useful for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- Since the `INFORMATION_SCHEMA` is a general-purpose way to monitor the MySQL server, use this table rather than the corresponding `InnoDB` system table for any new monitoring application you develop.
- Querying this table can require MySQL to allocate a large block of contiguous memory, more than 64 bytes time the number of active pages in the buffer pool. This allocation could potentially cause an out-of-memory error, especially for systems with multi-gigabyte buffer pools.
 - Querying this table requires MySQL to lock the data structure representing the buffer pool while traversing the LRU list, which can reduce concurrency, especially for systems with multi-gigabyte buffer pools.

20.28.8. The `INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS` Table

The `INNODB_BUFFER_POOL_STATS` table represents the same kinds of `InnoDB` buffer pool information as the output from `SHOW ENGINE INNODB STATUS`.

The idea of making pages in the buffer pool “young” or “not young” refers to transferring them between the `sublists` at the head and tail of the buffer pool data structure. Pages made “young” take longer to age out of the buffer pool, while pages made “not young” are moved much closer to the point of `eviction`.

Column meanings in this table are intended to be self-explanatory for developers who formerly parsed the `SHOW ENGINE INNODB STATUS` to get these values.

Table 20.7. INNODB_BUFFER_POOL_STATS Columns

Column name	Description
POOL_ID	
POOL_SIZE	
FREE_BUFFERS	
DATABASE_PAGES	
OLD_DATABASE_PAGES	
MODIFIED_DATABASE_PAGES	
PENDING_DECOMPRESS	
PENDING_READS	
PENDING_FLUSH_LRU	
PENDING_FLUSH_LIST	
PAGES_MADE_YOUNG	
PAGES_NOT_MADE_YOUNG	
PAGES_MADE_YOUNG_RATE	
PAGES_MADE_NOT_YOUNG_RATE	
NUMBER_PAGES_READ	
NUMBER_PAGES_CREATED	
NUMBER_PAGES_WRITTEN	
PAGES_READ_RATE	
PAGES_CREATE_RATE	
PAGES_WRITTEN_RATE	
NUMBER_PAGES_GET	
HIT_RATE	
YOUNG_MAKE_PER_THOUSAND_GETS	
NOT_YOUNG_MAKE_PER_THOUSAND_GETS	
NUMBER_PAGES_READ_AHEAD	
NUMBER_READ_AHEAD_EVICTED	
READ_AHEAD_RATE	
READ_AHEAD_EVICTED_RATE	
LRU_IO_TOTAL	
LRU_IO_CURRENT	
UNCOMPRESS_TOTAL	
UNCOMPRESS_CURRENT	

Notes:

- This table is primarily useful for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- Since the `INFORMATION_SCHEMA` is a general-purpose way to monitor the MySQL server, use this table rather than the corresponding `InnoDB` system table for any new monitoring application you develop.

- You must have the `PROCESS` privilege to query this table.

20.29. INFORMATION_SCHEMA Tables for MySQL Cluster

The following sections provide information about `INFORMATION_SCHEMA` tables which are specific to MySQL Cluster. The `FILES` table was added in MySQL Server 5.1.6 as part of MySQL Cluster data-on-disk support (it is available in standard MySQL 5.5 but is not used there). The `ndb_transid_mysql_connection_map` table was added as part of additions made to MySQL Cluster's `ndbinfo` information database in MySQL Cluster NDB 7.2.2; it is implemented as an `INFORMATION_SCHEMA` plugin available only in MySQL Cluster binaries or source, and does not exist in MySQL Server 5.5.

Additional statistical and other data about MySQL Cluster transactions, operations, threads, blocks, and other aspects of performance can be obtained from the tables in the `ndbinfo` database. Information about these tables, see [Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”](#).

20.29.1. The INFORMATION_SCHEMA FILES Table

The `FILES` table provides information about the files in which MySQL NDB Disk Data tables are stored.



Note

This table provides information about Disk Data *files* only; you cannot use it for determining disk space allocation or availability for individual NDB tables. However, it is possible to see how much space is allocated for each NDB table having data stored on disk—as well as how much remains available for storage of data on disk for that table—using `ndb_desc`. For more information, see [Section 17.4.9, “`ndb_desc` — Describe NDB Tables”](#).

INFORMATION_SCHEMA Name	SHOW Name	Remarks
<code>FILE_ID</code>		MySQL extension
<code>FILE_NAME</code>		MySQL extension
<code>FILE_TYPE</code>		MySQL extension
<code>TABLESPACE_NAME</code>		MySQL extension
<code>TABLE_CATALOG</code>		MySQL extension
<code>TABLE_SCHEMA</code>		MySQL extension
<code>TABLE_NAME</code>		MySQL extension
<code>LOGFILE_GROUP_NAME</code>		MySQL extension
<code>LOGFILE_GROUP_NUMBER</code>		MySQL extension
<code>ENGINE</code>		MySQL extension
<code>FULLTEXT_KEYS</code>		MySQL extension
<code>DELETED_ROWS</code>		MySQL extension
<code>UPDATE_COUNT</code>		MySQL extension
<code>FREE_EXTENTS</code>		MySQL extension
<code>TOTAL_EXTENTS</code>		MySQL extension
<code>EXTENT_SIZE</code>		MySQL extension
<code>INITIAL_SIZE</code>		MySQL extension
<code>MAXIMUM_SIZE</code>		MySQL extension

INFORMATION_SCHEMA Name	SHOW Name	Remarks
AUTOEXTEND_SIZE		MySQL extension
CREATION_TIME		MySQL extension
LAST_UPDATE_TIME		MySQL extension
LAST_ACCESS_TIME		MySQL extension
RECOVER_TIME		MySQL extension
TRANSACTION_COUNTER		MySQL extension
VERSION		MySQL extension
ROW_FORMAT		MySQL extension
TABLE_ROWS		MySQL extension
AVG_ROW_LENGTH		MySQL extension
DATA_LENGTH		MySQL extension
MAX_DATA_LENGTH		MySQL extension
INDEX_LENGTH		MySQL extension
DATA_FREE		MySQL extension
CREATE_TIME	CREATE_TIME	MySQL extension
UPDATE_TIME		MySQL extension
CHECK_TIME		MySQL extension
CHECKSUM		MySQL extension
STATUS		MySQL extension
EXTRA		MySQL extension

Notes:

- `FILE_ID` column values are auto-generated.
- `FILE_NAME` is the name of an `UNDO` log file created by `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP`, or of a data file created by `CREATE TABLESPACE` or `ALTER TABLESPACE`.
- `FILE_TYPE` is one of the values `UNDOFILE`, `DATAFILE`, or `TABLESPACE`.
- `TABLESPACE_NAME` is the name of the tablespace with which the file is associated.
- Currently, the value of the `TABLESPACE_CATALOG` column is always `NULL`.
- `TABLE_NAME` is the name of the Disk Data table with which the file is associated, if any.
- The `LOGFILE_GROUP_NAME` column gives the name of the log file group to which the log file or data file belongs.
- For an `UNDO` log file, the `LOGFILE_GROUP_NUMBER` contains the auto-generated ID number of the log file group to which the log file belongs.
- For a MySQL Cluster Disk Data log file or data file, the value of the `ENGINE` column is always `NDB` or `NDBCLUSTER`.
- For a MySQL Cluster Disk Data log file or data file, the value of the `FULLTEXT_KEYS` column is always empty.

- The `FREE_EXTENTS` column displays the number of extents which have not yet been used by the file. The `TOTAL_EXTENTS` column show the total number of extents allocated to the file.

The difference between these two columns is the number of extents currently in use by the file:

```
SELECT TOTAL_EXTENTS - FREE_EXTENTS AS extents_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

You can approximate the amount of disk space in use by the file by multiplying this difference by the value of the `EXTENT_SIZE` column, which gives the size of an extent for the file in bytes:

```
SELECT (TOTAL_EXTENTS - FREE_EXTENTS) * EXTENT_SIZE AS bytes_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

Similarly, you can estimate the amount of space that remains available in a given file by multiplying `FREE_EXTENTS` by `EXTENT_SIZE`:

```
SELECT FREE_EXTENTS * EXTENT_SIZE AS bytes_free
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```



Important

The byte values produced by the preceding queries are approximations only, and their precision is inversely proportional to the value of `EXTENT_SIZE`. That is, the larger `EXTENT_SIZE` becomes, the less accurate the approximations are.

It is also important to remember that once an extent is used, it cannot be freed again without dropping the data file of which it is a part. This means that deletes from a Disk Data table do *not* release disk space.

The extent size can be set in a `CREATE TABLESPACE` statement. See [Section 13.1.18, “CREATE TABLESPACE Syntax”](#), for more information.

- The `INITIAL_SIZE` column shows the size in bytes of the file. This is the same value that was used in the `INITIAL_SIZE` clause of the `CREATE LOGFILE GROUP`, `ALTER LOGFILE GROUP`, `CREATE TABLESPACE`, or `ALTER TABLESPACE` statement used to create the file.

For MySQL Cluster Disk Data files, the value of the `MAXIMUM_SIZE` column is always the same as `INITIAL_SIZE`, and the `AUTOEXTEND_SIZE` column is always empty.

- The `CREATION_TIME` column shows the date and time when the file was created. The `LAST_UPDATE_TIME` column displays the date and time when the file was last modified. The `LAST_ACCESSED` column provides the date and time when the file was last accessed by the server.

Currently, the values of these columns are as reported by the operating system, and are not supplied by the `NDB` storage engine. Where no value is provided by the operating system, these columns display `0000-00-00 00:00:00`.

- For MySQL Cluster Disk Data files, the value of the `RECOVER_TIME` and `TRANSACTION_COUNTER` columns is always `0`.
- For MySQL Cluster Disk Data files, the following columns are always `NULL`:
 - `VERSION`
 - `ROW_FORMAT`

- TABLE_ROWS
- AVG_ROW_LENGTH
- DATA_LENGTH
- MAX_DATA_LENGTH
- INDEX_LENGTH
- DATA_FREE
- CREATE_TIME
- UPDATE_TIME
- CHECK_TIME
- CHECKSUM
- For MySQL Cluster Disk Data files, the value of the STATUS column is always NORMAL.
- For MySQL Cluster Disk Data files, the EXTRA column shows which data node the file belongs to, as each data node has its own copy of the file. Suppose that you use this statement on a MySQL Cluster with four data nodes:

```
CREATE LOGFILE GROUP mygroup
  ADD UNDOFILE 'new_undo.dat'
  INITIAL_SIZE 2G
  ENGINE NDB;
```

After running the CREATE LOGFILE GROUP statement successfully, you should see a result similar to the one shown here for this query against the FILES table:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE FILE_NAME = 'new_undo.dat';
```

LOGFILE_GROUP_NAME	FILE_TYPE	EXTRA
mygroup	UNDO FILE	CLUSTER_NODE=3
mygroup	UNDO FILE	CLUSTER_NODE=4
mygroup	UNDO FILE	CLUSTER_NODE=5
mygroup	UNDO FILE	CLUSTER_NODE=6

4 rows in set (0.01 sec)

- The FILES table is a nonstandard table.
- An additional row is present in the FILES table following the creation of a logfile group. This row has NULL for the value of the FILE_NAME column. For this row, the value of the FILE_ID column is always 0, that of the FILE_TYPE column is always UNDO FILE, and that of the STATUS column is always NORMAL. Currently, the value of the ENGINE column is always NDBCLUSTER.

The FREE_EXTENTS column in this row shows the total number of free extents available to all undo files belonging to a given log file group whose name and number are shown in the LOGFILE_GROUP_NAME and LOGFILE_GROUP_NUMBER columns, respectively.

Suppose there are no existing log file groups on your MySQL Cluster, and you create one using the following statement:

```
mysql> CREATE LOGFILE GROUP lg1
-> ADD UNDOFILE 'undofile.dat'
-> INITIAL_SIZE = 16M
-> UNDO_BUFFER_SIZE = 1M
-> ENGINE = NDB;
Query OK, 0 rows affected (3.81 sec)
```

You can now see this NULL row when you query the FILES table:

```
mysql> SELECT DISTINCT
-> FILE_NAME AS File,
-> FREE_EXTENTS AS Free,
-> TOTAL_EXTENTS AS Total,
-> EXTENT_SIZE AS Size,
-> INITIAL_SIZE AS Initial
-> FROM INFORMATION_SCHEMA.FILES;
+-----+-----+-----+-----+-----+
| File          | Free      | Total    | Size  | Initial |
+-----+-----+-----+-----+-----+
| undofile.dat  | NULL      | 4194304  | 4     | 16777216 |
| NULL         | 4184068   | NULL     | 4     | NULL     |
+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

The total number of free extents available for undo logging is always somewhat less than the sum of the TOTAL_EXTENTS column values for all undo files in the log file group due to overhead required for maintaining the undo files. This can be seen by adding a second undo file to the log file group, then repeating the previous query against the FILES table:

```
mysql> ALTER LOGFILE GROUP lg1
-> ADD UNDOFILE 'undofile02.dat'
-> INITIAL_SIZE = 4M
-> ENGINE = NDB;
Query OK, 0 rows affected (1.02 sec)

mysql> SELECT DISTINCT
-> FILE_NAME AS File,
-> FREE_EXTENTS AS Free,
-> TOTAL_EXTENTS AS Total,
-> EXTENT_SIZE AS Size,
-> INITIAL_SIZE AS Initial
-> FROM INFORMATION_SCHEMA.FILES;
+-----+-----+-----+-----+-----+
| File          | Free      | Total    | Size  | Initial |
+-----+-----+-----+-----+-----+
| undofile.dat  | NULL      | 4194304  | 4     | 16777216 |
| undofile02.dat | NULL      | 1048576  | 4     | 4194304  |
| NULL         | 5223944   | NULL     | 4     | NULL     |
+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

The amount of free space in bytes which is available for undo logging by Disk Data tables using this log file group can be approximated by multiplying the number of free extents by the initial size:

```
mysql> SELECT
-> FREE_EXTENTS AS 'Free Extents',
-> FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg1'
-> AND FILE_NAME IS NULL;
+-----+-----+
| Free Extents | Free Bytes |
+-----+-----+
| 5223944     | 20895776  |
+-----+-----+
```

```
+-----+-----+
1 row in set (0.02 sec)
```

If you create a MySQL Cluster Disk Data table and then insert some rows into it, you can see approximately how much space remains for undo logging afterward, for example:

```
mysql> CREATE TABLESPACE ts1
-> ADD DATAFILE 'data1.dat'
-> USE LOGFILE GROUP lg1
-> INITIAL_SIZE 512M
-> ENGINE = NDB;
Query OK, 0 rows affected (8.71 sec)

mysql> CREATE TABLE dd (
-> c1 INT NOT NULL PRIMARY KEY,
-> c2 INT,
-> c3 DATE
-> )
-> TABLESPACE ts1 STORAGE DISK
-> ENGINE = NDB;
Query OK, 0 rows affected (2.11 sec)

mysql> INSERT INTO dd VALUES
-> (NULL, 1234567890, '2007-02-02'),
-> (NULL, 1126789005, '2007-02-03'),
-> (NULL, 1357924680, '2007-02-04'),
-> (NULL, 1642097531, '2007-02-05');
Query OK, 4 rows affected (0.01 sec)

mysql> SELECT
-> FREE_EXTENTS AS 'Free Extents',
-> FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg1'
-> AND FILE_NAME IS NULL;
+-----+-----+
| Free Extents | Free Bytes |
+-----+-----+
|          5207565 |    20830260 |
+-----+-----+
1 row in set (0.01 sec)
```

- An additional row is present in the `FILES` table for any MySQL Cluster tablespace, whether or not any data files are associated with the tablespace. This row has `NULL` for the value of the `FILE_NAME` column. For this row, the value of the `FILE_ID` column is always 0, that of the `FILE_TYPE` column is always `TABLESPACE`, and that of the `STATUS` column is always `NORMAL`. Currently, the value of the `ENGINE` column is always `NDBCLUSTER`.
- There are no `SHOW` statements associated with the `FILES` table.
- For additional information, and examples of creating and dropping MySQL Cluster Disk Data objects, see [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#).

20.29.2. The `INFORMATION_SCHEMA ndb_transid_mysql_connection_map` Table

The `ndb_transid_mysql_connection_map` table provides a mapping between `NDB` transactions, `NDB` transaction coordinators, and MySQL Servers attached to a MySQL Cluster as API nodes. This information is used when populating the `server_operations` and `server_transactions` tables of the `ndbinfo` MySQL Cluster information database.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
mysql_connection_id		MySQL Server connection ID
node_id		Transaction coordinator node ID
ndb_transid		NDB transaction ID

The `mysql_connection_id` is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`.

There are no `SHOW` statements associated with this table.

This is a nonstandard table, added in MySQL Cluster NDB 7.2.2. It is implemented as an `INFORMATION_SCHEMA` plugin; you can verify that it is supported by checking the output of `SHOW PLUGINS`. If `ndb_transid_mysql_connection_map` support is enabled, the output from this statement includes a plugin having this name, of type `INFORMATION_SCHEMA`, and having status `ACTIVE`, as shown here (using emphasized text):

```
mysql> SHOW PLUGINS;
+-----+-----+-----+-----+-----+-----+
| Name          | Status | Type          | Library | License |
+-----+-----+-----+-----+-----+-----+
| binlog        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| mysql_native_password | ACTIVE | AUTHENTICATION | NULL    | GPL     |
| mysql_old_password | ACTIVE | AUTHENTICATION | NULL    | GPL     |
| CSV           | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MEMORY        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MRG_MYISAM    | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MyISAM        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| PERFORMANCE_SCHEMA | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| BLACKHOLE     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ARCHIVE       | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndbcluster    | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndbinfo       | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndb_transid_mysql_connection_map | ACTIVE | INFORMATION_SCHEMA | NULL | GPL |
| InnoDB        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| INNODB_TRX    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS  | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_RESET | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM_RESET | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| partition     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
+-----+-----+-----+-----+-----+-----+
22 rows in set (0.00 sec)
```

The plugin is enabled by default. You can disable it (or force the server not to run unless the plugin starts) by starting the server with the `--ndb-transid-mysql-connection-map [2145]` option. If the plugin is disabled, the status is shown by `SHOW PLUGINS` as `DISABLED`. The plugin cannot be enabled or disabled at runtime.

Although the names of this table and its columns are displayed using lowercase, you can use uppercase or lowercase when referring to them in SQL statements.

For this table to be created, the MySQL Server must be a binary that was supplied with the MySQL Cluster distribution, or one that was built from the MySQL Cluster sources with `NDB` storage engine support enabled. It is not available in the standard MySQL 5.5 Server.

20.30. Thread Pool `INFORMATION_SCHEMA` Tables

The following sections describe the `INFORMATION_SCHEMA` tables associated with the thread pool plugin. They provide information about thread pool operation:

- `TP_THREAD_STATE`: Information about thread pool thread states
- `TP_THREAD_GROUP_STATE`: Information about thread pool thread group states
- `TP_THREAD_GROUP_STATS`: Thread group statistics

Rows in these tables represent snapshots in time. In the case of `TP_THREAD_STATE`, all rows for a thread group comprise a snapshot in time. Thus, the MySQL server holds the mutex of the thread group while producing the snapshot. But it does not hold mutexes on all thread groups at the same time, to prevent a statement against `TP_THREAD_STATE` from blocking the entire MySQL server.

The thread pool `INFORMATION_SCHEMA` tables are implemented by individual plugins and the decision whether to load one can be made independently of the others (see [Section 8.11.6.1, “Thread Pool Components and Installation”](#)). However, the content of all the tables depends on the thread pool plugin being enabled. If a table plugin is enabled but the thread pool plugin is not, the table becomes visible and can be accessed but will be empty.

20.30.1. The `INFORMATION_SCHEMA TP_THREAD_STATE` Table

This table has one row per thread created by the thread pool to handle connections. The table has these columns:

- `TP_GROUP_ID`

The thread group ID.

- `TP_THREAD_NUMBER`

The ID of the thread within its thread group. `TP_GROUP_ID` and `TP_THREAD_NUMBER` together provide a unique key within the table.

- `PROCESS_COUNT`

The 10ms interval in which the statement that uses this thread is currently executing. 0 means no statement is executing, 1 means it is in the first 10ms, and so forth.

- `WAIT_TYPE`

The type of wait for the thread. `NULL` means the thread is not blocked. Otherwise, the thread is blocked by a call to `thd_wait_begin()` and the value specifies the type of wait. The `xxx_WAIT` columns of the `TP_THREAD_GROUP_STATS` table accumulate counts for each wait type.

The `WAIT_TYPE` value is a string that describes the type of wait, as shown in the following table.

Table 20.8. `WAIT_TYPE` Values

Wait Type	Meaning
<code>THD_WAIT_SLEEP</code>	Waiting for sleep
<code>THD_WAIT_DISKIO</code>	Waiting for Disk IO
<code>THD_WAIT_ROW_LOCK</code>	Waiting for row lock
<code>THD_WAIT_GLOBAL_LOCK</code>	Waiting for global lock

Wait Type	Meaning
<code>THD_WAIT_META_DATA_LOCK</code>	Waiting for metadata lock
<code>THD_WAIT_TABLE_LOCK</code>	Waiting for table lock
<code>THD_WAIT_USER_LOCK</code>	Waiting for user lock
<code>THD_WAIT_BINLOG</code>	Waiting for binlog
<code>THD_WAIT_GROUP_COMMIT</code>	Waiting for group commit
<code>THD_WAIT_SYNC</code>	Waiting for fsync

20.30.2. The `INFORMATION_SCHEMA TP_THREAD_GROUP_STATE` Table

This table has one row per thread group in the thread pool. Each row provides information about the current state of a group. The table has these columns:

- `TP_GROUP_ID`

The thread group ID. This is a unique key within the table.

- `CONSUMER_THREADS`

The number of consumer threads. There is at most one thread ready to start executing if the active threads become stalled or blocked.

- `RESERVE_THREADS`

The number of threads in the reserved state. This means that they will not be started until there is a need to wake a new thread and there is no consumer thread. This is where most threads end up when the thread group has created more threads than needed for normal operation. Often a thread group needs additional threads for a short while and then does not need them again for a while. In this case, they go into the reserved state and remain until needed again. They take up some extra memory resources, but no extra computing resources.

- `CONNECTION_COUNT`

The number of connections using this thread group.

- `QUEUED_QUERIES`

The number of statements waiting in the high-priority queue.

- `QUEUED_TRANSACTIONS`

The number of statements waiting in the low-priority queue. These are the initial statements for transactions that have not started, so they also represent queued transactions.

- `STALL_LIMIT`

The value of the `thread_pool_stall_limit` [566] variable on the thread group. This is the same value for all thread groups.

- `PRIO_KICKUP_TIMER`

The value of the `thread_pool_prio_kickup_timer` [565] on the thread group. This is the same value for all thread groups.

- `ALGORITHM`

The value of the `thread_pool_algorithm` [563] on the thread group. This is the same value for all thread groups.

- `THREAD_COUNT`

The number of threads started in the thread pool as part of this thread group.

- `ACTIVE_THREAD_COUNT`

The number of threads active executing statements.

- `MAX_THREAD_IDS_IN_GROUP`

The maximum thread ID of the threads in the group. This is the same as `MAX(TP_THREAD_NUMBER)` [1205] for the threads when selected from the `TP_THREAD_GROUP_STATE` table. That is, these two queries are equivalent:

```
SELECT TP_GROUP_ID, MAX_THREAD_IDS_IN_GROUP
FROM TP_THREAD_GROUP_STATE;

SELECT TP_GROUP_ID, MAX(TP_THREAD_NUMBER)
FROM TP_THREAD_STATE GROUP BY TP_GROUP_ID;
```

- `STALLED_THREAD_COUNT`

The number of stalled statements in the thread group. A stalled statement could be executing, but from a thread pool perspective it is stalled and making no progress. A long-running statement quickly ends up in this category.

- `WAITING_THREAD_NUMBER`

If there is a thread handling the polling of statements in the thread group, this specifies the thread number within this thread group. It is possible that this thread could be executing a statement.

- `OLDEST_QUEUED`

How long in milliseconds the oldest queued statement has been waiting for execution.

20.30.3. The `INFORMATION_SCHEMA TP_THREAD_GROUP_STATS` Table

This table reports statistics per thread group. There is one row per group. The table has these columns:

- `TP_GROUP_ID`

The thread group ID. This is a unique key within the table.

- `CONNECTIONS_STARTED`

The number of connections started.

- `CONNECTIONS_CLOSED`

The number of connections closed.

- `QUERIES_EXECUTED`

The number of statements executed. This number is incremented when a statement starts executing, not when it finishes.

- `QUERIES_QUEUED`

The number of statements received that were queued for execution. This does not count statements that the thread group was able to begin executing immediately without queuing, which can happen under the conditions described in [Section 8.11.6.2, “Thread Pool Operation”](#).

- `THREADS_STARTED`

The number of threads started.

- `PRIO_KICKUPS`

The number of statements that have been moved from low-priority queue to high-priority queue based on the value of the `thread_pool_prio_kickup_timer` [565] system variable. If this number increases quickly, consider increasing the value of that variable. A quickly increasing counter means that the priority system is not keeping transactions from starting too early. For `InnoDB`, this most likely means deteriorating performance due to too many concurrent transactions..

- `STALLED_QUERIES_EXECUTED`

The number of statements that have become defined as stalled due to executing for a time longer than the value of the `thread_pool_stall_limit` [566] system variable.

- `BECOME_CONSUMER_THREAD`

The number of times thread have been assigned the consumer thread role.

- `BECOME_RESERVE_THREAD`

The number of times threads have been assigned the reserve thread role.

- `BECOME_WAITING_THREAD`

The number of times threads have been assigned the waiter thread role. When statements are queued, this happens very often, even in normal operation, so rapid increases in this value are normal in the case of a highly loaded system where statements are queued up.

- `WAKE_THREAD_STALL_CHECKER`

The number of times the stall check thread decided to wake or create a thread to possibly handle some statements or take care of the waiter thread role.

- `SLEEP_WAITS`

The number of `THD_WAIT_SLEEP` waits. These occur when threads go to sleep; for example, by calling the `SLEEP()` [1201] function.

- `DISK_IO_WAITS`

The number of `THD_WAIT_DISKIO` waits. These occur when threads perform disk I/O that is likely to not hit the file system cache. Such waits occur when the buffer pool reads and writes data to disk, not for normal reads from and writes to files.

- `ROW_LOCK_WAITS`

The number of `THD_WAIT_ROW_LOCK` waits for release of a row lock by another transaction.

- `GLOBAL_LOCK_WAITS`

The number of `THD_WAIT_GLOBAL_LOCK` waits for a global lock to be released.

- `META_DATA_LOCK_WAITS`

The number of `THD_WAIT_META_DATA_LOCK` waits for a metadata lock to be released.

- `TABLE_LOCK_WAITS`

The number of `THD_WAIT_TABLE_LOCK` waits for a table to be unlocked that the statement needs to access.

- `USER_LOCK_WAITS`

The number of `THD_WAIT_USER_LOCK` waits for a special lock constructed by the user thread.

- `BINLOG_WAITS`

The number of `THD_WAIT_BINLOG_WAITS` waits for the binary log to become free.

- `GROUP_COMMIT_WAITS`

The number of `THD_WAIT_GROUP_COMMIT` waits. These occur when a group commit must wait for the other parties to complete their part of a transaction.

- `FSYNC_WAITS`

The number of `THD_WAIT_SYNC` waits for a file sync operation.

20.31. Extensions to `SHOW` Statements

Some extensions to `SHOW` statements accompany the implementation of `INFORMATION_SCHEMA`:

- `SHOW` can be used to get information about the structure of `INFORMATION_SCHEMA` itself.
- Several `SHOW` statements accept a `WHERE` clause that provides more flexibility in specifying which rows to display.

`INFORMATION_SCHEMA` is an information database, so its name is included in the output from `SHOW DATABASES`. Similarly, `SHOW TABLES` can be used with `INFORMATION_SCHEMA` to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
```

```
+-----+
| Tables_in_INFORMATION_SCHEMA |
+-----+
| CHARACTER_SETS              |
| COLLATIONS                   |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                     |
| COLUMN_PRIVILEGES            |
| ENGINES                      |
| EVENTS                       |
| FILES                        |
| GLOBAL_STATUS                |
| GLOBAL_VARIABLES             |
| KEY_COLUMN_USAGE             |
| PARTITIONS                   |
| PLUGINS                      |
| PROCESSLIST                  |
+-----+
```

```

| REFERENTIAL_CONSTRAINTS
| ROUTINES
| SCHEMATA
| SCHEMA_PRIVILEGES
| SESSION_STATUS
| SESSION_VARIABLES
| STATISTICS
| TABLES
| TABLE_CONSTRAINTS
| TABLE_PRIVILEGES
| TRIGGERS
| USER_PRIVILEGES
| VIEWS
+-----+
27 rows in set (0.00 sec)

```

`SHOW COLUMNS` and `DESCRIBE` can display information about the columns in individual `INFORMATION_SCHEMA` tables.

`SHOW` statements that accept a `LIKE [1111]` clause to limit the rows displayed also permit a `WHERE` clause that specifies more general conditions that selected rows must satisfy:

```

SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES

```

The `WHERE` clause, if present, is evaluated against the column names displayed by the `SHOW` statement. For example, the `SHOW CHARACTER SET` statement produces these output columns:

```

mysql> SHOW CHARACTER SET;
+-----+-----+-----+-----+
| Charset | Description                | Default collation | Maxlen |
+-----+-----+-----+-----+
| big5    | Big5 Traditional Chinese   | big5_chinese_ci  | 2      |
| dec8    | DEC West European         | dec8_swedish_ci  | 1      |
| cp850   | DOS West European         | cp850_general_ci | 1      |
| hp8     | HP West European          | hp8_english_ci   | 1      |
| koi8r   | KOI8-R Relcom Russian     | koi8r_general_ci | 1      |
| latin1  | cp1252 West European      | latin1_swedish_ci | 1      |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1      |
...

```

To use a `WHERE` clause with `SHOW CHARACTER SET`, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string `'japanese'`:

```

mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
+-----+-----+-----+-----+
| Charset | Description                | Default collation | Maxlen |
+-----+-----+-----+-----+
| ujis    | EUC-JP Japanese           | ujis_japanese_ci  | 3      |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci  | 2      |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci | 2      |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci | 3      |
+-----+-----+-----+-----+

```

This statement displays the multi-byte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
euckr	EUC-KR Korean	euckr_korean_ci	2
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

Chapter 21. MySQL Performance Schema

Table of Contents

21.1. Performance Schema Quick Start	2520
21.2. Performance Schema Configuration	2526
21.2.1. Performance Schema Build Configuration	2526
21.2.2. Performance Schema Startup Configuration	2527
21.2.3. Performance Schema Runtime Configuration	2528
21.3. Performance Schema Queries	2535
21.4. Performance Schema Instrument Naming Conventions	2535
21.5. Performance Schema Status Monitoring	2537
21.6. Performance Schema General Table Characteristics	2540
21.7. Performance Schema Table Descriptions	2540
21.7.1. Performance Schema Setup Tables	2541
21.7.2. Performance Schema Instance Tables	2543
21.7.3. Performance Schema Wait Event Tables	2546
21.7.4. Performance Schema Summary Tables	2548
21.7.5. Performance Schema Miscellaneous Tables	2551
21.8. Performance Schema System Variables	2552
21.9. Performance Schema Status Variables	2559
21.10. Performance Schema and Plugins	2560
21.11. Using the Performance Schema to Diagnose Problems	2560

The MySQL Performance Schema is a feature for monitoring MySQL Server execution at a low level. The Performance Schema is available as of MySQL 5.5.3 and has these characteristics:

- The Performance Schema provides a way to inspect internal execution of the server at runtime. It is implemented using the [PERFORMANCE_SCHEMA](#) storage engine and the [performance_schema](#) database. The Performance Schema focuses primarily on performance data. This differs from [INFORMATION_SCHEMA](#), which serves for inspection of metadata.
- The Performance Schema monitors server events. An “event” is anything the server does that takes time and has been instrumented so that timing information can be collected. In general, an event could be a function call, a wait for the operating system, a stage of an SQL statement execution such as parsing or sorting, or an entire statement or group of statements. Currently, event collection provides access to information about synchronization calls (such as for mutexes) and file I/O calls for the server and for several storage engines.
- Performance Schema events are distinct from events written to the server's binary log (which describe data modifications) and Event Scheduler events (which are a type of stored program).
- Current events are available, as well as event histories and summaries. This enables you to determine how many times instrumented activities were performed and how much time they took. Event information is available to show the activities of specific threads, or activity associated with particular objects such as a mutex or file.
- The [PERFORMANCE_SCHEMA](#) storage engine collects event data using “instrumentation points” in server source code.
- Collected events are stored in tables in the [performance_schema](#) database. These tables can be queried using [SELECT](#) statements like other tables.

- Performance Schema configuration can be modified dynamically by updating tables in the `performance_schema` database through SQL statements. Configuration changes affect data collection immediately.
- Tables in the `performance_schema` database are views or temporary tables that use no persistent on-disk storage.
- Monitoring is available on all platforms supported by MySQL.

Some limitations might apply: The types of timers might vary per platform. Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer. See also [Section E.8, “Restrictions on Performance Schema”](#).

- Data collection is implemented by modifying the server source code to add instrumentation. There are no separate threads associated with the Performance Schema, unlike other features such as replication or the Event Scheduler.

The Performance Schema is intended to provide access to useful information about server execution while having minimal impact on server performance. The implementation follows these design goals:

- Activating the Performance Schema causes no changes in server behavior. For example, it does not cause thread scheduling to change, and it does not cause query execution plans (as shown by `EXPLAIN`) to change.
- No memory allocation is done beyond that which occurs during server startup. By using early allocation of structures with a fixed size, it is never necessary to resize or reallocate them, which is critical for achieving good runtime performance.
- Server monitoring occurs continuously and unobtrusively with very little overhead. Activating the Performance Schema does not make the server unusable.
- The parser is unchanged. There are no new keywords or statements.
- Execution of server code proceeds normally even if the Performance Schema fails internally.
- When there is a choice between performing processing during event collection initially or during event retrieval later, priority is given to making collection faster. This is because collection is ongoing whereas retrieval is on demand and might never happen at all.
- It is easy to add new instrumentation points.
- Instrumentation is versioned. If the instrumentation implementation changes, previously instrumented code will continue to work. This benefits developers of third-party plugins because it is not necessary to upgrade each plugin to stay synchronized with the latest Performance Schema changes.

21.1. Performance Schema Quick Start

This section briefly introduces the Performance Schema with examples that show how to use it. For additional examples, see [Section 21.11, “Using the Performance Schema to Diagnose Problems”](#).

For the Performance Schema to be available, support for it must have been configured when MySQL was built. You can verify whether this is the case by checking the server's help output. If the Performance Schema is available, the output will mention several variables with names that begin with `performance_schema`:

```
shell> mysqld --verbose --help
...
```

```
--performance_schema
    Enable the performance schema.
--performance_schema_events_waits_history_long_size=#
    Number of rows in events_waits_history_long.
...
```

If such variables do not appear in the output, your server has not been built to support the Performance Schema. In this case, see [Section 21.2, “Performance Schema Configuration”](#).

Assuming that the Performance Schema is available, it is disabled by default. To enable it, start the server with the `performance_schema` [\[2554\]](#) variable enabled. For example, use these lines in your `my.cnf` file:

```
[mysqld]
performance_schema
```

When the server starts, it sees `performance_schema` [\[2554\]](#) and attempts to initialize the Performance Schema. To verify successful initialization, use this statement:

```
mysql> SHOW VARIABLES LIKE 'performance_schema';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| performance_schema | ON    |
+-----+-----+
```

A value of `ON` means that the Performance Schema initialized successfully and is ready for use. A value of `OFF` means that some error occurred. Check the server error log for information about what went wrong.

The Performance Schema is implemented as a storage engine. If this engine is available (which you should already have checked earlier), you should see it listed with a `SUPPORT` value of `YES` in the output from the `INFORMATION_SCHEMA.ENGINES` table or the `SHOW ENGINES` statement:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ENGINES
-> WHERE ENGINE='PERFORMANCE_SCHEMA'\G
***** 1. row *****
ENGINE: PERFORMANCE_SCHEMA
SUPPORT: YES
COMMENT: Performance Schema
TRANSACTIONS: NO
XA: NO
SAVEPOINTS: NO

mysql> SHOW ENGINES\G
...
Engine: PERFORMANCE_SCHEMA
Support: YES
Comment: Performance Schema
Transactions: NO
XA: NO
Savepoints: NO
...
```

The `PERFORMANCE_SCHEMA` storage engine operates on tables in the `performance_schema` database. You can make `performance_schema` the default database so that references to its tables need not be qualified with the database name:

```
mysql> USE performance_schema;
```

Many examples in this chapter assume that `performance_schema` is the default database.

Performance Schema tables are stored in the `performance_schema` database. Information about the structure of this database and its tables can be obtained, as for any other database, by selecting from

the `INFORMATION_SCHEMA` database or by using `SHOW` statements. For example, use either of these statements to see what Performance Schema tables exist:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
-> WHERE TABLE_SCHEMA = 'performance_schema';
+-----+
| TABLE_NAME |
+-----+
| cond_instances |
| events_waits_current |
| events_waits_history |
| events_waits_history_long |
| events_waits_summary_by_instance |
| events_waits_summary_by_thread_by_event_name |
| events_waits_summary_global_by_event_name |
| file_instances |
| file_summary_by_event_name |
| file_summary_by_instance |
| mutex_instances |
| performance_timers |
| rwlock_instances |
| setup_consumers |
| setup_instruments |
| setup_timers |
| threads |
+-----+

mysql> SHOW TABLES FROM performance_schema;
+-----+
| Tables_in_performance_schema |
+-----+
| cond_instances |
| events_waits_current |
| events_waits_history |
| ... |
+-----+
```

The number of Performance Schema tables is expected to increase over time as implementation of additional instrumentation proceeds.

The name of the `performance_schema` database is lowercase, as are the names of tables within it. Queries should specify the names in lowercase.



Note

Before MySQL 5.5.8, the table names were uppercase, which caused problems on some systems for certain values of the `lower_case_table_names` [518] system variable.

To see the structure of individual tables, use `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE setup_timers\G
***** 1. row *****
      Table: setup_timers
Create Table: CREATE TABLE `setup_timers` (
  `NAME` varchar(64) NOT NULL,
  `TIMER_NAME` enum('CYCLE','NANOSECOND','MICROSECOND','MILLISECOND','TICK')
  NOT NULL
) ENGINE=PERFORMANCE_SCHEMA DEFAULT CHARSET=utf8
```

Table structure is also available by selecting from tables such as `INFORMATION_SCHEMA.COLUMNS` or by using statements such as `SHOW COLUMNS`.

Tables in the `performance_schema` database can be grouped according to the type of information in them: Current events, event histories and summaries, object instances, and setup (configuration)

information. The following examples illustrate a few uses for these tables. For detailed information about the tables in each group, see [Section 21.7, “Performance Schema Table Descriptions”](#).

To see what the server is doing at the moment, examine the `events_waits_current` table. It contains one row per thread showing each thread's most recent monitored event:

```
mysql> SELECT * FROM events_waits_current\G
***** 1. row *****
      THREAD_ID: 0
      EVENT_ID: 5523
      EVENT_NAME: wait/synch/mutex/mysys/THR_LOCK::mutex
      SOURCE: thr_lock.c:525
      TIMER_START: 201660494489586
      TIMER_END: 201660494576112
      TIMER_WAIT: 86526
      SPINS: NULL
      OBJECT_SCHEMA: NULL
      OBJECT_NAME: NULL
      OBJECT_TYPE: NULL
      OBJECT_INSTANCE_BEGIN: 142270668
      NESTING_EVENT_ID: NULL
      OPERATION: lock
      NUMBER_OF_BYTES: NULL
      FLAGS: 0
      ...
```

This event indicates that thread 0 was waiting for 86,526 picoseconds to acquire a lock on `THR_LOCK::mutex`, a mutex in the `mysys` subsystem. The first few columns provide the following information:

- The ID columns indicate which thread the event comes from and the event number.
- `EVENT_NAME` indicates what was instrumented and `SOURCE` indicates which source file contains the instrumented code.
- The timer columns show when the event started and stopped and how long it took. If an event is still in progress, the `TIMER_END` and `TIMER_WAIT` values are `NULL`. Timer values are approximate and expressed in picoseconds. For information about timers and event time collection, see [Section 21.2.3.1, “Performance Schema Event Timing”](#).

The history tables contain the same kind of rows as the current-events table but have more rows and show what the server has been doing “recently” rather than “currently.” The `events_waits_history` and `events_waits_history_long` tables contain the most recent 10 events per thread and most recent 10,000 events, respectively. For example, to see information for recent events produced by thread 13, do this:

```
mysql> SELECT EVENT_ID, EVENT_NAME, TIMER_WAIT
-> FROM events_waits_history WHERE THREAD_ID = 13
-> ORDER BY EVENT_ID;
```

EVENT_ID	EVENT_NAME	TIMER_WAIT
86	wait/synch/mutex/mysys/THR_LOCK::mutex	686322
87	wait/synch/mutex/mysys/THR_LOCK_malloc	320535
88	wait/synch/mutex/mysys/THR_LOCK_malloc	339390
89	wait/synch/mutex/mysys/THR_LOCK_malloc	377100
90	wait/synch/mutex/sql/LOCK_plugin	614673
91	wait/synch/mutex/sql/LOCK_open	659925
92	wait/synch/mutex/sql/THD::LOCK_thd_data	494001
93	wait/synch/mutex/mysys/THR_LOCK_malloc	222489
94	wait/synch/mutex/mysys/THR_LOCK_malloc	214947
95	wait/synch/mutex/mysys/LOCK_alarm	312993

As new events are added to a history table, older events are discarded if the table is full.

Summary tables provide aggregated information for all events over time. The tables in this group summarize event data in different ways. To see which instruments have been executed the most times or have taken the most wait time, sort the `events_waits_summary_global_by_event_name` table on the `COUNT_STAR` or `SUM_TIMER_WAIT` column, which correspond to a `COUNT(*)` or `SUM(TIMER_WAIT)` value, respectively, calculated over all events:

```
mysql> SELECT EVENT_NAME, COUNT_STAR
-> FROM events_waits_summary_global_by_event_name
-> ORDER BY COUNT_STAR DESC LIMIT 10;
```

EVENT_NAME	COUNT_STAR
wait/synch/mutex/mysys/THR_LOCK_malloc	6419
wait/io/file/sql/FRM	452
wait/synch/mutex/sql/LOCK_plugin	337
wait/synch/mutex/mysys/THR_LOCK_open	187
wait/synch/mutex/mysys/LOCK_alarm	147
wait/synch/mutex/sql/THD::LOCK_thd_data	115
wait/io/file/myisam/kfile	102
wait/synch/mutex/sql/LOCK_global_system_variables	89
wait/synch/mutex/mysys/THR_LOCK::mutex	89
wait/synch/mutex/sql/LOCK_open	88

```
mysql> SELECT EVENT_NAME, SUM_TIMER_WAIT
-> FROM events_waits_summary_global_by_event_name
-> ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;
```

EVENT_NAME	SUM_TIMER_WAIT
wait/io/file/sql/MYSQL_LOG	1599816582
wait/synch/mutex/mysys/THR_LOCK_malloc	1530083250
wait/io/file/sql/binlog_index	1385291934
wait/io/file/sql/FRM	1292823243
wait/io/file/myisam/kfile	411193611
wait/io/file/myisam/dfile	322401645
wait/synch/mutex/mysys/LOCK_alarm	145126935
wait/io/file/sql/casetest	104324715
wait/synch/mutex/sql/LOCK_plugin	86027823
wait/io/file/sql/pid	72591750

These results show that the `THR_LOCK_malloc` mutex is “hot,” both in terms of how often it is used and amount of time that threads wait attempting to acquire it.



Note

The `THR_LOCK_malloc` mutex is used only in debug builds. In production builds it is not hot because it is nonexistent.

Instance tables document what types of objects are instrumented. An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information. For example, the `file_instances` table lists instances of instruments for file I/O operations and their associated files:

```
mysql> SELECT * FROM file_instances\G
***** 1. row *****
FILE_NAME: /opt/mysql-log/60500/binlog.000007
EVENT_NAME: wait/io/file/sql/binlog
OPEN_COUNT: 0
***** 2. row *****
```

```

FILE_NAME: /opt/mysql/60500/data/mysql/tables_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
***** 3. row *****
FILE_NAME: /opt/mysql/60500/data/mysql/columns_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
...

```

Setup tables are used to configure and display monitoring characteristics. For example, to see which event timer is selected, query the `setup_timers` tables:

```

mysql> SELECT * FROM setup_timers;
+-----+-----+
| NAME | TIMER_NAME |
+-----+-----+
| wait | CYCLE      |
+-----+-----+

```

`setup_instruments` lists the set of instruments for which events can be collected and shows which of them are enabled:

```

mysql> SELECT * FROM setup_instruments;
+-----+-----+-----+-----+
| NAME                                     | ENABLED | TIMED |
+-----+-----+-----+-----+
...
| wait/synch/mutex/sql/LOCK_global_read_lock | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave | YES     | YES   |
...
| wait/io/file/sql/binlog | YES     | YES   |
| wait/io/file/sql/binlog_index | YES     | YES   |
| wait/io/file/sql/casetest | YES     | YES   |
| wait/io/file/sql/dbopt | YES     | YES   |
...

```

To understand how to interpret instrument names, see [Section 21.4, “Performance Schema Instrument Naming Conventions”](#).

To control whether events are collected for an instrument, set its `ENABLED` value to `YES` or `NO`. For example:

```

mysql> UPDATE setup_instruments SET ENABLED = 'NO'
-> WHERE NAME = 'wait/synch/mutex/sql/LOCK_mysql_create_db';

```

The Performance Schema uses collected events to update tables in the `performance_schema` database, which act as “consumers” of event information. The `setup_consumers` table lists the available consumers and shows which of them are enabled:

```

mysql> SELECT * FROM setup_consumers;
+-----+-----+
| NAME                                     | ENABLED |
+-----+-----+
| events_waits_current | YES     |
| events_waits_history | YES     |
| events_waits_history_long | YES     |
| events_waits_summary_by_thread_by_event_name | YES     |

```

events_waits_summary_by_event_name	YES
events_waits_summary_by_instance	YES
file_summary_by_event_name	YES
file_summary_by_instance	YES

To control whether the Performance Schema maintains a consumer as a destination for event information, set its `ENABLED` value.

For more information about the setup tables and how to use them to control event collection, see [Section 21.2.3.2, “Performance Schema Event Filtering”](#).

There are some miscellaneous tables that do not fall into any of the previous groups. For example, `performance_timers` lists the available event timers and their characteristics. For information about timers, see [Section 21.2.3.1, “Performance Schema Event Timing”](#).

21.2. Performance Schema Configuration

To use the MySQL Performance Schema, these configuration considerations apply:

- The Performance Schema must be configured into MySQL Server at build time to make it available. Performance Schema support is included in binary MySQL distributions. If you are building from source, you must ensure that it is configured into the build as described in [Section 21.2.1, “Performance Schema Build Configuration”](#).
- The Performance Schema must be enabled at server startup to enable event collection to occur. Specific Performance Schema features can be enabled at server startup or at runtime to control which types of event collection occur. See [Section 21.2.2, “Performance Schema Startup Configuration”](#), [Section 21.2.3, “Performance Schema Runtime Configuration”](#), and [Section 21.2.3.2, “Performance Schema Event Filtering”](#).

21.2.1. Performance Schema Build Configuration

For the Performance Schema to be available, it must be configured into the MySQL server at build time. Binary MySQL distributions provided by Oracle Corporation are configured to support the Performance Schema. If you use a binary MySQL distribution from another provider, check with the provider whether the distribution has been appropriately configured.

If you build MySQL from a source distribution, enable the Performance Schema by running `CMake` with the `WITH_PERFSCHEMA_STORAGE_ENGINE` option enabled:

```
shell> cmake . -DWITH_PERFSCHEMA_STORAGE_ENGINE=1
```

Configuring MySQL with the `-DWITHOUT_PERFSCHEMA_STORAGE_ENGINE=1` option prevents inclusion of the Performance Schema, so if you want it included, do not use this option. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

If you install MySQL over a previous installation that was configured without the Performance Schema (or with an older version of the Performance Schema that may not have all the current tables), run `mysql_upgrade` after starting the server to ensure that the `performance_schema` database exists with all current tables. Then restart the server. One indication that you need to do this is the presence of messages such as the following in the error log:

```
[ERROR] Native table 'performance_schema`.`events_waits_history'
has the wrong structure
[ERROR] Native table 'performance_schema`.`events_waits_history_long'
has the wrong structure
```

...

To verify whether a server was built with Performance Schema support, check its help output. If the Performance Schema is available, the output will mention several variables with names that begin with `performance_schema`:

```
shell> mysqld --verbose --help
...
--performance_schema
           Enable the performance schema.
--performance_schema_events_waits_history_long_size=#
           Number of rows in events_waits_history_long.
...
```

You can also connect to the server and look for a line that names the `PERFORMANCE_SCHEMA` storage engine in the output from `SHOW ENGINES`:

```
mysql> SHOW ENGINES\G
...
  Engine: PERFORMANCE_SCHEMA
  Support: YES
  Comment: Performance Schema
  Transactions: NO
    XA: NO
  Savepoints: NO
...
```

If the Performance Schema was not configured into the server at build time, no row for `PERFORMANCE_SCHEMA` will appear in the output from `SHOW ENGINES`. You might see `performance_schema` listed in the output from `SHOW DATABASES`, but it will have no tables and you will not be able to use it.

A line for `PERFORMANCE_SCHEMA` in the `SHOW ENGINES` output means that the Performance Schema is available, not that it is enabled. To enable it, you must do so at server startup, as described in the next section.

21.2.2. Performance Schema Startup Configuration

The Performance Schema is disabled by default. To enable it, start the server with the `performance_schema` [2554] variable enabled. For example, use these lines in your `my.cnf` file:

```
[mysqld]
performance_schema
```

If the server is unable to allocate any internal buffer during Performance Schema initialization, the Performance Schema disables itself and sets `performance_schema` [2554] to `OFF`, and the server runs without instrumentation.

The Performance Schema includes several system variables that provide configuration information:

```
mysql> SHOW VARIABLES LIKE 'perf%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| performance_schema | ON |
| performance_schema_events_waits_history_long_size | 10000 |
| performance_schema_events_waits_history_size | 10 |
| performance_schema_max_cond_classes | 80 |
| performance_schema_max_cond_instances | 1000 |
| performance_schema_max_file_classes | 50 |
| performance_schema_max_file_handles | 32768 |
| performance_schema_max_file_instances | 10000 |
+-----+-----+
```

performance_schema_max_mutex_classes	200
performance_schema_max_mutex_instances	1000000
performance_schema_max_rwlock_classes	30
performance_schema_max_rwlock_instances	1000000
performance_schema_max_table_handles	100000
performance_schema_max_table_instances	50000
performance_schema_max_thread_classes	50
performance_schema_max_thread_instances	1000

The `performance_schema` [2554] variable is `ON` or `OFF` to indicate whether the Performance Schema is enabled or disabled. The other variables indicate table sizes (number of rows) or memory allocation values.



Note

With the Performance Schema enabled, the number of Performance Schema instances affects the server memory footprint, perhaps to a large extent. It may be necessary to tune the values of Performance Schema system variables to find the number of instances that balances insufficient instrumentation against excessive memory consumption.

To change the value of Performance Schema system variables, set them at server startup. For example, put the following lines in a `my.cnf` file to change the sizes of the history tables:

```
[mysqld]
performance_schema
performance_schema_events_waits_history_size=20
performance_schema_events_waits_history_long_size=15000
```

21.2.3. Performance Schema Runtime Configuration

Performance Schema setup tables contain information about monitoring configuration:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
-> WHERE TABLE_SCHEMA = 'performance_schema'
-> AND TABLE_NAME LIKE 'setup%';
+-----+
| TABLE_NAME |
+-----+
| setup_consumers |
| setup_instruments |
| setup_timers |
+-----+
```

You can examine the contents of these tables to obtain information about Performance Schema monitoring characteristics. If you have the `UPDATE` [697] privilege, you can change Performance Schema operation by modifying setup tables to affect how monitoring occurs. For additional details about these tables, see [Section 21.7.1, “Performance Schema Setup Tables”](#).

To see which event timer is selected, query the `setup_timers` tables:

```
mysql> SELECT * FROM setup_timers;
+-----+-----+
| NAME | TIMER_NAME |
+-----+-----+
| wait | CYCLE |
+-----+-----+
```

The `NAME` value indicates the type of instrument to which the timer applies, and `TIMER_NAME` indicates which timer applies to those instruments. The timer applies to instruments where their name begins with

a component matching the `NAME` value. Currently, there are only “wait” instruments, so this table has only one row and the timer applies to all instruments.

To change the timer, update the `NAME` value. For example, to use the `NANOSECOND` timer:

```
mysql> UPDATE setup_timers SET TIMER_NAME = 'NANOSECOND'
      -> WHERE NAME = 'wait';

mysql> SELECT * FROM setup_timers;
+-----+-----+
| NAME | TIMER_NAME |
+-----+-----+
| wait | NANOSECOND |
+-----+-----+
```

For discussion of timers, see [Section 21.2.3.1, “Performance Schema Event Timing”](#).

The `setup_instruments` and `setup_consumers` tables list the instruments for which events can be collected and the types of consumers for which event information actually is collected, respectively. [Section 21.2.3.2, “Performance Schema Event Filtering”](#), discusses how you can modify these tables to affect event collection.

If there are Performance Schema configuration changes that must be made at runtime using SQL statements and you would like to these changes take effect each time the server starts, put the statements in a file and start the server with the `--init-file=file_name` [445] option. This strategy can also be useful if you have multiple monitoring configurations, each tailored to produce a different kind of monitoring, such as casual server health monitoring, incident investigation, application behavior troubleshooting, and so forth. Put the statements for each monitoring configuration into their own file and specify the appropriate file as the `--init-file` [445] argument when you start the server.

21.2.3.1. Performance Schema Event Timing

Events are collected by means of instrumentation added to the server source code. Instruments time events, which is how the Performance Schema provides an idea of how long events take. It is also possible to configure instruments not to collect timing information. This section discusses the available timers and their characteristics, and how timing values are represented in events.

Two tables provide timer information:

- `performance_timers` lists the available timers and their characteristics.
- `setup_timers` indicates which timers are used for which instruments.

Each timer row in `setup_timers` must refer to one of the timers listed in `performance_timers`.

Timers vary in precision and the amount of overhead they involve. To see what timers are available and their characteristics, check the `performance_timers` table:

```
mysql> SELECT * FROM performance_timers;
+-----+-----+-----+-----+
| TIMER_NAME | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-----+-----+-----+-----+
| CYCLE | 2389029850 | 1 | 72 |
| NANOSECOND | NULL | NULL | NULL |
| MICROSECOND | 1000000 | 1 | 585 |
| MILLISECOND | 1035 | 1 | 738 |
| TICK | 101 | 1 | 630 |
+-----+-----+-----+-----+
```

The `TIMER_NAME` column shows the names of the available timers. `CYCLE` refers to the timer that is based on the CPU (processor) cycle counter. If the values associated with a given timer name are `NULL`, that

timer is not supported on your platform. The rows that do not have `NULL` indicate which timers you can use in `setup_timers`.

`TIMER_FREQUENCY` indicates the number of timer units per second. For a cycle timer, the frequency is generally related to the CPU speed. The value shown was obtained on a system with a 2.4GHz processor. The other timers are based on fixed fractions of seconds. For `TICK`, the frequency may vary by platform (for example, some use 100 ticks/second, others 1000 ticks/second).

`TIMER_RESOLUTION` indicates the number of timer units by which timer values increase at a time. If a timer has a resolution of 10, its value increases by 10 each time.

`TIMER_OVERHEAD` is the minimal number of cycles of overhead to obtain one timing with the given timer. The overhead per event is twice the value displayed because the timer is invoked at the beginning and end of the event.

To see which timer is in effect or to change the timer, access the `setup_timers` table:

```
mysql> SELECT * FROM setup_timers;
+-----+-----+
| NAME | TIMER_NAME |
+-----+-----+
| wait | CYCLE      |
+-----+-----+

mysql> UPDATE setup_timers SET TIMER_NAME = 'MICROSECOND'
-> WHERE NAME = 'wait';

mysql> SELECT * FROM setup_timers;
+-----+-----+
| NAME | TIMER_NAME |
+-----+-----+
| wait | MICROSECOND |
+-----+-----+
```

By default, the Performance Schema uses the best timer available for each instrument type, but you can select a different one. Generally the best timer is `CYCLE`, which uses the CPU cycle counter whenever possible to provide high precision and low overhead.

The precision offered by the cycle counter depends on processor speed. If the processor runs at 1 GHz (one billion cycles/second) or higher, the cycle counter delivers sub-nanosecond precision. Using the cycle counter is much cheaper than getting the actual time of day. For example, the standard `gettimeofday()` function can take hundreds of cycles, which is an unacceptable overhead for data gathering that may occur thousands or millions of times per second.

Cycle counters also have disadvantages:

- End users expect to see timings in wall-clock units, such as fractions of a second. Converting from cycles to fractions of seconds can be expensive. For this reason, the conversion is a quick and fairly rough multiplication operation.
- Processor cycle rate might change, such as when a laptop goes into power-saving mode or when a CPU slows down to reduce heat generation. If a processor's cycle rate fluctuates, conversion from cycles to real-time units is subject to error.
- Cycle counters might be unreliable or unavailable depending on the processor or the operating system. For example, on Pentiums, the instruction is `RDTSC` (an assembly-language rather than a C instruction) and it is theoretically possible for the operating system to prevent user-mode programs from using it.
- Some processor details related to out-of-order execution or multiprocessor synchronization might cause the counter to seem fast or slow by up to 1000 cycles.

Currently, MySQL works with cycle counters on x386 (Windows, Mac OS X, Linux, Solaris, and other Unix flavors), PowerPC, and IA-64.

The `setup_instruments` table has an `ENABLED` column to indicate the instruments for which to collect events. The table also has a `TIMED` column to indicate which instruments are timed. If an instrument is not enabled, it produces no events. If an enabled instrument is not timed, events produced by the instrument have `NULL` for the `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` timer values. This in turn causes those values to be ignored when calculating the sum, minimum, maximum, and average time values in summary tables.

Within events, times are stored in picoseconds (trillionths of a second) to normalize them to a standard unit, regardless of which timer is selected. The timer used for an event is the one in effect when event timing begins. This timer is used to convert start and end values to picoseconds for storage in the event.

Modifications to the `setup_timers` table affect monitoring immediately. Events already in progress use the original timer for the begin time and the new timer for the end time, which leads to unpredictable results. If you make timer changes, you may want to use `TRUNCATE TABLE` to reset Performance Schema statistics.

The timer baseline (“time zero”) occurs at Performance Schema initialization during server startup. `TIMER_START` and `TIMER_END` values in events represent picoseconds since the baseline. `TIMER_WAIT` values are durations in picoseconds.

Picosecond values in events are approximate. Their accuracy is subject to the usual forms of error associated with conversion from one unit to another. If the `CYCLE` timer is used and the processor rate varies, there might be drift. For these reasons, it is not reasonable to look at the `TIMER_START` value for an event as an accurate measure of time elapsed since server startup. On the other hand, it is reasonable to use `TIMER_START` or `TIMER_WAIT` values in `ORDER BY` clauses to order events by start time or duration.

The choice of picoseconds in events rather than a value such as microseconds has a performance basis. One implementation goal was to show results in a uniform time unit, regardless of the timer. In an ideal world this time unit would look like a wall-clock unit and be reasonably precise; in other words, microseconds. But to convert cycles or nanoseconds to microseconds, it would be necessary to perform a division for every instrumentation. Division is expensive on many platforms. Multiplication is not expensive, so that is what is used. Therefore, the time unit is an integer multiple of the highest possible `TIMER_FREQUENCY` value, using a multiplier large enough to ensure that there is no major precision loss. The result is that the time unit is “picoseconds.” This precision is spurious, but the decision enables overhead to be minimized.

21.2.3.2. Performance Schema Event Filtering

Events are processed in a producer/consumer fashion:

- Instrumented code is the source for events and produces events to be collected. The `setup_instruments` table lists the instruments for which events can be collected, whether they are enabled, and whether to collect timing information:

```
mysql> SELECT * FROM setup_instruments;
+-----+-----+-----+
| NAME                                     | ENABLED | TIMED |
+-----+-----+-----+
...
| wait/synch/mutex/sql/LOCK_global_read_lock | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager         | YES     | YES   |
+-----+-----+-----+
```

```

...
| wait/synch/rwlock/sql/LOCK_grant           | YES   | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger  | YES   | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect | YES   | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave  | YES   | YES   |
...
| wait/io/file/sql/binlog                    | YES   | YES   |
| wait/io/file/sql/binlog_index              | YES   | YES   |
| wait/io/file/sql/casetest                  | YES   | YES   |
| wait/io/file/sql/dbopt                     | YES   | YES   |
...

```

- Performance Schema tables are the destinations for events and consume events. The `setup_consumers` table lists the types of consumers to which event information can be sent:

```

mysql> SELECT * FROM setup_consumers;
+-----+-----+
| NAME                                     | ENABLED |
+-----+-----+
| events_waits_current                    | YES     |
| events_waits_history                     | YES     |
| events_waits_history_long                | YES     |
| events_waits_summary_by_thread_by_event_name | YES     |
| events_waits_summary_by_event_name       | YES     |
| events_waits_summary_by_instance         | YES     |
| file_summary_by_event_name               | YES     |
| file_summary_by_instance                 | YES     |
+-----+-----+

```

Filtering can be done at different stages of performance monitoring:

- **Pre-filtering.** This is done by modifying Performance Schema configuration so that only certain types of events are collected from producers, and collected events update only certain consumers. This type of filtering is done by the Performance Schema and has a global effect that applies to all users.

Reasons to use pre-filtering:

- Pre-filtering reduces overhead. The overhead should be minimal even with all instruments enabled, but perhaps you want to reduce it further. Or you do not care about timing events and want to disable the timing code to eliminate timing overhead.
- You can avoid filling the current-events or history tables with events in which you have no interest. Pre-filtering leaves more “room” in these tables for instances of rows for enabled instrument types. If you enable only file instruments with pre-filtering, no rows are collected for nonfile instruments. With post-filtering, nonfile events are collected, leaving fewer rows for file events.
- You can avoid maintaining some kinds of event tables. If you disable a consumer, the server does not spend time maintaining it. For example, if you do not care about event histories, you can disable the history table consumers to improve performance.
- **Post-filtering.** This involves the use of `WHERE` clauses in queries that select information from Performance Schema tables, to specify which of the available events you want to see. This type of filtering is performed on a per-user basis because individual users select which of the available events are of interest.

Reasons to use post-filtering:

- To avoid making decisions for individual users about which event information is of interest.
- To use the Performance Schema to investigate a performance issue when the restrictions to impose using pre-filtering are not known in advance.

The following sections provide more detail about pre-filtering and provide guidelines for naming instruments or consumers in filtering operations. For information about writing queries to retrieve information (post-filtering), see [Section 21.3, “Performance Schema Queries”](#).

21.2.3.2.1. Event Pre-Filtering

Pre-filtering is done by modifying Performance Schema configuration so that only certain types of events are collected from producers, and collected events update only certain consumers. This type of filtering is done by the Performance Schema and has a global effect that applies to all users.

Pre-filtering can be applied to either the producer or consumer stage of event processing:

- To affect pre-filtering at the producer stage, modify the `setup_instruments` table. An instrument can be enabled or disabled by setting its `ENABLED` value to `YES` or `NO`. An instrument can be configured whether to collect timing information by setting its `TIMED` value to `YES` or `NO`.
- To affect pre-filtering at the consumer stage, modify the `setup_consumers` table. A consumer can be enabled or disabled by setting its `ENABLED` value to `YES` or `NO`.

Here are some examples that show the types of pre-filtering operations available:

- Disable all instruments:

```
mysql> UPDATE setup_instruments SET ENABLED = 'NO';
```

Now no events will be collected. This change, like other pre-filtering operations, affects other users as well, even if they want to see event information.

- Disable all file instruments, adding them to the current set of disabled instruments:

```
mysql> UPDATE setup_instruments SET ENABLED = 'NO'
-> WHERE NAME LIKE 'wait/io/file/%';
```

- Disable only file instruments, enable all other instruments:

```
mysql> UPDATE setup_instruments
-> SET ENABLED = IF(NAME LIKE 'wait/io/file/%', 'NO', 'YES');
```

The preceding queries use the `LIKE [1111]` operator and the pattern `'wait/io/file/%'` to match all instrument names that begin with `'wait/io/file/`. For additional information about specifying patterns to select instruments, see [Section 21.2.3.2.2, “Naming Instruments or Consumers for Filtering Operations”](#).

- Enable all but those instruments in the `mysys` library:

```
mysql> UPDATE setup_instruments
-> SET ENABLED = CASE WHEN NAME LIKE '%/mysys/%' THEN 'YES' ELSE 'NO' END;
```

- Disable a specific instrument:

```
mysql> UPDATE setup_instruments SET ENABLED = 'NO'
-> WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- To toggle the state of an instrument, “flip” its `ENABLED` value:

```
mysql> UPDATE setup_instruments
-> SET ENABLED = IF(ENABLED = 'YES', 'NO', 'YES')
-> WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- Disable timing for all events:

```
mysql> UPDATE setup_instruments SET TIMED = 'NO';
```

Setting the `TIMED` column for instruments affects Performance Schema table contents as described in [Section 21.2.3.1, “Performance Schema Event Timing”](#).

When you change the monitoring configuration, the Performance Schema does not flush the history tables. Events already collected remain in the current-events and history tables until displaced by newer events. If you disable instruments, you might need to wait a while before events for them are displaced by newer events of interest. Alternatively, use `TRUNCATE TABLE` to empty the history tables.

After making instrumentation changes, you might want to truncate the summary tables as well to clear aggregate information for previously collected events. The effect of `TRUNCATE TABLE` for summary tables is to reset the summary columns to 0 or `NULL`, not to remove rows.

If you disable a consumer, the server does not spend time maintaining it. For example, you can disable the history table consumers if you do not care about historical event information:

```
mysql> UPDATE setup_consumers
-> SET ENABLED = 'NO' WHERE NAME LIKE '%history%';
```

21.2.3.2.2. Naming Instruments or Consumers for Filtering Operations

Names given for filtering operations can be as specific or general as required. To indicate a single instrument or consumer, specify its name in full:

```
mysql> UPDATE setup_instruments
-> SET ENABLED = 'NO'
-> WHERE NAME = 'wait/synch/mutex/myisammrg/MYRG_INFO::mutex';

mysql> UPDATE setup_consumers
-> SET ENABLED = 'NO' WHERE NAME = 'file_summary_by_instance';
```

To specify a group of instruments or consumers, use a pattern that matches the group members:

```
mysql> UPDATE setup_instruments
-> SET ENABLED = 'NO'
-> WHERE NAME LIKE 'wait/synch/mutex/%';

mysql> UPDATE setup_consumers
-> SET ENABLED = 'NO' WHERE NAME LIKE '%history%';
```

If you use a pattern, it should be chosen so that it matches all the items of interest and no others. For example, to select all file I/O instruments, it is better to use a pattern that includes the entire instrument name prefix:

```
... WHERE NAME LIKE 'wait/io/file/%';
```

A pattern of `'%/file/%'` will match other instruments that have a component of `'/file/'` anywhere in the name. Even less suitable is the pattern `'%file%'` because it will match instruments with `'file'` anywhere in the name, such as `wait/synch/mutex/sql/LOCK_des_key_file`.

To check which instrument or consumer names a pattern matches, perform a simple test:

```
mysql> SELECT NAME FROM setup_instruments WHERE NAME LIKE 'pattern';

mysql> SELECT NAME FROM setup_consumers WHERE NAME LIKE 'pattern';
```

For information about the types of names that are supported, see [Section 21.4, “Performance Schema Instrument Naming Conventions”](#).

21.2.3.3. Determining What Is Instrumented

It is always possible to determine what instruments the Performance Schema includes by checking the `setup_instruments` table. For example, to see what file-related events are instrumented for the `InnoDB` storage engine, use this query:

```
mysql> SELECT * FROM setup_instruments WHERE NAME LIKE 'wait/io/file/innodb/%';
```

NAME	ENABLED	TIMED
wait/io/file/innodb/innodb_data_file	YES	YES
wait/io/file/innodb/innodb_log_file	YES	YES
wait/io/file/innodb/innodb_temp_file	YES	YES

An exhaustive description of precisely what is instrumented is not given in this documentation, for several reasons:

- What is instrumented is the server code. Changes to this code occur often, which also affects the set of instruments.
- It is not practical to list all the instruments because there are hundreds of them.
- As described earlier, it is possible to find out by querying the `setup_instruments` table. This information is always up to date for your version of MySQL, also includes instrumentation for instrumented plugins you might have installed that are not part of the core server, and can be used by automated tools.

21.3. Performance Schema Queries

Pre-filtering limits which event information is collected and is independent of any particular user. By contrast, post-filtering is performed by individual users through the use of queries with appropriate `WHERE` clauses that restrict what event information to select from the events available after pre-filtering has been applied.

In [Section 21.2.3.2.1, “Event Pre-Filtering”](#), an example showed how to pre-filter for file instruments. If the event tables contain both file and nonfile information, post-filtering is another way to see information only for file events. Add a `WHERE` clause to queries to restrict event selection appropriately:

```
mysql> SELECT THREAD_ID, NUMBER_OF_BYTES
-> FROM events_waits_history
-> WHERE EVENT_NAME LIKE 'wait/io/file/%'
-> AND NUMBER_OF_BYTES IS NOT NULL;
```

THREAD_ID	NUMBER_OF_BYTES
11	66
11	47
11	139
5	24
5	834

21.4. Performance Schema Instrument Naming Conventions

An instrument name consists of a sequence of components separated by `'/'` characters. Example names:

```
wait/io/file/myisam/log
wait/io/file/mysys/charset
wait/synch/cond/mysys/COND_alarm
```

```
wait/synch/cond/sql/BINLOG::update_cond
wait/synch/mutex/mysys/BITMAP_mutex
wait/synch/mutex/sql/LOCK_delete
wait/synch/rwlock/innodb/trx_sys_lock
wait/synch/rwlock/sql/Query_cache_query::lock
```

The instrument name space has a tree-like structure. The components of an instrument name from left to right provide a progression from more general to more specific. The number of components a name has depends on the type of instrument.

The interpretation of a given component in a name depends on the components to the left of it. For example, `myisam` appears in both of the following names, but `myisam` in the first name is related to file I/O, whereas in the second it is related to a synchronization instrument:

```
wait/io/file/myisam/log
wait/synch/cond/myisam/MI_SORT_INFO::cond
```

Instrument names consist of a prefix with a structure defined by the Performance Schema implementation and a suffix defined by the developer implementing the instrument code. The top-level component of an instrument prefix indicates the type of instrument. This component also determines which event timer in the `setup_timers` table applies to the instrument. For the prefix part of instrument names, the top level indicates the type of instrument.

The suffix part of instrument names comes from the code for the instruments themselves. Suffixes may include levels such as these:

- A name for the major component (a server module such as `myisam`, `innodb`, `mysys`, or `sql`) or a plugin name.
- The name of a variable in the code, in the form `XXX` (a global variable) or `CCC:MMM` (a member `MMM` in class `CCC`). Examples: `COND_thread_cache`, `THR_LOCK_myisam`, `BINLOG::LOCK_index`.

In MySQL 5.5, there is a single top-level component, `wait`, indicating a wait instrument. The naming tree for wait instruments has this structure:

- `wait/io`

An instrumented I/O operation.

- `wait/io/file`

An instrumented file I/O operation. For files, the wait is the time waiting for the file operation to complete (for example, a call to `fwrite()`). Due to caching, the physical file I/O on the disk might not happen within this call.

- `wait/synch`

An instrumented synchronization object. For synchronization objects, the `TIMER_WAIT` time includes the amount of time blocked while attempting to acquire a lock on the object, if any.

- `wait/synch/cond`

A condition is used by one thread to signal to other threads that something they were waiting for has happened. If a single thread was waiting for a condition, it can wake up and proceed with its execution. If several threads were waiting, they can all wake up and compete for the resource for which they were waiting.

- `wait/synch/mutex`

A mutual exclusion object used to permit access to a resource (such as a section of executable code) while preventing other threads from accessing the resource.

- `wait/synch/rwlock`

A read/write lock object used to lock a specific variable for access while preventing its use by other threads. A shared read lock can be acquired simultaneously by multiple threads. An exclusive write lock can be acquired by only one thread at a time.

21.5. Performance Schema Status Monitoring

There are several status variables associated with the Performance Schema:

```
mysql> SHOW STATUS LIKE 'perf%';
+-----+-----+
| Variable_name                | Value |
+-----+-----+
| Performance_schema_cond_classes_lost | 0      |
| Performance_schema_cond_instances_lost | 0      |
| Performance_schema_file_classes_lost  | 0      |
| Performance_schema_file_handles_lost  | 0      |
| Performance_schema_file_instances_lost | 0      |
| Performance_schema_locker_lost        | 0      |
| Performance_schema_mutex_classes_lost  | 0      |
| Performance_schema_mutex_instances_lost | 0      |
| Performance_schema_rwlock_classes_lost | 0      |
| Performance_schema_rwlock_instances_lost | 0      |
| Performance_schema_table_handles_lost  | 0      |
| Performance_schema_table_instances_lost | 0      |
| Performance_schema_thread_classes_lost | 0      |
| Performance_schema_thread_instances_lost | 0      |
+-----+-----+
```

The Performance Schema status variables provide information about instrumentation that could not be loaded or created due to memory constraints. Names for these variables have several forms:

- `Performance_schema_xxx_classes_lost` indicates how many instruments of type `xxx` could not be loaded.
- `Performance_schema_xxx_instances_lost` indicates how many instances of object type `xxx` could not be created.
- `Performance_schema_xxx_handles_lost` indicates how many instances of object type `xxx` could not be opened.
- `Performance_schema_locker_lost` indicates how many events are “lost” or not recorded.

For example, if a mutex is instrumented in the server source but the server cannot allocate memory for the instrumentation at runtime, it increments `Performance_schema_mutex_classes_lost` [2559]. The mutex still functions as a synchronization object (that is, the server continues to function normally), but performance data for it will not be collected. If the instrument can be allocated, it can be used for initializing instrumented mutex instances. For a singleton mutex such as a global mutex, there will be only one instance. Other mutexes have an instance per connection, or per page in various caches and data buffers, so the number of instances varies over time. Increasing the maximum number of connections or the maximum size of some buffers will increase the maximum number of instances that might be allocated at once. If the server cannot create a given instrumented mutex instance, it increments `Performance_schema_mutex_instances_lost` [2559].

Suppose that the following conditions hold:

- The server was started with the `--performance_schema_max_mutex_classes=200` [2556] option and thus has room for 200 mutex instruments.
- 150 mutex instruments have been loaded already.
- The plugin named `plugin_a` contains 40 mutex instruments.
- The plugin named `plugin_b` contains 20 mutex instruments.

The server allocates mutex instruments for the plugins depending on how many they need and how many are available, as illustrated by the following sequence of statements:

```
INSTALL PLUGIN plugin_a
```

The server now has $150+40 = 190$ mutex instruments.

```
UNINSTALL PLUGIN plugin_a;
```

The server still has 190 instruments. All the historical data generated by the plugin code is still available, but new events for the instruments are not collected.

```
INSTALL PLUGIN plugin_a;
```

The server detects that the 40 instruments are already defined, so no new instruments are created, and previously assigned internal memory buffers are reused. The server still has 190 instruments.

```
INSTALL PLUGIN plugin_b;
```

The server has room for $200-190 = 10$ instruments (in this case, mutex classes), and sees that the plugin contains 20 new instruments. 10 instruments are loaded, and 10 are discarded or “lost.” The `Performance_schema_mutex_classes_lost` [2559] indicates the number of instruments (mutex classes) lost:

```
mysql> SHOW STATUS LIKE "perf%mutex_classes_lost";
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Performance_schema_mutex_classes_lost | 10    |
+-----+-----+
1 row in set (0.10 sec)
```

The instrumentation still works and collects (partial) data for `plugin_b`.

When the server cannot create a mutex instrument, these results occur:

- No row for the instrument is inserted into the `setup_instruments` table.
- `Performance_schema_mutex_classes_lost` [2559] increases by 1.
- `Performance_schema_mutex_instances_lost` [2559] does not change. (When the mutex instrument is not created, it cannot be used to create instrumented mutex instances later.)

The pattern just described applies to all types of instruments, not just mutexes.

A value of `Performance_schema_mutex_classes_lost` [2559] greater than 0 can happen in two cases:

- To save a few bytes of memory, you start the server with `--performance_schema_max_mutex_classes=N` [2556], where N is less than the default value. The default value is chosen to be sufficient to load all the plugins provided in the MySQL distribution, but this can be reduced if some plugins are never loaded. For example, you might choose not to load some of the storage engines in the distribution.

- You load a third-party plugin that is instrumented for the Performance Schema but do not allow for the plugin's instrumentation memory requirements when you start the server. Because it comes from a third party, the instrument memory consumption of this engine is not accounted for in the default value chosen for `performance_schema_max_mutex_classes` [2556].

If the server has insufficient resources for the plugin's instruments and you do not explicitly allocate more using `--performance_schema_max_mutex_classes=N` [2556], loading the plugin leads to starvation of instruments.

If the value chosen for `performance_schema_max_mutex_classes` [2556] is too small, no error is reported in the error log and there is no failure at runtime. However, the content of the tables in the `performance_schema` database will miss events. The `Performance_schema_mutex_classes_lost` [2559] status variable is the only visible sign to indicate that some events were dropped internally due to failure to create instruments.

If an instrument is not lost, it is known to the Performance Schema, and is used when instrumenting instances. For example, `wait/synch/mutex/sql/LOCK_delete` is the name of a mutex instrument in the `setup_instruments` table. This single instrument is used when creating in the code (in `THD::LOCK_delete`) however many instances of the mutex are needed as the server runs. In this case, `LOCK_delete` is a mutex that is per connection (`THD`), so if a server has 1000 connections, there are 1000 threads, and 1000 instrumented `LOCK_delete` mutex instances (`THD::LOCK_delete`).

If the server does not have room for all these 1000 instrumented mutexes (instances), some mutexes are created with instrumentation, and some are created without instrumentation. If the server can create only 800 instances, 200 instances are lost. The server continues to run, but increments `Performance_schema_mutex_instances_lost` [2559] by 200 to indicate that instances could not be created.

A value of `Performance_schema_mutex_instances_lost` [2559] greater than 0 can happen when the code initializes more mutexes at runtime than were allocated for `--performance_schema_max_mutex_instances=N` [2556].

The bottom line is that if `SHOW STATUS LIKE 'perf%'` says that nothing was lost (all values are zero), the Performance Schema data is accurate and can be relied upon. If something was lost, the data is incomplete, and the Performance Schema could not record everything given the insufficient amount of memory it was given to use. In this case, the specific `Performance_schema_xxx_lost` variable indicates the problem area.

It might be appropriate in some cases to cause deliberate instrument starvation. For example, if you do not care about performance data for file I/O, you can start the server with all Performance Schema parameters related to file I/O set to 0. No memory will be allocated for file-related classes, instances, or handles, and all file events will be lost.

Use `SHOW ENGINE PERFORMANCE_SCHEMA STATUS` to inspect the internal operation of the Performance Schema code:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
***** 3. row *****
  Type: performance_schema
  Name: events_waits_history.row_size
  Status: 76
***** 4. row *****
  Type: performance_schema
  Name: events_waits_history.row_count
  Status: 10000
***** 5. row *****
  Type: performance_schema
```

```

Name: events_waits_history.memory
Status: 760000
...
***** 57. row *****
Type: performance_schema
Name: performance_schema.memory
Status: 26459600
...

```

This statement is intended to help the DBA understand the effects that different Performance Schema options have on memory requirements. For a description of the field meanings, see [Section 13.7.5.16](#), “`SHOW ENGINE Syntax`”.

21.6. Performance Schema General Table Characteristics

The name of the `performance_schema` database is lowercase, as are the names of tables within it. Queries should specify the names in lowercase.

Most tables in the `performance_schema` database are read only and cannot be modified. Some of the setup tables have columns that can be modified to affect Performance Schema operation. Truncation is permitted to clear collected events, so `TRUNCATE TABLE` can be used on tables containing those kinds of information, such as tables named with a prefix of `events_waits_`.

`TRUNCATE TABLE` can also be used with summary tables, but the effect is to reset the summary columns to 0 or `NULL`, not to remove rows.

Privileges are as for other databases and tables:

- To retrieve from `performance_schema` tables, you must have the `SELECT` [697] privilege.
- To change those columns that can be modified, you must have the `UPDATE` [697] privilege.
- To truncate tables that can be truncated, you must have the `DROP` [696] privilege.

21.7. Performance Schema Table Descriptions

Tables in the `performance_schema` database can be grouped as follows:

- Setup tables. These tables are used to configure and display monitoring characteristics.
- Current events table. The `events_waits_current` table contains the most recent event for each thread.
- History tables. These tables have the same structure as `events_waits_current` but contain more rows. The `events_waits_history` table contains the most recent 10 events per thread. `events_waits_history_long` contains the most recent 10,000 events.

To change the sizes of these tables, set the

`performance_schema_events_waits_history_size` [2554] and `performance_schema_events_waits_history_long_size` [2554] system variables at server startup.

- Summary tables. These tables contain information aggregated over groups of events, including those that have been discarded from the history tables.
- Instance tables. These tables document what types of objects are instrumented. An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information.

- Miscellaneous tables. These do not fall into any of the other table groups.

21.7.1. Performance Schema Setup Tables

The setup tables provide information about the current instrumentation and enable the monitoring configuration to be changed. For this reason, some columns in these tables can be changed if you have the `UPDATE` [697] privilege.

The use of tables rather than individual variables for setup information provides a high degree of flexibility in modifying Performance Schema configuration. For example, you can use a single statement with standard SQL syntax to make multiple simultaneous configuration changes.

These setup tables are available:

- `setup_consumers`: The types of consumers for which event information can be stored
- `setup_instruments`: The classes of instrumented objects for which events can be collected
- `setup_timers`: The current event timer

21.7.1.1. The `setup_consumers` Table

The `setup_consumers` table lists the types of consumers for which event information can be stored:

```
mysql> SELECT * FROM setup_consumers;
+-----+-----+
| NAME                                     | ENABLED |
+-----+-----+
| events_waits_current                    | YES     |
| events_waits_history                    | YES     |
| events_waits_history_long               | YES     |
| events_waits_summary_by_thread_by_event_name | YES     |
| events_waits_summary_by_event_name      | YES     |
| events_waits_summary_by_instance        | YES     |
| file_summary_by_event_name              | YES     |
| file_summary_by_instance                | YES     |
+-----+-----+
```

The `setup_consumers` table has these columns:

- `NAME`

The consumer name.

- `ENABLED`

Whether the consumer is enabled. This column can be modified. If you disable a consumer, the server does not spend time adding event information to it.

Disabling the `events_waits_current` consumer disables everything else that depends on waits, such as the `events_waits_history` and `events_waits_history_long` tables, and all summary tables.

21.7.1.2. The `setup_instruments` Table

The `setup_instruments` table lists classes of instrumented objects for which events can be collected:

```
mysql> SELECT * FROM setup_instruments;
+-----+-----+-----+
| NAME                                     | ENABLED | TIMED |
+-----+-----+-----+
```

```

+-----+-----+-----+
...
| wait/synch/mutex/sql/LOCK_global_read_lock          | YES      | YES      |
| wait/synch/mutex/sql/LOCK_global_system_variables   | YES      | YES      |
| wait/synch/mutex/sql/LOCK_lock_db                  | YES      | YES      |
| wait/synch/mutex/sql/LOCK_manager                  | YES      | YES      |
...
| wait/synch/rwlock/sql/LOCK_grant                    | YES      | YES      |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger          | YES      | YES      |
| wait/synch/rwlock/sql/LOCK_sys_init_connect        | YES      | YES      |
| wait/synch/rwlock/sql/LOCK_sys_init_slave          | YES      | YES      |
...
| wait/io/file/sql/binlog                             | YES      | YES      |
| wait/io/file/sql/binlog_index                      | YES      | YES      |
| wait/io/file/sql/casetest                          | YES      | YES      |
| wait/io/file/sql/dbopt                             | YES      | YES      |
...

```

Each instrument added to the source code provides a row for this table, even when the instrumented code is not executed. When an instrument is enabled and executed, instrumented instances are created, which are visible in the `*_instances` tables.

The `setup_instruments` table has these columns:

- `NAME`

The instrument name. Instrument names have multiple parts and form a hierarchy, as discussed in [Section 21.4, “Performance Schema Instrument Naming Conventions”](#). Events produced from execution of an instrument have an `EVENT_NAME` value that is taken from the instrument `NAME` value. (Events do not really have a “name,” but this provides a way to associate events with instruments.)

- `ENABLED`

Whether the instrument is enabled. This column can be modified. A disabled instrument produces no events.

- `TIMED`

Whether the instrument is timed. This column can be modified.

If an enabled instrument is not timed, the instrument code is enabled, but the timer is not. Events produced by the instrument have `NULL` for the `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` timer values. This in turn causes those values to be ignored when calculating the sum, minimum, maximum, and average time values in summary tables.

21.7.1.3. The `setup_timers` Table

The `setup_timers` table shows the currently selected event timer:

```

mysql> SELECT * FROM setup_timers;
+-----+-----+
| NAME | TIMER_NAME |
+-----+-----+
| wait | CYCLE      |
+-----+-----+

```

The `setup_timers.TIMER_NAME` value can be changed to select a different timer. The value can be any of the values in the `performance_timers.TIMER_NAME` column. For an explanation of how event timing occurs, see [Section 21.2.3.1, “Performance Schema Event Timing”](#).

The `setup_timers` table has these columns:

- `NAME`

The type of instrument the timer is used for.

- `TIMER_NAME`

The timer that applies to the instrument type. This column can be modified.

The `setup_timers.TIMER_NAME` value can be changed to select a different timer. The value can be any of the values in the `performance_timers.TIMER_NAME` column. For an explanation of how event timing occurs, see [Section 21.2.3.1, “Performance Schema Event Timing”](#).

Modifications to the `setup_timers` table affect monitoring immediately. Events already in progress use the original timer for the begin time and the new timer for the end time, which leads to unpredictable results. If you make timer changes, you may want to use `TRUNCATE TABLE` to reset Performance Schema statistics.

21.7.2. Performance Schema Instance Tables

Instance tables document what types of objects are instrumented. They provide event names and explanatory notes or status information:

- `cond_instances`: Condition synchronization object instances
- `file_instances`: File instances
- `mutex_instances`: Mutex synchronization object instances
- `rwlock_instances`: Lock synchronization object instances

These tables list instrumented synchronization objects and files. There are three types of synchronization objects: `cond`, `mutex`, and `rwlock`. Each instance table has an `EVENT_NAME` or `NAME` column to indicate the instrument associated with each row. Instrument names have multiple parts and form a hierarchy, as discussed in [Section 21.4, “Performance Schema Instrument Naming Conventions”](#).

The `mutex_instances.LOCKED_BY_THREAD_ID` and `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` columns are extremely important for investigating performance bottlenecks or deadlocks. For examples of how to use them for this purpose, see [Section 21.11, “Using the Performance Schema to Diagnose Problems”](#)

21.7.2.1. The `cond_instances` Table

The `cond_instances` table lists all the conditions seen by the Performance Schema while the server executes. A condition is a synchronization mechanism used in the code to signal that a specific event has happened, so that a thread waiting for this condition can resume work.

When a thread is waiting for something to happen, the condition name is an indication of what the thread is waiting for, but there is no immediate way to tell which other thread, or threads, will cause the condition to happen.

The `cond_instances` table has these columns:

- `NAME`

The instrument name associated with the condition.

- `OBJECT_INSTANCE_BEGIN`

The address in memory of the condition that was instrumented.

21.7.2.2. The `file_instances` Table

The `file_instances` table lists all the files seen by the Performance Schema when executing file I/O instrumentation. If a file on disk has never been opened, it will not be in `file_instances`. When a file is deleted from the disk, it is also removed from the `file_instances` table.

The `file_instances` table has these columns:

- `FILE_NAME`

The file name.

- `EVENT_NAME`

The instrument name associated with the file.

- `OPEN_COUNT`

The count of open handles on the file. If a file was opened and then closed, it was opened 1 time, but `OPEN_COUNT` will be 0. To list all the files currently opened by the server, use `WHERE OPEN_COUNT > 0`.

21.7.2.3. The `mutex_instances` Table

The `mutex_instances` table lists all the mutexes seen by the Performance Schema while the server executes. A mutex is a synchronization mechanism used in the code to enforce that only one thread at a given time can have access to some common resource. The resource is said to be “protected” by the mutex.

When two threads executing in the server (for example, two user sessions executing a query simultaneously) do need to access the same resource (a file, a buffer, or some piece of data), these two threads will compete against each other, so that the first query to obtain a lock on the mutex will cause the other query to wait until the first is done and unlocks the mutex.

The work performed while holding a mutex is said to be in a “critical section,” and multiple queries do execute this critical section in a serialized way (one at a time), which is a potential bottleneck.

The `mutex_instances` table has these columns:

- `NAME`

The instrument name associated with the mutex.

- `OBJECT_INSTANCE_BEGIN`

The address in memory of the mutex that was instrumented.

- `LOCKED_BY_THREAD_ID`

When a thread currently has a mutex locked, `LOCKED_BY_THREAD_ID` is the `THREAD_ID` of the locking thread, otherwise it is `NULL`.

For every mutex instrumented in the code, the Performance Schema provides the following information.

- The `setup_instruments` table lists the name of the instrumentation point, with the prefix `wait/synch/mutex/`.

- When some code creates a mutex, a row is added to the `mutex_instances` table. The `OBJECT_INSTANCE_BEGIN` column is a property that uniquely identifies the mutex.
- When a thread attempts to lock a mutex, the `events_waits_current` table shows a row for that thread, indicating that it is waiting on a mutex (in the `EVENT_NAME` column), and indicating which mutex is waited on (in the `OBJECT_INSTANCE_BEGIN` column).
- When a thread succeeds in locking a mutex:
 - `events_waits_current` shows that the wait on the mutex is completed (in the `TIMER_END` and `TIMER_WAIT` columns)
 - The completed wait event is added to the `events_waits_history` and `events_waits_history_long` tables
 - `mutex_instances` shows that the mutex is now owned by the thread (in the `THREAD_ID` column).
- When a thread unlocks a mutex, `mutex_instances` shows that the mutex now has no owner (the `THREAD_ID` column is `NULL`).
- When a mutex object is destroyed, the corresponding row is removed from `mutex_instances`.

By performing queries on both of the following tables, a monitoring application or a DBA can detect bottlenecks or deadlocks between threads that involve mutexes:

- `events_waits_current`, to see what mutex a thread is waiting for
- `mutex_instances`, to see which other thread currently owns a mutex

21.7.2.4. The `rwlock_instances` Table

The `rwlock_instances` table lists all the `rwlock` instances (read write locks) seen by the Performance Schema while the server executes. An `rwlock` is a synchronization mechanism used in the code to enforce that threads at a given time can have access to some common resource following certain rules. The resource is said to be “protected” by the `rwlock`. The access is either shared (many threads can have a read lock at the same time) or exclusive (only one thread can have a write lock at a given time).

Depending on how many threads are requesting a lock, and the nature of the locks requested, access can be either granted in shared mode, granted in exclusive mode, or not granted at all, waiting for other threads to finish first.

The `rwlock_instances` table has these columns:

- `NAME`

The instrument name associated with the lock.

- `OBJECT_INSTANCE_BEGIN`

The address in memory of the lock that was instrumented.

- `WRITE_LOCKED_BY_THREAD_ID`

When a thread currently has an `rwlock` locked in exclusive (write) mode, `WRITE_LOCKED_BY_THREAD_ID` is the `THREAD_ID` of the locking thread, otherwise it is `NULL`.

- `READ_LOCKED_BY_COUNT`

When a thread currently has an `rwlock` locked in shared (read) mode, `READ_LOCKED_BY_COUNT` is incremented by 1. This is a counter only, so it cannot be used directly to find which thread holds a read lock, but it can be used to see whether there is a read contention on an `rwlock`, and see how many readers are currently active.

By performing queries on both of the following tables, a monitoring application or a DBA may detect some bottlenecks or deadlocks between threads that involve locks:

- `events_waits_current`, to see what `rwlock` a thread is waiting for
- `rwlock_instances`, to see which other thread currently owns an `rwlock`

There is a limitation: The `rwlock_instances` can be used only to identify the thread holding a write lock, but not the threads holding a read lock.

21.7.3. Performance Schema Wait Event Tables

These tables store wait events:

- `events_waits_current`: Current wait events
- `events_waits_history`: The most recent wait events for each thread
- `events_waits_history_long`: The most recent wait events overall

21.7.3.1. The `events_waits_current` Table

The `events_waits_current` table contains current wait events, one row per thread showing the current status of the thread's most recent monitored wait event.

The `events_waits_current` table can be truncated with `TRUNCATE TABLE`.

Of the tables that contain wait event rows, `events_waits_current` is the most fundamental. Other tables that contain wait event rows are logically derived from the current events. For example, the `events_waits_history` and `events_waits_history_long` tables are collections of the most recent wait events, up to a fixed number of rows.

The `events_waits_current` table has these columns:

- `THREAD_ID`

The thread associated with the event. The `THREAD_ID` and `EVENT_ID` values taken together form a primary key that uniquely identifies the row. No two rows will have the same pair of values.

- `EVENT_ID`

The thread current event number when the event starts.

- `EVENT_NAME`

The name of the instrument that produced the event. This is a `setup_instruments.NAME` value. Instrument names have multiple parts and form a hierarchy, as discussed in [Section 21.4, “Performance Schema Instrument Naming Conventions”](#).

- `SOURCE`

The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to

determine exactly what code is involved. For example, if a mutex or lock is being blocked, you can check the context in which this occurs.

- `TIMER_START`, `TIMER_END`, `TIMER_WAIT`

Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The `TIMER_START` and `TIMER_END` values indicate when event timing started and ended. `TIMER_WAIT` is the event elapsed time (duration).

If an event has not finished, `TIMER_END` and `TIMER_WAIT` are `NULL`.

If an event is produced from an instrument that has `TIMED = NO`, timing information is not collected, and `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` are all `NULL`.

For discussion of picoseconds as the unit for event times and factors that affect time values, see [Section 21.2.3.1, “Performance Schema Event Timing”](#).

- `SPINS`

For a mutex, the number of spin rounds. If the value is `NULL`, the code does not use spin rounds or spinning is not instrumented.

- `OBJECT_SCHEMA`, `OBJECT_NAME`, `OBJECT_TYPE`, `OBJECT_INSTANCE_BEGIN`

These columns identify the object “being acted on.” What that means depends on the object type.

For a synchronization object (`cond`, `mutex`, `rwlock`):

- `OBJECT_SCHEMA`, `OBJECT_NAME`, and `OBJECT_TYPE` are `NULL`.
- `OBJECT_INSTANCE_BEGIN` is the address of the synchronization object in memory.

For a file I/O object:

- `OBJECT_SCHEMA` is `NULL`.
- `OBJECT_NAME` is the file name.
- `OBJECT_TYPE` is `FILE`.
- `OBJECT_INSTANCE_BEGIN` is an address in memory.

An `OBJECT_INSTANCE_BEGIN` value itself has no meaning, except that different values indicate different objects. `OBJECT_INSTANCE_BEGIN` can be used for debugging. For example, it can be used with `GROUP BY OBJECT_INSTANCE_BEGIN` to see whether the load on 1,000 mutexes (that protect, say, 1,000 pages or blocks of data) is spread evenly or just hitting a few bottlenecks. This can help you correlate with other sources of information if you see the same object address in a log file or another debugging or performance tool.

- `NESTING_EVENT_ID`

Currently `NULL`.

- `OPERATION`

The type of operation performed, such as `lock`, `read`, or `write`.

- `NUMBER_OF_BYTES`

The number of bytes read or written by the operation.

- `FLAGS`

Reserved for future use.

21.7.3.2. The `events_waits_history` Table

The `events_waits_history` table contains the most recent 10 wait events per thread. The table size may be changed by modifying the `performance_schema_events_waits_history_size` [2554] system variable at server startup. As new events are added to the table, older events are discarded if the table is full. Events are not added to the table until they have ended.

The `events_waits_history` table has the same structure as `events_waits_current`. See [Section 21.7.3.1, “The `events_waits_current` Table”](#).

The `events_waits_history` table can be truncated with `TRUNCATE TABLE`.

21.7.3.3. The `events_waits_history_long` Table

The `events_waits_history_long` table contains the most recent 10,000 wait events. The table size may be changed by modifying the `performance_schema_events_waits_history_long_size` [2554] system variable at server startup. As new events are added to the table, older events are discarded if the table is full. Events are not added to the table until they have ended.

The `events_waits_history_long` table has the same structure as `events_waits_current`. See [Section 21.7.3.1, “The `events_waits_current` Table”](#).

The `events_waits_history_long` table can be truncated with `TRUNCATE TABLE`.

21.7.4. Performance Schema Summary Tables

Summary tables provide aggregated information for terminated events over time. The tables in this group summarize event data in different ways.

Event Wait Summaries:

- `events_waits_summary_global_by_event_name`: Wait events summarized per event name
- `events_waits_summary_by_instance`: Wait events summarized per instance
- `events_waits_summary_by_thread_by_event_name`: Wait events summarized per thread and event name

File I/O Summaries:

- `file_summary_by_event_name`: File events summarized per event name
- `file_summary_by_instance`: File events summarized per file instance

The `events_waits_summary_global_by_event_name` table was named `EVENTS_WAITS_SUMMARY_BY_EVENT_NAME` before MySQL 5.5.7.

Each summary table has grouping columns that determine how to group the data to be aggregated, and summary columns that contain the aggregated values. Tables that summarize events in similar ways often have similar sets of summary columns and differ only in the grouping columns used to determine how events are aggregated.

Summary tables can be truncated with `TRUNCATE TABLE`. The effect is to reset the summary columns to 0 or `NULL`, not to remove rows. This enables you to clear collected values and restart aggregation. That might be useful, for example, after you have made a runtime configuration change.

21.7.4.1. Event Wait Summary Tables

The event waits summary tables aggregate general information about event waits:

- `events_waits_summary_global_by_event_name`: Wait events summarized per event name
- `events_waits_summary_by_instance`: Wait events summarized per instance
- `events_waits_summary_by_thread_by_event_name`: Wait events summarized per thread and event name

For example:

```
mysql> SELECT * FROM events_waits_summary_global_by_event_name\G
...
***** 6. row *****
EVENT_NAME: wait/synch/mutex/sql/BINARY_LOG::LOCK_index
COUNT_STAR: 8
SUM_TIMER_WAIT: 2119302
MIN_TIMER_WAIT: 196092
AVG_TIMER_WAIT: 264912
MAX_TIMER_WAIT: 569421
...
***** 9. row *****
EVENT_NAME: wait/synch/mutex/sql/hash_filo::lock
COUNT_STAR: 69
SUM_TIMER_WAIT: 16848828
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 244185
MAX_TIMER_WAIT: 735345
...
```

`TRUNCATE TABLE` is permitted for wait summary tables. It resets the counters to zero rather than removing rows.

The event wait summary tables have these grouping columns to indicate how events are aggregated:

- `events_waits_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given instrument. An instrument might be used to create multiple instances of the instrumented object. For example, if there is an instrument for a mutex that is created for each connection, there are as many instances as there are connections. The summary row for the instrument summarizes over all these instances.
- `events_waits_summary_by_instance` has `EVENT_NAME` and `OBJECT_INSTANCE_BEGIN` columns. Each row summarizes events for a given instrument instance. If an instrument is used to create multiple instances, each instance has a unique `OBJECT_INSTANCE_BEGIN` value, so these instances are summarized separately in this table.
- `events_waits_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and instrument.

The event waits summary tables have these summary columns containing aggregated values:

- `COUNT_STAR`

The number of summarized events. This value includes all events, whether timed or not.

- [SUM_TIMER_WAIT](#)

The total wait time of the summarized timed events. This value is calculated only for timed events because nontimed events have a wait time of `NULL`. The same is true for the other `xxx_TIMER_WAIT` values.

- [MIN_TIMER_WAIT](#)

The minimum wait time of the summarized timed events.

- [AVG_TIMER_WAIT](#)

The average wait time of the summarized timed events.

- [MAX_TIMER_WAIT](#)

The maximum wait time of the summarized timed events.

21.7.4.2. File I/O Summary Tables

The file I/O summary tables aggregate information about I/O operations:

- [file_summary_by_event_name](#): File events summarized per event name
- [file_summary_by_instance](#): File events summarized per file instance

For example:

```
mysql> SELECT * FROM file_summary_by_instance\G
...
***** 2. row *****
      FILE_NAME: /var/mysql/share/english/errmsg.sys
      EVENT_NAME: wait/io/file/sql/ERRMSG
      COUNT_READ: 3
      COUNT_WRITE: 0
      SUM_NUMBER_OF_BYTES_READ: 42211
      SUM_NUMBER_OF_BYTES_WRITE: 0
...
***** 6. row *****
      FILE_NAME: /var/mysql/data/binlog.000001
      EVENT_NAME: wait/io/file/sql/binlog
      COUNT_READ: 0
      COUNT_WRITE: 0
      SUM_NUMBER_OF_BYTES_READ: 0
      SUM_NUMBER_OF_BYTES_WRITE: 0
...

```

`TRUNCATE TABLE` is permitted for file I/O summary tables. It resets the counters to zero rather than removing rows.

The file I/O summary tables have these grouping columns to indicate how events are aggregated:

- [file_summary_by_event_name](#) has an `EVENT_NAME` column. Each row summarizes events for a given instrument.
- [file_summary_by_instance](#) has `FILE_NAME` and `EVENT_NAME` columns. Each row summarizes events for a given file instrument instance.

The file I/O summary tables have these summary columns containing aggregated values:

- [COUNT_READ](#)

The number of read operations in the summarized events.

- [COUNT_WRITE](#)

The number of write operations in the summarized events.

- [SUM_NUMBER_OF_BYTES_READ](#)

The number of bytes read in the summarized events.

- [SUM_NUMBER_OF_BYTES_WRITE](#)

The number of bytes written in the summarized events.

The MySQL server uses several techniques to avoid I/O operations by caching information read from files, so it is possible that statements you might expect to result in I/O events will not. You may be able to ensure that I/O does occur by flushing caches or restarting the server to reset its state.

21.7.5. Performance Schema Miscellaneous Tables

The following sections describe tables that do not fall into the table categories discussed in the preceding sections:

- [performance_timers](#): Which event timers are available.
- [threads](#): Information about server threads.

21.7.5.1. The [performance_timers](#) Table

The [performance_timers](#) table shows which event timers are available:

```
mysql> SELECT * FROM performance_timers;
```

TIMER_NAME	TIMER_FREQUENCY	TIMER_RESOLUTION	TIMER_OVERHEAD
CYCLE	2389029850	1	72
NANOSECOND	NULL	NULL	NULL
MICROSECOND	1000000	1	585
MILLISECOND	1035	1	738
TICK	101	1	630

If the values associated with a given timer name are [NULL](#), that timer is not supported on your platform. The rows that do not contain [NULL](#) indicate which timers you can use in [setup_timers](#).

The [performance_timers](#) table has these columns:

- [TIMER_NAME](#)

The name by which to refer to the timer when configuring the [setup_timers](#) table.

- [TIMER_FREQUENCY](#)

The number of timer units per second. For a cycle timer, the frequency is generally related to the CPU speed. For example, on a system with a 2.4GHz processor, the [CYCLE](#) may be close to 2400000000.

- [TIMER_RESOLUTION](#)

Indicates the number of timer units by which timer values increase. If a timer has a resolution of 10, its value increases by 10 each time.

- [TIMER_OVERHEAD](#)

The minimal number of cycles of overhead to obtain one timing with the given timer. The Performance Schema determines this value by invoking the timer 20 times during initialization and picking the smallest value. The total overhead really is twice this amount because the instrumentation invokes the timer at the start and end of each event. The timer code is called only for timed events, so this overhead does not apply for nontimed events.

21.7.5.2. The `threads` Table

The `threads` table contains a row for each server thread:

```
mysql> SELECT * FROM threads;
+-----+-----+-----+
| THREAD_ID | PROCESSLIST_ID | NAME |
+-----+-----+-----+
| 0 | 0 | thread/sql/main |
| 1 | 0 | thread/innodb/io_handler_thread |
| 16 | 0 | thread/sql/signal_handler |
| 23 | 7 | thread/sql/one_connection |
| 5 | 0 | thread/innodb/io_handler_thread |
| 12 | 0 | thread/innodb/srv_lock_timeout_thread |
| 22 | 6 | thread/sql/one_connection |
| ... |
```

If you have the [PROCESS \[696\]](#) privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using).

The `threads` table has these columns:

- [THREAD_ID](#)

This is the unique identifier of an instrumented thread.

- [PROCESSLIST_ID](#)

For threads that are displayed in `INFORMATION_SCHEMA.PROCESSLIST`, this is the `INFORMATION_SCHEMA.ID` value, which is also the value that `CONNECTION_ID()` [1191] would return within that thread. For background threads (threads not associated with a user connection), `PROCESSLIST_ID` is 0, so the values are not unique.

This column was named `ID` before MySQL 5.5.8.

- [NAME](#)

`NAME` is the name associated with the instrumentation of the code in the server. For example, `thread/sql/one_connection` corresponds to the thread function in the code responsible for handling a user connection, and `thread/sql/main` stands for the `main()` function of the server.

The `threads` table was named `PROCESSLIST` before MySQL 5.5.6.

21.8. Performance Schema System Variables

The Performance Schema implements several system variables that provide configuration information:

```
mysql> SHOW VARIABLES LIKE 'perf%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| performance_schema | ON |
```

Performance Schema System Variables

performance_schema_events_waits_history_long_size	10000
performance_schema_events_waits_history_size	10
performance_schema_max_cond_classes	80
performance_schema_max_cond_instances	1000
performance_schema_max_file_classes	50
performance_schema_max_file_handles	32768
performance_schema_max_file_instances	10000
performance_schema_max_mutex_classes	200
performance_schema_max_mutex_instances	1000000
performance_schema_max_rwlock_classes	30
performance_schema_max_rwlock_instances	1000000
performance_schema_max_table_handles	100000
performance_schema_max_table_instances	50000
performance_schema_max_thread_classes	50
performance_schema_max_thread_instances	1000

Table 21.1. Performance Schema Variable Reference

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
performance_schema_events_waits_history_long_size [2554]	Yes	Yes	Yes		Global	No
Performance_schema_cond_classes_lost [2559]				Yes	Global	No
Performance_schema_cond_instances_lost [2559]				Yes	Global	No
performance_schema_events_waits_history_long_size [2554]	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_size [2554]	Yes	Yes	Yes		Global	No
Performance_schema_file_classes_lost [2559]				Yes	Global	No
Performance_schema_file_handles_lost [2559]				Yes	Global	No
Performance_schema_file_instances_lost [2559]				Yes	Global	No
Performance_schema_locker_lost [2559]				Yes	Global	No
performance_schema_max_cond_classes [2554]	Yes	Yes	Yes		Global	No
performance_schema_max_cond_instances [2554]	Yes	Yes	Yes		Global	No
performance_schema_max_file_classes [2555]	Yes	Yes	Yes		Global	No
performance_schema_max_file_handles [2555]	Yes	Yes	Yes		Global	No
performance_schema_max_file_instances [2556]	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_classes [2557]	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_instances [2558]	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_classes [2559]	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_instances [2560]	Yes	Yes	Yes		Global	No
performance_schema_max_table_handles [2557]	Yes	Yes	Yes		Global	No
performance_schema_max_table_instances [2558]	Yes	Yes	Yes		Global	No
performance_schema_max_thread_classes [2559]	Yes	Yes	Yes		Global	No
performance_schema_max_thread_instances [2560]	Yes	Yes	Yes		Global	No
Performance_schema_mutex_classes_lost [2559]				Yes	Global	No
Performance_schema_mutex_instances_lost [2559]				Yes	Global	No
Performance_schema_rwlock_classes_lost [2559]				Yes	Global	No
Performance_schema_rwlock_instances_lost [2559]				Yes	Global	No
Performance_schema_table_handles_lost [2559]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Performance_schema_table_instances_lost [2559]				Yes	Global	No
Performance_schema_thread_classes_lost [2559]				Yes	Global	No
Performance_schema_thread_instances_lost [2559]				Yes	Global	No

The variables have the following meanings:

- [performance_schema \[2554\]](#)

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema=#</code>
Option-File Format	<code>performance_schema=#</code>
Option Sets Variable	Yes, performance_schema [2554]
Variable Name	<code>performance_schema</code>
Variable Scope	Global
Dynamic Variable	No

The value of this variable is `ON` or `OFF` to indicate whether the Performance Schema is enabled. By default, the value is `OFF`. At server startup, you can specify this variable with no value or a value of 1 to enable it, or with a value of 0 to disable it.

- [performance_schema_events_waits_history_long_size \[2554\]](#)

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema_events_waits_history_long_size=#</code>
Option-File Format	<code>performance_schema_events_waits_history_long_size=#</code>
Option Sets Variable	Yes, performance_schema_events_waits_history_long_size [2554]
Variable Name	<code>performance_schema_events_waits_history_long_size</code>
Variable Scope	Global
Dynamic Variable	No

The number of rows in the `events_waits_history_long` table.

- [performance_schema_events_waits_history_size \[2554\]](#)

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema_events_waits_history_size=#</code>
Option-File Format	<code>performance_schema_events_waits_history_size=#</code>
Option Sets Variable	Yes, performance_schema_events_waits_history_size [2554]
Variable Name	<code>performance_schema_events_waits_history_size</code>
Variable Scope	Global
Dynamic Variable	No

The number of rows per thread in the `events_waits_history` table.

- [performance_schema_max_cond_classes \[2554\]](#)

Version Introduced	5.5.3	
Command-Line Format	<code>--performance_schema_max_cond_classes=#</code>	
Option-File Format	<code>performance_schema_max_cond_classes=#</code>	
Option Sets Variable	Yes, performance_schema_max_cond_classes [2554]	
Variable Name	<code>performance_schema_max_cond_classes</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>80</code>

The maximum number of condition instruments.

- [performance_schema_max_cond_instances \[2555\]](#)

Version Introduced	5.5.3	
Command-Line Format	<code>--performance_schema_max_cond_instances=#</code>	
Option-File Format	<code>performance_schema_max_cond_instances=#</code>	
Option Sets Variable	Yes, performance_schema_max_cond_instances [2555]	
Variable Name	<code>performance_schema_max_cond_instances</code>	
Variable Scope	Global	
Dynamic Variable	No	

The maximum number of instrumented condition objects.

- [performance_schema_max_file_classes \[2555\]](#)

Version Introduced	5.5.3	
Command-Line Format	<code>--performance_schema_max_file_classes=#</code>	
Option-File Format	<code>performance_schema_max_file_classes=#</code>	
Option Sets Variable	Yes, performance_schema_max_file_classes [2555]	
Variable Name	<code>performance_schema_max_file_classes</code>	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	<code>numeric</code>
	Default	<code>50</code>

The maximum number of file instruments.

- [performance_schema_max_file_handles \[2555\]](#)

Version Introduced	5.5.3	
Command-Line Format	<code>--performance_schema_max_file_handles=#</code>	

Option-File Format	<code>performance_schema_max_file_handles=#</code>	
Option Sets Variable	Yes, performance_schema_max_file_handles [2555]	
Variable Name	performance_schema_max_file_handles	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	32768

The maximum number of opened file objects.

The value of [performance_schema_max_file_handles \[2555\]](#) should be greater than the value of [open_files_limit \[535\]](#): [open_files_limit \[535\]](#) affects the maximum number of open file handles the server can support and [performance_schema_max_file_handles \[2555\]](#) affects how many of these file handles can be instrumented.

- [performance_schema_max_file_instances \[2556\]](#)

Version Introduced	5.5.3	
Command-Line Format	<code>--performance_schema_max_file_instances=#</code>	
Option-File Format	<code>performance_schema_max_file_instances=#</code>	
Option Sets Variable	Yes, performance_schema_max_file_instances [2556]	
Variable Name	performance_schema_max_file_instances	
Variable Scope	Global	
Dynamic Variable	No	

The maximum number of instrumented file objects.

- [performance_schema_max_mutex_classes \[2556\]](#)

Version Introduced	5.5.3	
Command-Line Format	<code>--performance_schema_max_mutex_classes=#</code>	
Option-File Format	<code>performance_schema_max_mutex_classes=#</code>	
Option Sets Variable	Yes, performance_schema_max_mutex_classes [2556]	
Variable Name	performance_schema_max_mutex_classes	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	200

The maximum number of mutex instruments.

- [performance_schema_max_mutex_instances \[2556\]](#)

Version Introduced	5.5.3	
---------------------------	-------	--

Command-Line Format	<code>--performance_schema_max_mutex_instances=#</code>
Option-File Format	<code>performance_schema_max_mutex_instances=#</code>
Option Sets Variable	Yes, performance_schema_max_mutex_instances [2556]
Variable Name	<code>performance_schema_max_mutex_instances</code>
Variable Scope	Global
Dynamic Variable	No

The maximum number of instrumented mutex objects.

- [performance_schema_max_rwlock_classes \[2557\]](#)

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema_max_rwlock_classes=#</code>
Option-File Format	<code>performance_schema_max_rwlock_classes=#</code>
Option Sets Variable	Yes, performance_schema_max_rwlock_classes [2557]
Variable Name	<code>performance_schema_max_rwlock_classes</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>numeric</code>
	Default <code>20</code>

The maximum number of rwlock instruments.

- [performance_schema_max_rwlock_instances \[2557\]](#)

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema_max_rwlock_instances=#</code>
Option-File Format	<code>performance_schema_max_rwlock_instances=#</code>
Option Sets Variable	Yes, performance_schema_max_rwlock_instances [2557]
Variable Name	<code>performance_schema_max_rwlock_instances</code>
Variable Scope	Global
Dynamic Variable	No

The maximum number of instrumented rwlock objects.

- [performance_schema_max_table_handles \[2557\]](#)

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema_max_table_handles=#</code>
Option-File Format	<code>performance_schema_max_table_handles=#</code>
Option Sets Variable	Yes, performance_schema_max_table_handles [2557]
Variable Name	<code>performance_schema_max_table_handles</code>
Variable Scope	Global

Dynamic Variable	No
-------------------------	----

The maximum number of opened table objects.

- [performance_schema_max_table_instances](#) [2558]

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema_max_table_instances=#</code>
Option-File Format	<code>performance_schema_max_table_instances=#</code>
Option Sets Variable	Yes, performance_schema_max_table_instances [2558]
Variable Name	<code>performance_schema_max_table_instances</code>
Variable Scope	Global
Dynamic Variable	No

The maximum number of instrumented table objects.

- [performance_schema_max_thread_classes](#) [2558]

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema_max_thread_classes=#</code>
Option-File Format	<code>performance_schema_max_thread_classes=#</code>
Option Sets Variable	Yes, performance_schema_max_thread_classes [2558]
Variable Name	<code>performance_schema_max_thread_classes</code>
Variable Scope	Global
Dynamic Variable	No
	Permitted Values
	Type <code>numeric</code>
	Default <code>50</code>

The maximum number of thread instruments.

- [performance_schema_max_thread_instances](#) [2558]

Version Introduced	5.5.3
Command-Line Format	<code>--performance_schema_max_thread_instances=#</code>
Option-File Format	<code>performance_schema_max_thread_instances=#</code>
Option Sets Variable	Yes, performance_schema_max_thread_instances [2558]
Variable Name	<code>performance_schema_max_thread_instances</code>
Variable Scope	Global
Dynamic Variable	No

The maximum number of instrumented thread objects.

The [max_connections](#) [519] and [max_delayed_threads](#) [520] system variables affect how many threads are run in the server. [performance_schema_max_thread_instances](#) [2558] affects how many of these running threads can be instrumented. If you increase

`max_connections` [519] or `max_delayed_threads` [520], you should consider increasing `performance_schema_max_thread_instances` [2558] so that `performance_schema_max_thread_instances` [2558] is greater than the sum of `max_connections` [519] and `max_delayed_threads` [520].

21.9. Performance Schema Status Variables

The Performance Schema implements several status variables that provide information about instrumentation that could not be loaded or created due to memory constraints:

```
mysql> SHOW STATUS LIKE 'perf%';
+-----+-----+
| Variable_name                | Value |
+-----+-----+
| Performance_schema_cond_classes_lost | 0     |
| Performance_schema_cond_instances_lost | 0     |
| Performance_schema_file_classes_lost  | 0     |
| Performance_schema_file_handles_lost  | 0     |
| Performance_schema_file_instances_lost | 0     |
| Performance_schema_locker_lost        | 0     |
| Performance_schema_mutex_classes_lost  | 0     |
| Performance_schema_mutex_instances_lost | 0     |
| Performance_schema_rwlock_classes_lost  | 0     |
| Performance_schema_rwlock_instances_lost | 0     |
| Performance_schema_table_handles_lost   | 0     |
| Performance_schema_table_instances_lost | 0     |
| Performance_schema_thread_classes_lost  | 0     |
| Performance_schema_thread_instances_lost | 0     |
+-----+-----+
```

Names for these variables have several forms:

- `Performance_schema_xxx_classes_lost`

How many instruments of type `xxx` could not be loaded.

- `Performance_schema_xxx_instances_lost`

How many instances of object type `xxx` could not be created.

- `Performance_schema_xxx_handles_lost`

How many instances of object type `xxx` could not be opened.

- `Performance_schema_locker_lost`

How many events are “lost” or not recorded, due to the following conditions:

- Events are recursive (for example, waiting for A caused a wait on B, which caused a wait on C).
- The depth of the nested events stack is greater than the limit imposed by the implementation.

Currently, events recorded by the Performance Schema are not recursive, so this variable should always be 0.

For information on using these variables to check Performance Schema status, see [Section 21.5, “Performance Schema Status Monitoring”](#).

21.10. Performance Schema and Plugins

Removing a plugin with `UNINSTALL PLUGIN` does not affect information already collected for code in that plugin. Time spent executing the code while the plugin was loaded was still spent even if the plugin is unloaded later. The associated event information, including aggregate information, remains readable in `performance_schema` database tables. For additional information about the effect of plugin installation and removal, see [Section 21.5, “Performance Schema Status Monitoring”](#).

A plugin implementor who instruments plugin code should document its instrumentation characteristics to enable those who load the plugin to account for its requirements. For example, a third-party storage engine should include in its documentation how much memory the engine needs for mutex and other instruments.

21.11. Using the Performance Schema to Diagnose Problems

The Performance Schema is a tool to help a DBA do performance tuning by taking real measurements instead of “wild guesses.” This section demonstrates some ways to use the Performance Schema for this purpose. The discussion here relies on the use of event filtering, which is described in [Section 21.2.3.2, “Performance Schema Event Filtering”](#).

The following example provides one methodology that you can use to analyze a repeatable problem, such as investigating a performance bottleneck. To begin, you should have a repeatable use case where performance is deemed “too slow” and needs optimization, and you should enable all instrumentation (no pre-filtering at all).

1. Run the use case.
2. Using the Performance Schema tables, analyze the root cause of the performance problem. This analysis will rely heavily on post-filtering.
3. For problem areas that are ruled out, disable the corresponding instruments. For example, if analysis shows that the issue is not related to file I/O in a particular storage engine, disable the file I/O instruments for that engine. Then truncate the history and summary tables to remove previously collected events.
4. Repeat the process at step 1.

At each iteration, the Performance Schema output, particularly the `events_waits_history_long` table, will contain less and less “noise” caused by nonsignificant instruments, and given that this table has a fixed size, will contain more and more data relevant to the analysis of the problem at hand.

At each iteration, investigation should lead closer and closer to the root cause of the problem, as the “signal/noise” ratio will improve, making analysis easier.

5. Once a root cause of performance bottleneck is identified, take the appropriate corrective action, such as:
 - Tune the server parameters (cache sizes, memory, and so forth).
 - Tune a query by writing it differently,
 - Tune the database schema (tables, indexes, and so forth).
 - Tune the code (this applies to storage engine or server developers only).
6. Start again at step 1, to see the effects of the changes on performance.

The `mutex_instances.LOCKED_BY_THREAD_ID` and `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` columns are extremely important for investigating performance bottlenecks or deadlocks. This is made possible by Performance Schema instrumentation as follows:

1. Suppose that thread 1 is stuck waiting for a mutex.
2. You can determine what the thread is waiting for:

```
SELECT * FROM events_waits_current WHERE THREAD_ID = thread_1;
```

Say the query result identifies that the thread is waiting for mutex A, found in `events_waits_current.OBJECT_INSTANCE_BEGIN`.

3. You can determine which thread is holding mutex A:

```
SELECT * FROM mutex_instances WHERE OBJECT_INSTANCE_BEGIN = mutex_A;
```

Say the query result identifies that it is thread 2 holding mutex A, as found in `mutex_instances.LOCKED_BY_THREAD_ID`.

4. You can see what thread 2 is doing:

```
SELECT * FROM events_waits_current WHERE THREAD_ID = thread_2;
```

Chapter 22. Connectors and APIs

Table of Contents

22.1. MySQL Connector/ODBC	2567
22.1.1. Connector/ODBC Versions	2568
22.1.2. Connector/ODBC Introduction	2569
22.1.3. Connector/ODBC Installation	2572
22.1.4. Configuring Connector/ODBC	2583
22.1.5. Connector/ODBC Examples	2604
22.1.6. Connector/ODBC Reference	2632
22.1.7. Connector/ODBC Notes and Tips	2639
22.1.8. Connector/ODBC Support	2650
22.2. MySQL Connector/Net	2651
22.2.1. Connector/Net Versions	2652
22.2.2. Installing Connector/Net	2654
22.2.3. Connector/Net Visual Studio Integration	2662
22.2.4. Connector/Net Tutorials	2697
22.2.5. Connector/Net Programming	2742
22.2.6. Connector/Net Connection String Options Reference	2784
22.2.7. Connector/Net API Reference	2791
22.2.8. Connector/Net Support	2901
22.2.9. Connector/Net FAQ	2902
22.3. MySQL Connector/J	2903
22.3.1. Overview of MySQL Connector/J	2904
22.3.2. Connector/J Versions	2904
22.3.3. Connector/J Installation	2905
22.3.4. Connector/J Examples	2910
22.3.5. Connector/J (JDBC) Reference	2911
22.3.6. JDBC Concepts	2945
22.3.7. Connection Pooling with Connector/J	2954
22.3.8. Load Balancing with Connector/J	2958
22.3.9. Failover with Connector/J	2960
22.3.10. Using the Connector/J Interceptor Classes	2962
22.3.11. Using Connector/J with Tomcat	2962
22.3.12. Using Connector/J with JBoss	2964
22.3.13. Using Connector/J with Spring	2964
22.3.14. Using Connector/J with GlassFish	2969
22.3.15. Troubleshooting Connector/J Applications	2975
22.3.16. Connector/J Support	2983
22.4. MySQL Connector/C++	2985
22.4.1. Installing the MySQL Connector/C++ Binary	2988
22.4.2. Installing MySQL Connector/C++ from Source	2990
22.4.3. MySQL Connector/C++ Building Windows Applications with Microsoft Visual Studio	2996
22.4.4. MySQL Connector/C++ Building Linux Applications with NetBeans	3004
22.4.5. MySQL Connector/C++ Getting Started: Usage Examples	3009
22.4.6. MySQL Connector/C++ Tutorials	3015
22.4.7. MySQL Connector/C++ Debug Tracing	3022
22.4.8. MySQL Connector/C++ Usage Notes	3024
22.4.9. MySQL Connector/C++ Known Bugs and Issues	3028
22.4.10. MySQL Connector/C++ Feature requests	3029
22.4.11. MySQL Connector/C++ Support	3029

22.5. MySQL Connector/C	3029
22.5.1. Building MySQL Connector/C from the Source Code	3030
22.5.2. Testing MySQL Connector/C	3032
22.5.3. MySQL Connector/C FAQ	3033
22.6. MySQL Connector/Python	3034
22.6.1. Guidelines for Python Developers	3034
22.6.2. Connector/Python Versions	3036
22.6.3. Connector/Python Installation	3036
22.6.4. Connector/Python Coding Examples	3038
22.6.5. Connector/Python Tutorials	3044
22.6.6. Connector/Python Connection Arguments	3045
22.6.7. Connector/Python API Reference	3047
22.6.8. Connector/Python FAQ	3066
22.7. libmysqld, the Embedded MySQL Server Library	3066
22.7.1. Compiling Programs with <code>libmysqld</code>	3066
22.7.2. Restrictions When Using the Embedded MySQL Server	3067
22.7.3. Options with the Embedded Server	3067
22.7.4. Embedded Server Examples	3068
22.8. MySQL C API	3071
22.8.1. C API Data Structures	3072
22.8.2. C API Function Overview	3077
22.8.3. C API Function Descriptions	3082
22.8.4. C API Prepared Statements	3132
22.8.5. C API Prepared Statement Data Structures	3133
22.8.6. C API Prepared Statement Function Overview	3140
22.8.7. C API Prepared Statement Function Descriptions	3142
22.8.8. C API Threaded Function Descriptions	3167
22.8.9. C API Embedded Server Function Descriptions	3168
22.8.10. C API Client Plugin Functions	3169
22.8.11. Common Questions and Problems When Using the C API	3172
22.8.12. Controlling Automatic Reconnection Behavior	3174
22.8.13. C API Support for Multiple Statement Execution	3175
22.8.14. C API Prepared Statement Problems	3177
22.8.15. C API Prepared Statement Handling of Date and Time Values	3177
22.8.16. C API Support for Prepared <code>CALL</code> Statements	3179
22.8.17. Building Client Programs	3183
22.9. MySQL PHP API	3185
22.9.1. Overview of the MySQL PHP drivers	3186
22.9.2. Original MySQL API (<code>Mysql</code>)	3193
22.9.3. MySQL Improved Extension (<code>Mysqli</code>)	3267
22.9.4. MySQL Functions (PDO_MYSQL) (<code>MySQL (PDO)</code>)	3497
22.9.5. MySQL Native Driver (<code>Mysqlnd</code>)	3502
22.9.6. Mysqlnd replication and load balancing plugin (<code>mysqlnd_ms</code>)	3535
22.9.7. Mysqlnd query result cache plugin (<code>mysqlnd_qc</code>)	3633
22.9.8. Mysqlnd user handler plugin (<code>mysqlnd_uh</code>)	3686
22.9.9. Mysqlnd connection multiplexing plugin (<code>mysqlnd_mux</code>)	3774
22.9.10. Mysqlnd Memache plugin (<code>mysqlnd_memcache</code>)	3778
22.9.11. Connector/PHP	3788
22.9.12. Common Problems with MySQL and PHP	3788
22.9.13. Enabling Both <code>mysql</code> and <code>mysqli</code> in PHP	3789
22.10. MySQL Perl API	3789
22.11. MySQL Python API	3790
22.12. MySQL Ruby APIs	3790
22.12.1. The MySQL/Ruby API	3790

22.12.2. The Ruby/MySQL API	3791
22.13. MySQL Tcl API	3791
22.14. MySQL Eiffel Wrapper	3791

MySQL Connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to the MySQL protocol and MySQL resources. Both Connectors and the APIs enable you to connect and execute MySQL statements from another language or environment, including Java (JDBC), ODBC, Perl, Python, PHP, Ruby, and native C and embedded MySQL instances.



Note

Connector version numbers do not correlate with MySQL Server version numbers. See also [Table 22.2, “MySQL Connector Versions and MySQL Server Versions”](#).

A number of connectors are developed by MySQL:

- [Connector/ODBC](#) provides driver support for connecting to a MySQL server using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix and Mac OS X platforms.
- [Connector/Net](#) enables developers to create .NET applications that use data stored in a MySQL database. Connector/Net implements a fully functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that want to use Connector/Net can be written in any of the supported .NET languages.

The MySQL Visual Studio Plugin works with Connector/Net and Visual Studio 2005. The plugin is a MySQL DDEX Provider, which means that you can use the schema and data manipulation tools within Visual Studio to create and edit objects within a MySQL database.

- [Connector/J](#) provides driver support for connecting to MySQL from a Java application using the standard Java Database Connectivity (JDBC) API.
- [Connector/Python](#) provides driver support for connecting to MySQL from a Python program using an API that is compliant with the [Python DB API version 2.0](#). No additional Python modules or MySQL client libraries are required.
- [Connector/C++](#) is a tool that enables easy deployment and management of MySQL server and database through your C++ application.
- [Connector/C](#) is a standalone replacement for the MySQL Client Library (`libmysql`).

There are two direct access methods for using MySQL natively within a C application:

- The C API provides low-level access to the MySQL protocol through the `libmysql` client library; this is the primary method used to connect to an instance of the MySQL server, and is used both by MySQL command line clients and many of the APIs also detailed in this section. MySQL Connector/C can now also be used for this purpose.
- `libmysqld` is an embedded MySQL server library that enables you to embed an instance of the MySQL server into your C applications.

If you need to access MySQL from a C application, or build an interface to MySQL for a language not supported by the Connectors or APIs in this chapter, the C API is where you would start. A number of programmers utilities are available to help with the process, and also covered in this section.

The remaining APIs provide an interface to MySQL from specific application languages. These solutions are not developed or supported by MySQL. Basic information on their usage and abilities is provided here for reference purposes only.

All the language APIs are developed using one of two methods, using `libmysql` or by building a *native driver*. The two solutions offer different benefits:

- Using `libmysql` offers complete compatibility with MySQL as it uses the same libraries as the MySQL client applications. However, the feature set is limited to the implementation and interfaces exposed through `libmysql` and the performance may be lower as data is copied between the native language, and the MySQL API components. MySQL Connector/C is a possible alternative to using `libmysql`.
- *Native drivers* are an implementation of the MySQL network protocol entirely within the host language or environment. Native drivers are fast, as there is less copying of data between components, and they can offer advanced functionality not available through the standard MySQL API. Native drivers are also easier to build and deploy, as you do not need a copy of the MySQL client libraries to build the native driver components.

A list of many of the libraries and interfaces available for MySQL are shown in the table. See [Table 22.1, “MySQL APIs and Interfaces”](#).

Table 22.1. MySQL APIs and Interfaces

Environment	API	Type	Notes
Ada	MySQL Bindings for GNU Ada	<code>libmysql</code>	See MySQL Bindings for GNU Ada
C	Connector/C	Replacement for <code>libmysql</code>	See Section 22.5, “MySQL Connector/C” .
C++	Connector/C++	<code>libmysql</code>	See Section 22.4, “MySQL Connector/C++” .
	MySQL++	<code>libmysql</code>	See MySQL++ Web site .
	MySQL wrapped	<code>libmysql</code>	See MySQL wrapped .
Cocoa	MySQL-Cocoa	<code>libmysql</code>	Compatible with the Objective-C Cocoa environment. See http://mysql-cocoa.sourceforge.net/
D	MySQL for D	<code>libmysql</code>	See MySQL for D .
Eiffel	Eiffel MySQL	<code>libmysql</code>	See Section 22.14, “MySQL Eiffel Wrapper” .
Erlang	<code>erlang-mysql-driver</code>	<code>libmysql</code>	See erlang-mysql-driver .
Haskell	Haskell MySQL Bindings	Native Driver	See Brian O’Sullivan’s pure Haskell MySQL bindings .
	<code>hsqldb-mysql</code>	<code>libmysql</code>	See MySQL driver for Haskell .
Java/JDBC	Connector/J	Native Driver	See Section 22.3, “MySQL Connector/J” .
Kaya	MyDB	<code>libmysql</code>	See MyDB .
Lua	LuaSQL	<code>libmysql</code>	See LuaSQL .
.NET/Mono	Connector/Net	Native Driver	See Section 22.2, “MySQL Connector/Net” .
Objective Caml	MySQL Bindings for Objective Caml	<code>libmysql</code>	See MySQL Bindings for Objective Caml .
Octave	Database bindings for GNU Octave	<code>libmysql</code>	See Database bindings for GNU Octave .
ODBC	Connector/ODBC	<code>libmysql</code>	See Section 22.1, “MySQL Connector/ODBC” .

Environment	API	Type	Notes
Perl	<code>DBI/DBD::mysql</code>	<code>libmysql</code>	See Section 22.10 , “MySQL Perl API”.
	<code>Net::MySQL</code>	Native Driver	See <code>Net::MySQL</code> at CPAN
PHP	<code>mysql</code> , <code>ext/mysql</code> interface (deprecated)	<code>libmysql</code>	See Section 22.9.2 , “Original MySQL API (<code>mysql</code>)”.
	<code>mysqli</code> , <code>ext/mysqli</code> interface	<code>libmysql</code>	See Section 22.9.3 , “MySQL Improved Extension (<code>mysqli</code>)”.
	<code>PDO_MYSQL</code>	<code>libmysql</code>	See Section 22.9.4 , “MySQL Functions (<code>PDO_MYSQL</code>) (<code>MySQL (PDO)</code>)”.
	<code>PDO mysqlnd</code>	Native Driver	
Python	<code>Connector/Python</code>	Native Driver	See Section 22.6 , “MySQL Connector/Python”.
	<code>MySQLdb</code>	<code>libmysql</code>	See Section 22.11 , “MySQL Python API”.
Ruby	<code>MySQL/Ruby</code>	<code>libmysql</code>	Uses <code>libmysql</code> . See Section 22.12.1 , “The MySQL/Ruby API”.
	<code>Ruby/MySQL</code>	Native Driver	See Section 22.12.2 , “The Ruby/MySQL API”.
Scheme	<code>Myscsh</code>	<code>libmysql</code>	See <code>Myscsh</code> .
SPL	<code>sql_mysql</code>	<code>libmysql</code>	See <code>sql_mysql</code> for SPL.
Tcl	<code>MySQLtcl</code>	<code>libmysql</code>	See Section 22.13 , “MySQL Tcl API”.

Table 22.2. MySQL Connector Versions and MySQL Server Versions

Connector	Connector version	MySQL Server version
Connector/C++	1.0.5 GA	5.6, 5.5, 5.4, 5.1
Connector/J	5.1.8	5.6, 5.5, 5.4, 5.1, 5.0, 4.1
Connector/Net	6.5	5.6, 5.5, 5.4, 5.1, 5.0
Connector/Net	6.4	5.6, 5.5, 5.4, 5.1, 5.0
Connector/Net	6.3	5.6, 5.5, 5.4, 5.1, 5.0
Connector/Net	6.2 (No longer supported)	5.6, 5.5, 5.4, 5.1, 5.0
Connector/Net	6.1 (No longer supported)	5.6, 5.5, 5.4, 5.1, 5.0
Connector/Net	6.0 (No longer supported)	5.6, 5.5, 5.4, 5.1, 5.0
Connector/Net	5.2 (No longer supported)	5.6, 5.5, 5.4, 5.1, 5.0
Connector/Net	1.0 (No longer supported)	5.0, 4.0
Connector/ODBC	5.1	5.6, 5.5, 5.4, 5.1, 5.0, 4.1.1+
Connector/ODBC	3.51 (Unicode not supported)	5.6, 5.5, 5.4, 5.1, 5.0, 4.1

22.1. MySQL Connector/ODBC

The MySQL Connector/ODBC is the name for the family of MySQL ODBC drivers (previously called MyODBC drivers) that provide access to a MySQL database using the industry standard Open Database Connectivity (ODBC) API. This reference covers Connector/ODBC 5.2, which includes the functionality

Key connector/ODBC topics include:

of the Unicode driver and the ANSI driver, which formerly were split between Connector/ODBC 5.1 and Connector/ODBC 3.51.

MySQL Connector/ODBC provides both driver-manager based and native interfaces to the MySQL database, with full support for MySQL functionality, including stored procedures, [transactions](#) and, with Connector/ODBC 5.1 and higher, full Unicode compliance.

For more information on the ODBC API standard and how to use it, refer to <http://support.microsoft.com/kb/110093>.

The application development section of the ODBC API reference assumes a good working knowledge of C, general DBMS, and a familiarity with MySQL. For more information about MySQL functionality and its syntax, refer to <http://dev.mysql.com/doc/>.

Typically, you need to install Connector/ODBC only on Windows machines. For Unix and Mac OS X, you can use the native MySQL network or named pipes to communicate with your MySQL database. You may need Connector/ODBC for Unix or Mac OS X if you have an application that requires an ODBC interface to communicate with the database. Applications that require ODBC to communicate with MySQL include ColdFusion, Microsoft Office, and Filemaker Pro.

Key connector/ODBC topics include:

- Installing Connector/ODBC: [Section 22.1.3, “Connector/ODBC Installation”](#).
- The configuration options: [Section 22.1.4.2, “Connector/ODBC Connection Parameters”](#).
- An example that connects to a MySQL database from a Windows host: [Section 22.1.5.2, “Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC”](#).
- An example that uses Microsoft Access as an interface to a MySQL database: [Section 22.1.5.4, “Using Connector/ODBC with Microsoft Access”](#).
- General tips and notes, including how to obtain the last auto-increment ID: [Section 22.1.7.1, “Connector/ODBC General Functionality”](#).
- Application-specific usage tips and notes: [Section 22.1.7.2, “Connector/ODBC Application-Specific Tips”](#).
- A FAQ (Frequently Asked Questions) list: [Section 22.1.7.3, “Connector/ODBC Errors and Resolutions \(FAQ\)”](#).
- Additional Connector/ODBC support options: [Section 22.1.8, “Connector/ODBC Support”](#).

For release notes detailing the changes in each release of Connector/ODBC, see [MySQL Connector/ODBC Release Notes](#).

22.1.1. Connector/ODBC Versions

These are the versions of Connector/ODBC that are currently available:

- Connector/ODBC 5.2, currently in GA status, upgrades the ANSI driver of Connector/ODBC 3.51 to the 5.x code base. It also includes new features, such as enabling server-side prepared statements by default. At installation time, you can choose the Unicode driver for the broadest compatibility with data sources using various character sets, or the ANSI driver for optimal performance with a more limited range of character sets. It works with MySQL versions 4.1.1 and higher.
- Connector/ODBC 5.1, now superceded by 5.2, is a partial rewrite of the of the 3.51 code base, and is designed to work with MySQL versions 4.1.1 and newer.

Connector/ODBC 5.1 also includes the following changes and improvements over the 3.51 release:

- Improved support on Windows 64-bit platforms.
- Full Unicode support at the driver level. This includes support for the `SQL_WCHAR` data type, and support for Unicode login, password and DSN configurations. For more information, see [Microsoft Knowledgebase Article #716246](#).
- Support for the `SQL_NUMERIC_STRUCT` data type, which provides easier access to the precise definition of numeric values. For more information, see [Microsoft Knowledgebase Article #714556](#)
- Native Windows setup library. This replaces the Qt library based interface for configuring DSN information within the ODBC Data Sources application.
- Support for the ODBC descriptor, which improves the handling and metadata of columns and parameter data. For more information, see [Microsoft Knowledgebase Article #716339](#).
- Connector/ODBC 3.51 is the current release of the 32-bit ODBC driver, also known as the MySQL ODBC 3.51 driver. Connector/ODBC 3.51 has support for ODBC 3.5x specification level 1 (complete core API + level 2 features) to continue to provide all functionality of ODBC for accessing MySQL.

With MySQL Connector/ODBC 5.2 now offering both ANSI and Unicode drivers, you should expect to upgrade relatively soon.

The manual for versions of Connector/ODBC older than 5.2 can be located in the corresponding binary or source distribution. Please note that versions of Connector/ODBC earlier than the 3.51 revision were not fully compliant with the ODBC specification.



Note

From this section onward, the primary focus of this guide is the Connector/ODBC 5.2 driver.



Note

Version numbers for MySQL products are formatted as X.X.X. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.XX.XX. For example, the official MySQL formatted version number 5.0.9 may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display format is different.

22.1.2. Connector/ODBC Introduction

ODBC (Open Database Connectivity) provides a way for client programs to access a wide range of databases or data sources. ODBC is a standardized API that enables connections to SQL database servers. It was developed according to the specifications of the SQL Access Group and defines a set of function calls, error codes, and data types that can be used to develop database-independent applications. ODBC usually is used when database independence or simultaneous access to different data sources is required.

For more information about ODBC, refer to <http://support.microsoft.com/kb/110093>.

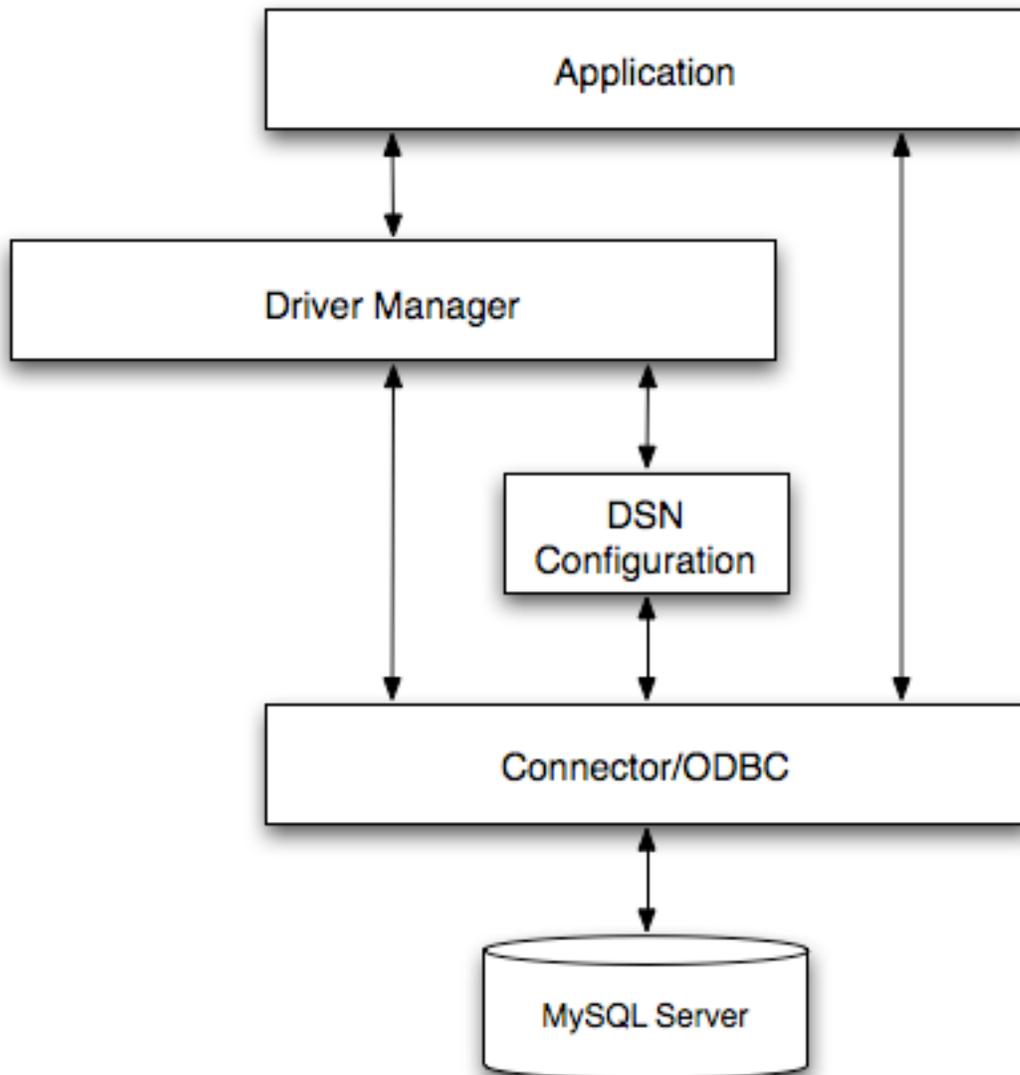
22.1.2.1. General Information About ODBC and Connector/ODBC

Open Database Connectivity (ODBC) is a widely accepted application-programming interface (API) for database access. It is based on the Call-Level Interface (CLI) specifications from X/Open and ISO/IEC for database APIs and uses Structured Query Language (SQL) as its database access language.

A survey of ODBC functions supported by Connector/ODBC is given at [Section 22.1.6.1, "Connector/ODBC API Reference"](#). For general information about ODBC, see <http://support.microsoft.com/kb/110093>.

22.1.2.1.1. Connector/ODBC Architecture

The Connector/ODBC architecture is based on five components, as shown in the following diagram:



- **Application:**

The Application uses the ODBC API to access the data from the MySQL server. The ODBC API in turn communicates with the Driver Manager. The Application communicates with the Driver Manager using the standard ODBC calls. The Application does not care where the data is stored, how it is stored, or even how the system is configured to access the data. It needs to know only the Data Source Name (DSN).

A number of tasks are common to all applications, no matter how they use ODBC. These tasks are:

- Selecting the MySQL server and connecting to it.

- Submitting SQL statements for execution.
- Retrieving results (if any).
- Processing errors.
- [Committing](#) or [rolling back](#) the [transaction](#) enclosing the SQL statement.
- Disconnecting from the MySQL server.

Because most data access work is done with SQL, the primary tasks for applications that use ODBC are submitting SQL statements and retrieving any results generated by those statements.

- **Driver manager:**

The Driver Manager is a library that manages communication between application and driver or drivers. It performs the following tasks:

- Resolves Data Source Names (DSN). The DSN is a configuration string that identifies a given database driver, database, database host and optionally authentication information that enables an ODBC application to connect to a database using a standardized reference.

Because the database connectivity information is identified by the DSN, any ODBC compliant application can connect to the data source using the same DSN reference. This eliminates the need to separately configure each application that needs access to a given database; instead you instruct the application to use a pre-configured DSN.

- Loading and unloading of the driver required to access a specific database as defined within the DSN. For example, if you have configured a DSN that connects to a MySQL database then the driver manager will load the Connector/ODBC driver to enable the ODBC API to communicate with the MySQL host.
- Processes ODBC function calls or passes them to the driver for processing.

- **Connector/ODBC Driver:**

The Connector/ODBC driver is a library that implements the functions supported by the ODBC API. It processes ODBC function calls, submits SQL requests to MySQL server, and returns results back to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by MySQL.

- **DSN Configuration:**

The ODBC configuration file stores the driver and database information required to connect to the server. It is used by the Driver Manager to determine which driver to be loaded according to the definition in the DSN. The driver uses this to read connection parameters based on the DSN specified. For more information, [Section 22.1.4, "Configuring Connector/ODBC"](#).

- **MySQL Server:**

The MySQL database where the information is stored. The database is used as the source of the data (during queries) and the destination for data (during inserts and updates).

22.1.2.1.2. ODBC Driver Managers

An ODBC Driver Manager is a library that manages communication between the ODBC-aware application and any drivers. Its main functionality includes:

- Resolving Data Source Names (DSN).
- Driver loading and unloading.
- Processing ODBC function calls or passing them to the driver.

Both Windows and Mac OS X include ODBC driver managers with the operating system. Most ODBC Driver Manager implementations also include an administration application that makes the configuration of DSN and drivers easier. Examples and information on these managers, including Unix ODBC driver managers, are listed below:

- Microsoft Windows ODBC Driver Manager (`odbc32.dll`), <http://support.microsoft.com/kb/110093>.
- Mac OS X includes `ODBC Administrator`, a GUI application that provides a simpler configuration mechanism for the Unix iODBC Driver Manager. You can configure DSN and driver information either through ODBC Administrator or through the iODBC configuration files. This also means that you can test ODBC Administrator configurations using the `iodbctest` command. <http://www.apple.com>.
- `unixODBC` Driver Manager for Unix (`libodbc.so`). See <http://www.unixodbc.org>, for more information. The `unixODBC` Driver Manager includes the Connector/ODBC driver 3.51 in the installation package, starting with version `unixODBC 2.1.2`.
- `iODBC` ODBC Driver Manager for Unix (`libiodbc.so`), see <http://www.iodbc.org>, for more information.

22.1.3. Connector/ODBC Installation

This section explains where to download Connector/ODBC, and how to run the installer, copy the files manually, or build from source.

Where to Get Connector/ODBC

You can get a copy of the latest version of Connector/ODBC binaries and sources from our Web site at <http://dev.mysql.com/downloads/connector/odbc/>.

For more information about Connector/ODBC, visit <http://www.mysql.com/products/myodbc/>.

For more information about licensing, visit <http://www.mysql.com/company/legal/licensing/>.

Choosing Unicode or ANSI Driver

Connector/ODBC offers the flexibility to handle data using any character set through its **Unicode-enabled** driver, or the maximum raw speed for a more limited range of character sets through its **ANSI** driver. Some users postponed their upgrade to Connector/ODBC 5.1, remaining with the older 3.51 version to keep this performance edge. As of Connector/ODBC 5.2, both kinds of drivers are available based on the 5.x code base: you can choose either a Unicode-enabled driver or an ANSI driver on the download page. The Unicode-enabled driver, recommended for most users, has no special qualifier in the download filename. The ANSI driver includes `-ansi-` in the download filename.



Note

You can install either the Unicode driver on a Windows system, or the ANSI driver, or both. The drivers are distinguished in the list of installed software and in the names of libraries and directories by a `w` (for “wide characters”) for the Unicode driver, and `a` in the ANSI driver.

Choosing Binary or Source Installation Method

You can install the Connector/ODBC drivers using two different methods:

- The **binary installation** is the easiest and most straightforward method of installation. You receive all the necessary libraries and other files pre-built, with an installer program or batch script to perform all necessary copying and configuration.
- The **source installation** method is intended for platforms where a binary installation package is not available, or in situations where you want to customize or modify the installation process or Connector/ODBC drivers before installation.

If a binary distribution is not available for a particular platform, and you build the driver from the original source code, you can contribute the binaries you create to MySQL by sending a mail message to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com), so that it becomes available for other users.

Supported Platforms

Connector/ODBC can be used on all major platforms supported by MySQL. You can install it on:

- Windows 95, 98, Me, NT, 2000, XP, 2003, Vista and 7.
- All Unix-like Operating Systems, including: AIX, Amiga, BSDI, DEC, FreeBSD, HP-UX 10/11, Linux, NetBSD, OpenBSD, OS/2, SGI Irix, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64 Unix.
- Mac OS X and Mac OS X Server.



Note

On all non-Windows platforms except Mac OS X, the driver is built against `unixODBC` and is expecting a 2-byte `SQLWCHAR`, not 4 bytes as `iODBC` is using. For this reason, the binaries are **only** compatible with `unixODBC`; recompile the driver against `iODBC` to use them together. For further information, see [Section 22.1.2.1.2, “ODBC Driver Managers”](#).

For further instructions, consult the documentation corresponding to the platform where you are installing and whether you are running a binary installer or building from source:

Platform	Binary Installer	Build from Source
Windows	Installation Instructions	Build Instructions
Unix/Linux	Installation Instructions	Build Instructions
Mac OS X	Installation Instructions	

22.1.3.1. Installing Connector/ODBC on Windows

Before installing the Connector/ODBC drivers on Windows, ensure that your Microsoft Data Access Components (MDAC) are up to date. You can obtain the latest version from the [Microsoft Data Access and Storage](#) Web site.

There are different distribution types to use when installing for Windows. The software that is installed is identical in each case, only the installation method is different.

- The **zipped installer** consists of a zipped package containing a standalone installation application. To install from this package, unzip the installer, and then run the installation application.

The **MSI installer** is an installation file that can be used with the installer included in Windows 2000, Windows XP and Windows Server 2003.

See [Section 22.1.3.1.1, “Installing the Windows Connector/ODBC Driver Using an Installer”](#) for the steps to complete the installation with either of these installers.

- The **zipped DLL** package contains DLL files that must be manually installed. See [Section 22.1.3.1.2, “Installing the Windows Connector/ODBC Driver using the Zipped DLL Package”](#) for the steps to complete this type of installation.

**Note**

An OLEDB/ODBC driver for Windows 64-bit is available from [Microsoft Downloads](#).

22.1.3.1.1. Installing the Windows Connector/ODBC Driver Using an Installer

The zipped or MSI installer packages offer a very simple method for installing the Connector/ODBC drivers. If you have downloaded the zipped installer, extract the installer application. The basic installation process is identical for both installers.

Follow these steps to complete the installation:

1. Double-click the standalone installer that you extracted, or the MSI file you downloaded.
2. The MySQL Connector/ODBC Setup Wizard starts. Click the **Next** button to begin the installation process.
3. Choose the installation type. The **Typical** installation provides the standard files needed to connect to a MySQL database using ODBC. The **Complete** option installs all the available files, including debug and utility components. Oracle recommends choosing one of these two options to complete the installation. If you choose one of these methods, click **Next**, then proceed to step 5.

You can also choose a **Custom** installation, where you select the individual components to install. If you choose this method, click **Next**, then proceed to step 4.

4. If you have chosen a custom installation, use the pop-ups to select which components to install, then click **Next** to install the necessary files.
5. Once the files are copied to their final locations, the installation is complete. Click **Finish** to exit the installer.

Now that the installation is complete, configure your ODBC connections using [Section 22.1.4, “Configuring Connector/ODBC”](#).

22.1.3.1.2. Installing the Windows Connector/ODBC Driver using the Zipped DLL Package

If you have downloaded the zipped DLL package:

1. Unzip the installation files.
2. Run the included batch file to perform an installation to the default locations.
3. Alternatively, install the individual files required for Connector/ODBC operation manually.

**Note**

The following instructions only work for 32-bit Windows systems. If you have a 64-bit Windows system, use the MSI installer, which installs both the 32-bit and 64-bit drivers to the correct locations.

To install using the **batch file**:

1. Unzip the Connector/ODBC zipped DLL package.
2. Open a command prompt.
3. Change to the directory created when you unzipped the Connector/ODBC zipped DLL package.
4. Run `Install.bat`:

```
C:\> Install.bat
```

This copies the necessary files into the default location, and then registers the Connector/ODBC driver with the Windows ODBC manager.



Note

Changing or adding a new DSN (data source name) may be accomplished using either the GUI, or from the command-line using `myodbc-installer.exe` with Connector/ODBC 5.x, or `myodbc3i.exe` with Connector/ODBC 3.51,

Although Oracle recommends installing these files in the standard location, you can also copy the files by hand to an alternative location - for example, to run or test different versions of the Connector/ODBC driver on the same machine. To **copy the files** to a location of your choice, use the following steps:

1. Unzip the Connector/ODBC zipped DLL package.
2. Open a command prompt.
3. Change to the directory created when you unzipped the Connector/ODBC zipped DLL package.
4. Copy the library files to a suitable directory. The default location is the default Windows system directory `\Windows\System32`:

```
C:\> copy lib\myodbc5S.dll \Windows\System32
C:\> copy lib\myodbc5S.lib \Windows\System32

If installing the Unicode-enabled driver:
C:\> copy lib\myodbc5w.dll \Windows\System32
C:\> copy lib\myodbc5w.lib \Windows\System32

If installing the ANSI driver:
C:\> copy lib\myodbc5a.dll \Windows\System32
C:\> copy lib\myodbc5a.lib \Windows\System32
```

5. Copy the Connector/ODBC tools. These must be placed in a directory that is in the system `%PATH%`. The default is to install these into the Windows system directory `\Windows\System32`:

```
C:\> copy bin\myodbc3i.exe \Windows\System32
C:\> copy bin\myodbc3m.exe \Windows\System32
C:\> copy bin\myodbc3c.exe \Windows\System32
```

6. Optionally, copy the help files. For these files to be accessible through the help system, they must be installed in the Windows system directory:

```
C:\> copy doc\*.hlp \Windows\System32
```

7. Finally, register the Connector/ODBC driver with the ODBC manager:

```
For Unicode-enabled driver:
C:\> myodbc3i -a -d -t"MySQL ODBC 5.2 Driver;\
```

```
DRIVER=myodbc5w.dll;SETUP=myodbc5S.dll"
```

For ANSI driver:

```
C:\> myodbc3i -a -d -t"MySQL ODBC 5.2 Driver;\
DRIVER=myodbc5a.dll;SETUP=myodbc5S.dll"
```

If you installed these files into a non-default location, change the references to the DLL files and command location in the above statement

22.1.3.2. Installing Connector/ODBC on Unix

There are two methods available for installing Connector/ODBC on Unix from a binary distribution. For most Unix environments, you will use the **tarball** distribution. For Linux systems, there is also an **RPM** distribution available.



Note

To install Connector/ODBC 5.x on Unix requires unixODBC 2.2.12 or later to be installed.

22.1.3.2.1. Installing Connector/ODBC from a Binary Tarball Distribution

To install the driver from a tarball distribution (`.tar.gz` file), download the latest version of the driver for your operating system and follow these steps that demonstrate the process using the Linux version of the tarball, substituting the appropriate file and directory names based on the package you download:

```
shell> su root
shell> gunzip mysql-connector-odbc-5.2.2-i686-pc-linux.tar.gz
shell> tar xvf mysql-connector-odbc-5.2.2-i686-pc-linux.tar
shell> cd mysql-connector-odbc-5.2.2-i686-pc-linux
```

Read the installation instructions in the `INSTALL` file and execute these commands.

Then see [Section 22.1.4.5, "Configuring a Connector/ODBC DSN on Unix"](#) to configure the DSN for Connector/ODBC. For more information, refer to the `INSTALL` file that comes with your distribution.

22.1.3.2.2. Installing Connector/ODBC from an RPM Distribution

To install or upgrade Connector/ODBC from an RPM distribution on Linux, simply download the RPM distribution of the latest version of Connector/ODBC and follow the instructions below. Use `su root` to become `root`, then install the RPM file.

If you are installing for the first time:

```
shell> su root
shell> rpm -ivh mysql-connector-odbc-5.2.2.i386.rpm
```

If the driver exists, upgrade it like this:

```
shell> su root
shell> rpm -Uvh mysql-connector-odbc-5.2.2.i386.rpm
```

If there is any dependency error for MySQL client library, `libmysqlclient`, simply ignore it by supplying the `--nodeps` option, and then make sure the MySQL client shared library is in the path or set through `LD_LIBRARY_PATH`.

This installs the driver libraries and related documents to `/usr/local/lib` and `/usr/share/doc/MyODBC`, respectively. See [Section 22.1.4.5, "Configuring a Connector/ODBC DSN on Unix"](#) for the post-installation configuration steps.

To **uninstall** the driver, become `root` and execute an `rpm` command:

```
shell> su root
shell> rpm -e mysql-connector-odbc
```

22.1.3.3. Installing Connector/ODBC on Mac OS X

Mac OS X is based on the FreeBSD operating system, and you can normally use the MySQL network port for connecting to MySQL servers on other hosts. Installing the Connector/ODBC driver lets you connect to MySQL databases on any platform through the ODBC interface. If your application requires an ODBC interface, install the Connector/ODBC driver. Applications that require or can use ODBC (and therefore the Connector/ODBC driver) include ColdFusion, Filemaker Pro, 4th Dimension and many other applications.

Mac OS X includes its own ODBC manager, based on the `iODBC` manager. Mac OS X includes an administration tool that provides easier administration of ODBC drivers and configuration, updating the underlying `iODBC` configuration files.

For Connector/ODBC 5.x, the package is provided as a compressed tar archive that you manually install.

In either case, the driver is designed to work with the `iODBC` driver manager included with Mac OS X.

To install Connector/ODBC 5.x:

1. Download the installation file for the appropriate level of OS X (10.5 or higher).
2. Extract the archive:

```
shell> tar xvzf mysql-connector-odbc-5.x.y-osx10.z-x86-(32/64)bit.tar.gz
```

3. The directory created contains the subdirectories, `lib`, `bin`, and `test`. Copy the contents of `lib` and `bin` to a suitable location such as `/usr/local`:

```
shell> sudo cp bin/* /usr/local/bin
shell> sudo cp lib/* /usr/local/lib
```

4. Finally, register the driver with `iODBC` using the `myodbc3i` tool you just installed:

```
shell> myodbc3i -a -d -t"MySQL ODBC 3.51 Driver;Driver=/usr/local/lib/libmyodbc3.so;Setup=/usr/local/li
```

To verify the installed drivers, either use the ODBC Administrator application or the `myodbc3i` utility:

```
shell> myodbc3i -q -d
```

22.1.3.4. Building Connector/ODBC from a Source Distribution on Windows

You only need to build Connector/ODBC from source on Windows to modify the source or installation location. If you are unsure whether to install from source, please use the binary installation detailed in [Section 22.1.3.1, "Installing Connector/ODBC on Windows"](#).

Building Connector/ODBC from source on Windows requires a number of different tools and packages:

- MDAC, Microsoft Data Access SDK from <http://support.microsoft.com/kb/110093>.
- Suitable C compiler, such as Microsoft Visual C++ or the C compiler included with Microsoft Visual Studio.

Microsoft Visual Studio 7 and 8 are preferred, and well-tested.

- Connector/ODBC 5.2: `cmake`.

- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because Connector/ODBC uses calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit <http://dev.mysql.com/downloads/>.

Build Steps

Set the environment variables for the Visual Studio toolchain. Visual Studio includes a batch file to set these for you, and installs a **Start** menu shortcut that opens a command prompt with these variables set.

Set `MYSQL_DIR` to the MySQL server installation path, while using the short-style file names. For example:

```
C:\> set MYSQL_DIR=C:\PROGRA~1\MySQL\MYSQLS~1.0
```

Build Connector/ODBC using the `cmake` command-line tool by executing the following from the source root directory (in a command prompt window):

```
C:\> cmake -G "Visual Studio 8 2005"
```

This produces a project file that you can open with Visual Studio, or build from the command line with either of the following commands:

```
C:\> devenv.com MySQL_Connector_ODBC.sln /build Release
C:\> devenv.com MySQL_Connector_ODBC.sln /build RelWithDebInfo
```

To compile a debug build, set the `cmake` build type so that the correct versions of the MySQL client libraries are used:

```
C:\> cmake -G "Visual Studio 8 2005" -DCMAKE_BUILD_TYPE=Debug
C:\> devenv.com MySQL_Connector_ODBC.sln /build Debug
```

Upon completion, the executables are in the `bin/` and `lib/` subdirectories.

See [Section 22.1.3.1.2, "Installing the Windows Connector/ODBC Driver using the Zipped DLL Package"](#) for the `copy` commands to complete the installation.

22.1.3.5. Building Connector/ODBC from a Source Distribution on Unix



Note

Connector/5.2 uses `cmake` for build, rather than `autoconf` and its `configure` command.

You need the following tools to build MySQL from source on Unix:

- A working ANSI C++ compiler. GCC 4.2.1 or later, Sun Studio 10 or later, Visual Studio 2008 or later, and many current vendor-supplied compilers are known to work.
- `cmake`.
- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because Connector/ODBC uses calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit <http://dev.mysql.com/downloads/>.

If you have built your own MySQL server or client libraries from source using the GNU autotools, use the `--enable-thread-safe-client` option to `configure` when the libraries were built. No special option is needed if you configure with `cmake`.

Also, ensure that the `libmysqlclient` library was built and installed as a shared library.

- A compatible ODBC manager must be installed. Connector/ODBC is known to work with the `iODBC` and `unixODBC` managers. See [Section 22.1.2.1.2, “ODBC Driver Managers”](#) for more information.
- If you are using a character set that is not compiled into the MySQL client library, install the MySQL character definitions from the `charsets` directory into `SHAREDIR` (by default, `/usr/local/mysql/share/mysql/charsets`). These should be in place if you have installed the MySQL server on the same machine. See [Section 10.1, “Character Set Support”](#) for more information on character set support.

Once you have all the required files, unpack the source files to a separate directory, then run `configure` and build the library using `make`.

Typical `cmake` Options

`iODBC` is the default ODBC library used by Connector/ODBC. Alternatively, `unixODBC` may be used by passing in the appropriate option to `cmake`. For example:

```
shell> cmake -G "Unix Makefiles" -DWITH_UNIXODBC=1
```

When you run `cmake`, you might add options to the command line. Here are some examples:

- `-DODBC_INCLUDES=dir_name [153]`: Used when the ODBC include directory is not found within the system `$PATH`.
- `-DODBC_LIB_DIR=dir_name [153]`: Used when the ODBC library directory is not found within the system `$PATH`.
- `-DWITH_UNIXODBC=1 [153]`: Enables `unixODBC` support.

Additional `configure` Options

There are a number of other options that you need, or want, to set when configuring the Connector/ODBC driver before it is built.

- To link the driver with MySQL thread safe client libraries `libmysqlclient_r.so` or `libmysqlclient_r.a`, specify the following `configure` option:

```
--enable-thread-safe
```

and can be disabled (default) using

```
--disable-thread-safe
```

This option enables the building of the driver thread-safe library `libmyodbc3_r.so` from by linking with MySQL thread-safe client library `libmysqlclient_r.so`. (The extensions are OS-dependent.)

If the compilation with the thread-safe option fails, it may be because the correct thread-libraries on the system could not be located. Set the value of `LIBS` to point to the correct thread library for your system.

```
LIBS="-lpthread" ./configure ..
```

- You can enable or disable the shared and static versions of Connector/ODBC using these options:

```
--enable-shared[=yes|no]
--disable-shared
--enable-static[=yes|no]
--disable-static
```

- By default, all the binary distributions are built as non-debug versions (configured with `--without-debug`).

To enable debugging information, build the driver from a source distribution with the proper configuration option to enable debugging support. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

- This option is available only for source trees that have been obtained from the Subversion repository. This option does not apply to the packaged source distributions.

By default, the driver is built with the `--without-docs` option. If you would like the documentation to be built, then execute `configure` with:

```
--with-docs
```

Build Steps for Unix

To build the driver libraries, execute `make`:

```
shell> make
```

If any errors occur, correct them and continue the build process. If you are not able to build, then send a detailed email to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com) for further assistance.

Building Shared Libraries

On most platforms, MySQL does not build or support `.so` (shared) client libraries by default. This is based on our experience of problems when building shared libraries.

In cases like this, you have to download the MySQL distribution and configure it with these options:

```
--without-server --enable-shared
```

To build shared driver libraries, specify the `--enable-shared` option for `configure`. By default, `configure` does not enable this option.

If you have configured with the `--disable-shared` option, you can build the `.so` file from the static libraries using the following commands:

```
shell> cd mysql-connector-odbc-3.51.01
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
    $CC -bundle -flat_namespace -undefined error \
    -o .libs/libmyodbc3-3.51.01.so \
    catalog.o connect.o cursor.o dll.o error.o execute.o \
    handle.o info.o misc.o myodbc3.o options.o prepare.o \
    results.o transact.o utility.o \
    -L/usr/local/mysql/lib/mysql/ \
    -L/usr/local/iodbc/lib/ \
    -lz -lc -lmysqlclient -liodbcinst
```

Make sure to change `-liodbcinst` to `-lodbcinst` if you are using `unixODBC` instead of `iODBC`, and configure the library paths accordingly.

This builds and places the `libmyodbc3-3.51.01.so` file in the `.libs` directory. Copy this file to the Connector/ODBC library installation directory (`/usr/local/lib`, or the `lib` directory under the installation directory that you supplied with the `--prefix` option).

```
shell> cd .libs
shell> cp libmyodbc3-3.51.01.so /usr/local/lib
shell> cd /usr/local/lib
```

```
shell> ln -s libmyodbc3-3.51.01.so libmyodbc3.so
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
  $CC -bundle -flat_namespace -undefined error
  -o .libs/libmyodbc3_r-3.51.01.so
  catalog.o connect.o cursor.o dll.o error.o execute.o
  handle.o info.o misc.o myodbc3.o options.o prepare.o
  results.o transact.o utility.o
  -L/usr/local/mysql/lib/mysql/
  -L/usr/local/iodbc/lib/
  -lz -lc -lmysqlclient_r -liodbcinst
```

Installing Driver Libraries

To install the driver libraries, execute the following command:

```
shell> make install
```

That command installs one of the following sets of libraries:

For Connector/ODBC 3.51:

- `libmyodbc3.so`
- `libmyodbc3-3.51.01.so`, where 3.51.01 is the version of the driver
- `libmyodbc3.a`

For thread-safe Connector/ODBC 3.51:

- `libmyodbc3_r.so`
- `libmyodbc3-3_r.51.01.so`
- `libmyodbc3_r.a`

For more information on build process, refer to the [INSTALL](#) file that comes with the source distribution. Note that if you are trying to use the `make` from Sun, you may end up with errors. On the other hand, GNU `gmake` should work fine on all platforms.

Testing Connector/ODBC on Unix

To run the basic samples provided in the distribution with the libraries that you built, use the following command:

```
shell> make test
```

Before running the tests, create the DSN 'myodbc3' in `odbc.ini` and set the environment variable `ODBCINI` to the correct `odbc.ini` file; and MySQL server is running. You can find a sample `odbc.ini` with the driver distribution.

You can even modify the `samples/run-samples` script to pass the desired DSN, UID, and PASSWORD values as the command-line arguments to each sample.

22.1.3.6. Building Connector/ODBC from Source on Mac OS X

To build the driver on Mac OS X (Darwin), make use of the following `configure` example:

```
shell> ./configure --prefix=/usr/local
```

```
--with-unixODBC=/usr/local
--with-mysql-path=/usr/local/mysql
--disable-shared
--enable-gui=no
--host=powerpc-apple
```

The command assumes that the `unixODBC` and MySQL are installed in the default locations. If not, configure accordingly.

On Mac OS X, `--enable-shared` builds `.dylib` files by default. You can build `.so` files like this:

```
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
    $CC -bundle -flat_namespace -undefined error
    -o .libs/libmyodbc3-3.51.01.so *.o
    -L/usr/local/mysql/lib/
    -L/usr/local/iodbc/lib
    -liodbcinst -lmysqlclient -lz -lc
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
    $CC -bundle -flat_namespace -undefined error
    -o .libs/libmyodbc3-3.51.01.so *.o
    -L/usr/local/mysql/lib/
    -L/usr/local/iodbc/lib
    -liodbcinst -lmysqlclienti_r -lz -lc -lpthread
```

Make sure to change the `-liodbcinst` to `-lodbcinst` in case of using `unixODBC` instead of `iODBC` and configure the libraries path accordingly.

In Apple's version of GCC, both `cc` and `gcc` are actually symbolic links to `gcc3`.

Copy this library to the `$prefix/lib` directory and symlink to `libmyodbc3.so`.

You can cross-check the output shared-library properties using this command:

```
shell> otool -LD .libs/libmyodbc3-3.51.01.so
```

22.1.3.7. Installing Connector/ODBC from the Development Source Tree



Caution

Read this section only if you are interested in helping us test our new code. To just get MySQL Connector/ODBC up and running on your system, use a standard release distribution.

To obtain the most recent development source tree, first download and install Bazaar from the [Bazaar VCS Web site](#). Bazaar is supported by any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows or Mac OS X host. Instructions for downloading and installing Bazaar on the different platforms are available on the Bazaar Web site.

Building from the source trees requires the following tools:

- `autoconf` 2.52 (or newer).
- `automake` 1.4 (or newer).
- `libtool` 1.4 (or newer).
- `m4`.

The most recent development source tree is available from our public Subversion trees at <http://dev.mysql.com/tech-resources/sources.html>.

To check out the Connector/ODBC sources, change to the directory where you want the copy of the Connector/ODBC tree to be stored, then use the following command:

```
shell> bzip branch lp:myodbc
```

You should now have a copy of the entire Connector/ODBC source tree in the directory `connector-odbc3`. To build from this source tree on Unix or Linux, follow these steps:

```
shell> cd myodbc
shell> aclocal
shell> autoheader
shell> libtoolize -c -f
shell> autoconf
shell> automake;
shell> ./configure # Add your favorite options here
shell> make
```

When the build is done, run `make install` to install the Connector/ODBC driver on your system.

If you have gotten to the `make` stage and the distribution does not compile, please report it to myodbc@lists.mysql.com.

On Windows, make use of Windows Makefiles `WIN-Makefile` and `WIN-Makefile_debug` in building the driver. For more information, see [Section 22.1.3.4, "Building Connector/ODBC from a Source Distribution on Windows"](#).

After the initial checkout operation to get the source tree, run `bzip pull` periodically to update your source according to the latest version.

22.1.4. Configuring Connector/ODBC

Before you connect to a MySQL database using the Connector/ODBC driver, you configure an ODBC Data Source Name (DSN). The DSN associates the various configuration parameters required to communicate with a database to a specific name. You use the DSN in an application to communicate with the database, rather than specifying individual parameters within the application itself. DSN information can be user-specific, system-specific, or provided in a special file. ODBC data source names are configured in different ways, depending on your platform and ODBC driver.

22.1.4.1. Overview of Connector/ODBC Data Source Names

A Data Source Name associates the configuration parameters for communicating with a specific database. Generally, a DSN consists of the following parameters:

- Name
- Host Name
- Database Name
- Login
- Password

In addition, different ODBC drivers, including Connector/ODBC, may accept additional driver-specific options and parameters.

There are three types of DSN:

- A *System DSN* is a global DSN definition that is available to any user and application on a particular system. A System DSN can normally only be configured by a systems administrator, or by a user who has specific permissions that let them create System DSNs.
- A *User DSN* is specific to an individual user, and can be used to store database connectivity information that the user regularly uses.
- A *File DSN* uses a simple file to define the DSN configuration. File DSNs can be shared between users and machines and are therefore more practical when installing or deploying DSN information as part of an application across many machines.

DSN information is stored in different locations depending on your platform and environment.

22.1.4.2. Connector/ODBC Connection Parameters

You can specify the parameters in the following tables for Connector/ODBC when configuring a DSN:

- [Table 22.3, “Connector/ODBC DSN Configuration Options”](#)
- [Table 22.4, “Connector/ODBC Option Flags”](#)

Users on Windows can use the Options and Advanced panels when configuring a DSN to set these parameters; see the table for information on which options relate to which fields and check boxes. On Unix and Mac OS X, use the parameter name and value as the keyword/value pair in the DSN configuration. Alternatively, you can set these parameters within the `InConnectionString` argument in the `SQLDriverConnect()` call.

Table 22.3. Connector/ODBC DSN Configuration Options

Parameter	Default Value	Comment
<code>user</code>	ODBC	The user name used to connect to MySQL.
<code>uid</code>	ODBC	Synonymous with <code>user</code> . Added in 3.51.16.
<code>server</code>	<code>localhost</code>	The host name of the MySQL server.
<code>database</code>		The default database.
<code>option</code>	0	Options that specify how Connector/ODBC works. See Table 22.4, “Connector/ODBC Option Flags” and Table 22.5, “Recommended Connector/ODBC option Values for Different Configurations” .
<code>port</code>	3306	The TCP/IP port to use if <code>server</code> is not <code>localhost</code> .
<code>initstmt</code>		Initial statement. A statement to execute when connecting to MySQL. In version 3.51 the parameter is called <code>stmt</code> . Note, the driver supports the initial statement being executed only at the time of the initial connection.
<code>password</code>		The password for the <code>user</code> account on <code>server</code> .
<code>pwd</code>		Synonymous with <code>password</code> . Added in 3.51.16.
<code>socket</code> [553]		The Unix socket file or Windows named pipe to connect to if <code>server</code> is <code>localhost</code> .
<code>sslca</code>		The path to a file with a list of trust SSL CAs. Added in 3.51.16.
<code>sslcapath</code>		The path to a directory that contains trusted SSL CA certificates in PEM format. Added in 3.51.16.
<code>sslcert</code>		The name of the SSL certificate file to use for establishing a secure connection. Added in 3.51.16.

Parameter	Default Value	Comment
<code>sslcipher</code>		A list of permissible ciphers to use for SSL encryption. The cipher list has the same format as the <code>openssl ciphers</code> command. Added in 3.51.16.
<code>sslkey</code>		The name of the SSL key file to use for establishing a secure connection. Added in 3.51.16.
<code>charset</code>		The character set to use for the connection. Added in 3.51.17.
<code>sslverify</code>		If set to 1, the SSL certificate will be verified when used with the MySQL connection. If not set, then the default behavior is to ignore SSL certificate verification.
<code>readtimeout</code>		The timeout in seconds for attempts to read from the server. Each attempt uses this timeout value and there are retries if necessary, so the total effective timeout value is three times the option value. You can set the value so that a lost connection can be detected earlier than the TCP/IP <code>Close_Wait_Timeout</code> value of 10 minutes. This option works only for TCP/IP connections, and only for Windows prior to MySQL 5.1.12. Corresponds to the <code>MYSQL_OPT_READ_TIMEOUT</code> option of the MySQL Client Library. This option was added in Connector/ODBC 3.51.27.
<code>writetimeout</code>		The timeout in seconds for attempts to write to the server. Each attempt uses this timeout value and there are <code>net_retry_count</code> retries if necessary, so the total effective timeout value is <code>net_retry_count</code> times the option value. This option works only for TCP/IP connections, and only for Windows prior to MySQL 5.1.12. Corresponds to the <code>MYSQL_OPT_WRITE_TIMEOUT</code> option of the MySQL Client Library. This option was added in Connector/ODBC 3.51.27.
<code>interactive</code>		Enables the <code>CLIENT_INTERACTIVE</code> connection option of <code>mysql_real_connect</code> .
<code>prefetch</code>	0	When set to a non-zero value <i>N</i> , causes all queries in the connection to return <i>N</i> rows at a time rather than the entire result set. Useful for queries against very large tables where it is not practical to retrieve the whole result set at once. You can scroll through the result set, <i>N</i> records at a time. This option works only with forward-only cursors. It does not work when the <code>FLAG_MULTI_STATEMENTS</code> flag is set. It can be used in combination with the <code>FLAG_NO_CACHE</code> option. Its behavior in ADO applications is undefined: the prefetching might or might not occur.
<code>no_ssps</code>	0	In Connector/ODBC 5.2, by default, server-side prepared statements are used. When this option is set to a non-zero value, prepared statements are emulated on the client side, which is the same behavior as in 5.1 and 3.51. Added in 5.2.

**Note**

The SSL configuration parameters can also be automatically loaded from a `my.ini` or `my.cnf` file. See [Section 4.2.3.3, “Using Option Files”](#).

The `option` argument is used to tell Connector/ODBC that the client is not 100% ODBC-compliant. On Windows, you normally select options by toggling the check boxes in the connection screen, but you can

also select them in the `option` argument. The following options are listed in the order in which they appear in the Connector/ODBC connect screen.

Table 22.4. Connector/ODBC Option Flags

Flag Name	GUI Option	Constant Value	Description
<code>FLAG_FIELD_LENGTH</code>	Do not Optimize Column Width		The client cannot handle that Connector/ODBC returns the real width of a column. This option was removed in 3.51.18.
<code>FLAG_FOUND_ROWS</code>	Return Matching Rows	2	The client cannot handle that MySQL returns the true value of affected rows. If this flag is set, MySQL returns “found rows” instead. You must have MySQL 3.21.14 or newer for this to work.
<code>FLAG_DEBUG</code>	Trace Driver Calls To myodbc.log		Make a debug log in <code>C:\myodbc.log</code> on Windows, or <code>/tmp/myodbc.log</code> on Unix variants. This option was removed in Connector/ODBC 3.51.18.
<code>FLAG_BIG_PACKETS</code>	Allow Big Results	8	Do not set any packet limit for results and bind parameters. Without this option, parameter binding will be truncated to 255 characters.
<code>FLAG_NO_PROMPT</code>	Do not Prompt Upon Connect	16	Do not prompt for questions even if driver would like to prompt.
<code>FLAG_DYNAMIC_CURSOR</code>	Enable Dynamic Cursor	32	Enable or disable the dynamic cursor support.
<code>FLAG_NO_SCHEMA</code>	Ignore # in Table Name	64	Ignore use of database name in <code>db_name.tbl_name.col_name</code> .
<code>FLAG_NO_DEFAULT_CURSOR</code>	User Manager Cursors	128	Force use of ODBC manager cursors (experimental).
<code>FLAG_NO_LOCALE</code>	Do not Use Set Locale	256	Disable the use of extended fetch (experimental).
<code>FLAG_PAD_SPACE</code>	Pad Char To Full Length	512	Pad <code>CHAR</code> columns to full column length.
<code>FLAG_FULL_COLUMN_NAMES</code>	Return Table Names for SQLDescribeCol	1024	<code>SQLDescribeCol()</code> returns fully qualified column names.
<code>FLAG_COMPRESSED_PROTO</code>	Use Compressed Protocol	2048	Use the compressed client/server protocol.
<code>FLAG_IGNORE_SPACE</code>	Ignore Space After Function Names	4096	Tell server to ignore space after function name and before “(” (needed by PowerBuilder). This makes all function names keywords.

Configuring Connector/ODBC

Flag Name	GUI Option	Constant Value	Description
FLAG_NAMED_PIPE	Force Use of Named Pipes	8192	Connect with named pipes to a <code>mysqld</code> server running on NT.
FLAG_NO_BIGINT	Change BIGINT Columns to Int	16384	Change <code>BIGINT</code> columns to <code>INT</code> columns (some applications cannot handle <code>BIGINT</code>).
FLAG_NO_CATALOG	No Catalog	32768	Forces results from the catalog functions, such as <code>SQLTables</code> , to always return <code>NULL</code> and the driver to report that catalogs are not supported.
FLAG_USE_MYCNF	Read Options From <code>my.cnf</code>	65536	Read parameters from the <code>[client]</code> and <code>[odbc]</code> groups from <code>my.cnf</code> .
FLAG_SAFE	Safe	131072	Add some extra safety checks.
FLAG_NO_TRANSACTIONS	Disable <code>transactions</code>	262144	Disable transactions.
FLAG_LOG_QUERY	Save queries to <code>myodbc.sql</code>	524288	Enable query logging to <code>c:\myodbc.sql(/tmp/myodbc.sql)</code> file. (Enabled only in debug mode.)
FLAG_NO_CACHE	Do not Cache Result (forward only cursors)	1048576	Do not cache the results locally in the driver, instead read from server (<code>mysql_use_result()</code>). This works only for forward-only cursors. This option is very important in dealing with large tables when you do not want the driver to cache the entire result set.
FLAG_FORWARD_CURSOR	Force Use Of Forward Only Cursors	2097152	Force the use of <code>Forward-only</code> cursor type. In case of applications setting the default static/dynamic cursor type, and one wants the driver to use noncache result sets, then this option ensures the forward-only cursor behavior.
FLAG_AUTO_RECONNECT	Enable auto-reconnect.	4194304	Enables auto-reconnection functionality. Do not use this option with <code>transactions</code> , since an auto-reconnection during a incomplete transaction may cause corruption. Note that an auto-reconnected connection will not inherit the same settings and environment as the original. This option was added in Connector/ODBC 3.51.13.

Configuring Connector/ODBC

Flag Name	GUI Option	Constant Value	Description
FLAG_AUTO_IS_NULL	Flag Auto Is Null	8388608	<p>When FLAG_AUTO_IS_NULL is set, the driver does not change the default value of sql_auto_is_null [554], leaving it at 1, so you get the MySQL default, not the SQL standard behavior.</p> <p>When FLAG_AUTO_IS_NULL is not set, the driver changes the default value of SQL_AUTO_IS_NULL to 0 after connecting, so you get the SQL standard, not the MySQL default behavior.</p> <p>Thus, omitting the flag disables the compatibility option and forces SQL standard behavior.</p> <p>See IS NULL [1091]. This option was added in Connector/ODBC 3.51.13.</p>
FLAG_ZERO_DATE_TO_MIN	Return SQL_NULL_DATA for zero date	16777216	<p>Translates zero dates (xxxx-00-00) into the minimum date values supported by ODBC, xxxx-01-01. This resolves an issue where some statements will not work because the date returned and the minimum ODBC date value are incompatible. This option was added in Connector/ODBC 3.51.17.</p>
FLAG_MIN_DATE_TO_ZERO	Bind minimal date as zero date	33554432	<p>Translates the minimum ODBC date value (xxxx-01-01) to the zero date format supported by MySQL (xxxx-00-00). This resolves an issue where some statements will not work because the date returned and the minimum ODBC date value are incompatible. This option was added in Connector/ODBC 3.51.17.</p>
FLAG_MULTI_STATEMENTS	Allow multiple statements	67108864	<p>Enables support for batched statements. This option was added in Connector/ODBC 3.51.18.</p>
FLAG_COLUMN_SIZE_S32	Limit column size to 32-bit value	134217728	<p>Limits the column size to a signed 32-bit value to prevent</p>

Flag Name	GUI Option	Constant Value	Description
			problems with larger column sizes in applications that do not support them. This option is automatically enabled when working with ADO applications. This option was added in Connector/ODBC 3.51.22.
<code>FLAG_NO_BINARY_RESULT</code>	Always handle binary function results as character data	268435456	When set this option disables charset 63 for columns with an empty <code>org_table</code> . This option was added in Connector/ODBC 3.51.26.
<code>FLAG_DFLT_BIGINT_BIND_STR</code>		536870912	Causes <code>BIGINT</code> parameters to be bound as strings. Microsoft Access treats <code>BIGINT</code> as a string on linked tables. The value is read correctly, but bound as a string. This option is used automatically if the driver is used by Microsoft Access.
<code>FLAG_NO_INFORMATION_SCHEMA</code>		1073741824	Tells catalog functions not to use <code>INFORMATION_SCHEMA</code> , but rather use legacy algorithms. The trade-off here is usually speed for information quality. Using <code>INFORMATION_SCHEMA</code> is often slow, but the information obtained is more complete.

To select multiple options, add together their values, using the numbers from the `Constant Value` column in the table.



Note

In MySQL Connector/ODBC 5.2, you can use the flag name directly as a parameter in the connection string, by specifying the flag name without the `FLAG_` prefix. So, in addition to using the `options` parameter with various flags set, you can use the flags directly as parameters. For example, `FIELD_LENGTH`, `FOUND_ROWS` and `DEBUG` could all be used as parameters.

The following table shows some recommended `option` values for various configurations:

Table 22.5. Recommended Connector/ODBC option Values for Different Configurations

Configuration	Option Value
Microsoft Access, Visual Basic	3
Driver trace generation (Debug mode)	4
Microsoft Access (with improved DELETE queries)	35
Large tables with too many rows	2049

Configuration	Option Value
Sybase PowerBuilder	135168
Query log generation (Debug mode)	524288
Generate driver trace as well as query log (Debug mode)	524292
Large tables with no-cache results	3145731

22.1.4.3. Configuring a Connector/ODBC DSN on Windows

The [ODBC Data Source Administrator](#) within Windows lets you create DSNs, check driver installation and configure ODBC systems such as tracing (used for debugging) and connection pooling.

Different editions and versions of Windows store the [ODBC Data Source Administrator](#) in different locations depending on the version of Windows that you are using.

To open the [ODBC Data Source Administrator](#) in Windows Server 2003:



Tip

To identify whether a DSN was created using the 32-bit or the 64-bit driver, include the driver being used within the DSN identifier. This will help you to identify the right DSN to use with applications such as Excel that are only compatible with the 32-bit driver. For example, you might add [Using32bitCOBDC](#) to the DSN identifier for the 32-bit interface and [Using64bitCOBDC](#) for those using the 64-bit Connector/ODBC driver.

1. On the [Start](#) menu, choose [Administrative Tools](#), and then click [Data Sources \(ODBC\)](#).

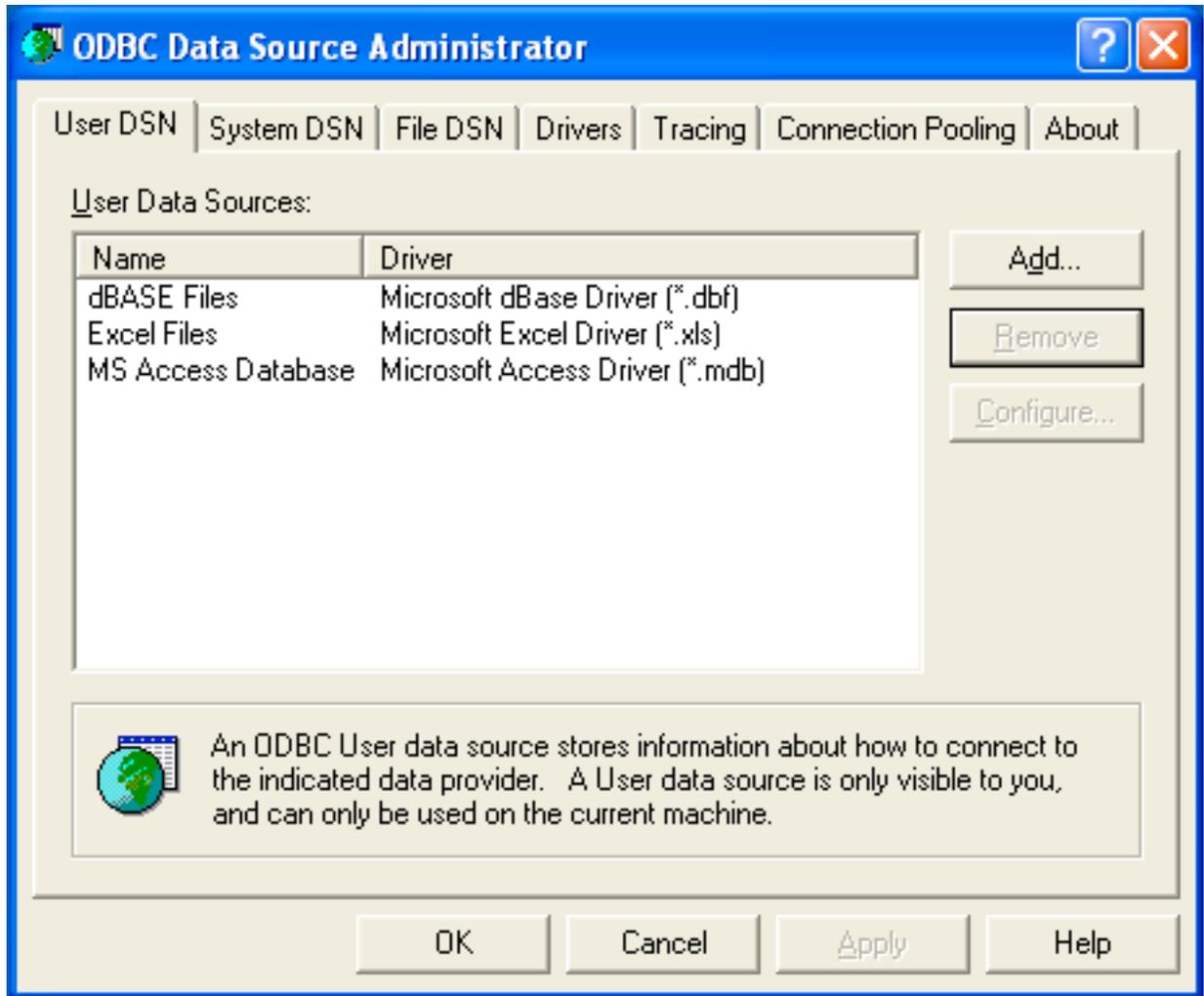
To open the [ODBC Data Source Administrator](#) in Windows 2000 Server or Windows 2000 Professional:

1. On the [Start](#) menu, choose [Settings](#), and then click [Control Panel](#).
2. In [Control Panel](#), click [Administrative Tools](#).
3. In [Administrative Tools](#), click [Data Sources \(ODBC\)](#).

To open the [ODBC Data Source Administrator](#) on Windows XP:

1. On the [Start](#) menu, click [Control Panel](#).
2. In the [Control Panel](#) when in [Category View](#) click [Performance and Maintenance](#) and then click [Administrative Tools](#).. If you are viewing the [Control Panel](#) in [Classic View](#), click [Administrative Tools](#).
3. In [Administrative Tools](#), click [Data Sources \(ODBC\)](#).

Irrespective of your Windows version, you should be presented the [ODBC Data Source Administrator](#) window:



Within Windows XP, you can add the [Administrative Tools](#) folder to your **Start** menu to make it easier to locate the ODBC Data Source Administrator. To do this:

1. Right-click the **Start** menu.
2. Select [Properties](#).
3. Click [Customize...](#)
4. Select the [Advanced](#) tab.
5. Within [Start menu items](#), within the [System Administrative Tools](#) section, select [Display on the All Programs menu](#).

Within both Windows Server 2003 and Windows XP, consider permanently adding the [ODBC Data Source Administrator](#) to your **Start** menu. To do this, locate the [Data Sources \(ODBC\)](#) icon using the methods shown, then right-click on the icon and then choose [Pin to Start Menu](#).

The interfaces for the 3.51 and 5.x versions of the Connector/ODBC driver are different, although the fields and information that you need to enter remain the same.

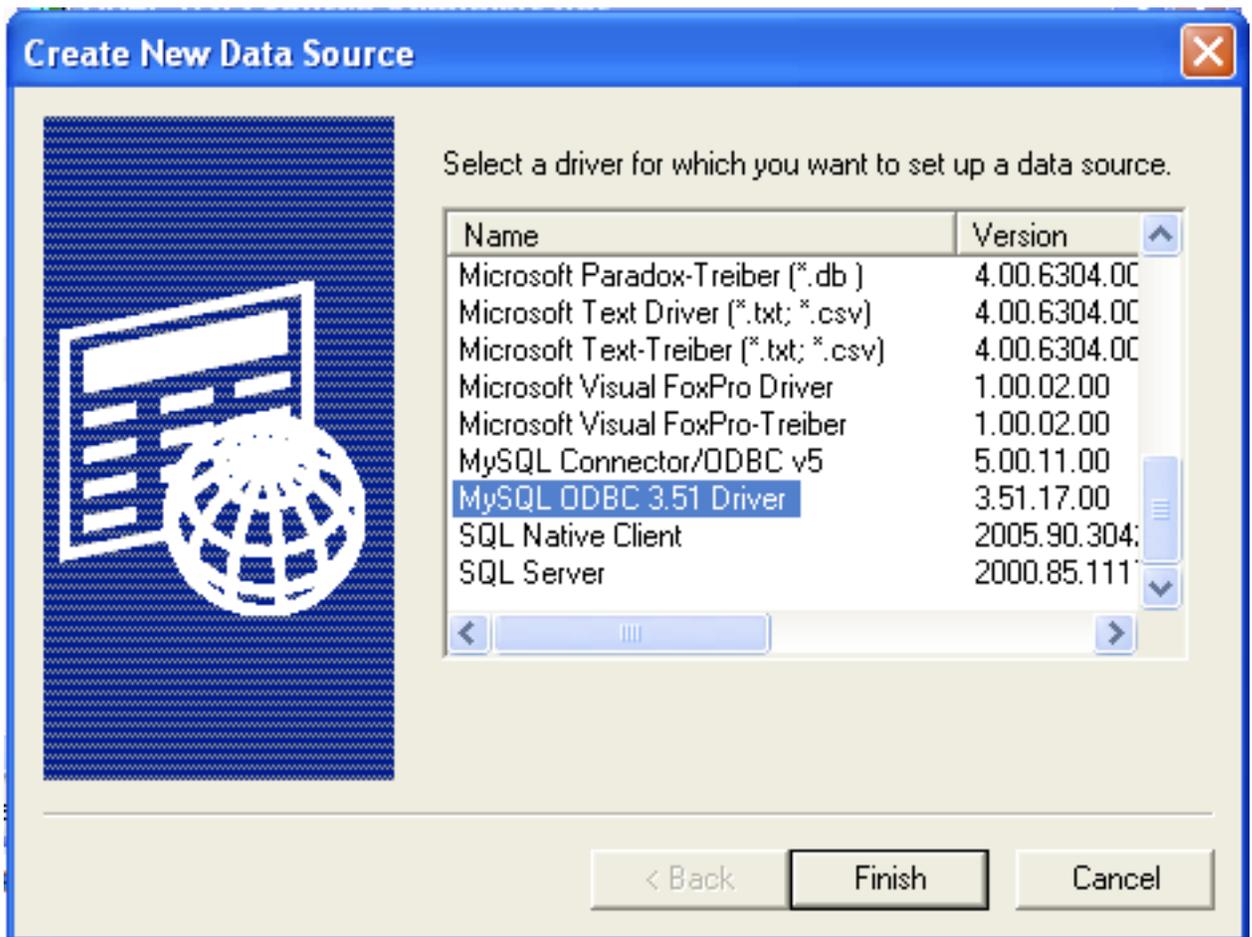
To configure a DSN using Connector/ODBC 5.2, see [Section 22.1.4.3.1, "Configuring a Connector/ODBC 5.x DSN on Windows"](#).

22.1.4.3.1. Configuring a Connector/ODBC 5.x DSN on Windows

Due to the native Unicode support within Connector/ODBC, you do not need to specify the initial character set to be used with your connection.

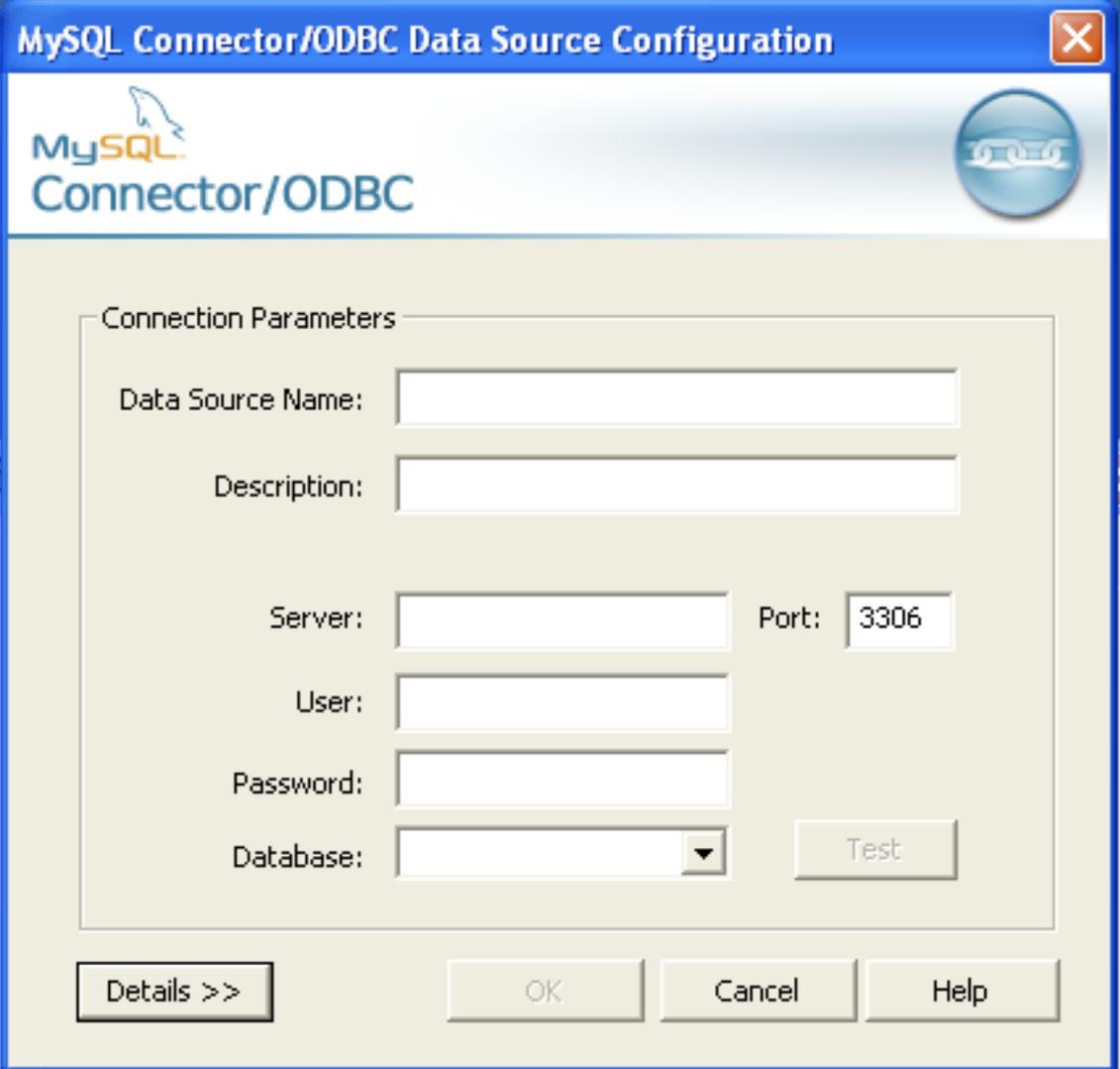
To add or configure a Connector/ODBC 5.x DSN on Windows, use either the command-line, or the [ODBC Data Source Administrator](#) GUI.

1. Open the [ODBC Data Source Administrator](#).
2. To create a System DSN (which will be available to all users), select the **System DSN** tab. To create a User DSN, which will be unique only to the current user, click the **Add...** button.
3. Select the ODBC driver for this DSN.



Select [MySQL ODBC 5.x Driver](#) for the appropriate level of Connector/ODBC, then click **Finish**.

4. You now need to configure the specific fields for the DSN you are creating through the [Connection Parameters](#) dialog.



MySQL Connector/ODBC Data Source Configuration

MySQL Connector/ODBC

Connection Parameters

Data Source Name:

Description:

Server: Port:

User:

Password:

Database:

In the **Data Source Name** box, enter the name of the data source to access. It can be any valid name that you choose.

5. In the **Description** box, enter some text to help identify the connection.
6. In the **Server** field, enter the name of the MySQL server host to access. By default, it is `localhost`.
7. In the **User** field, enter the user name to use for this connection.
8. In the **Password** field, enter the corresponding password for this connection.
9. The **Database** pop-up should automatically populate with the list of databases that the user has permissions to access.
10. To communicate over a different TCP/IP port than the default (3306), change the value of the **Port**.
11. Click **OK** to save the DSN.

To verify the connection using the parameters you have entered, click the **Test** button. If the connection could be made successfully, you will be notified with a `Success; connection was made!` dialog.

You can configure a number of options for a specific DSN by using the **Details** button.

MySQL Connector/ODBC Data Source Configuration

MySQL Connector/ODBC

Connection Parameters

Data Source Name:

Description:

Server: Port:

User:

Password:

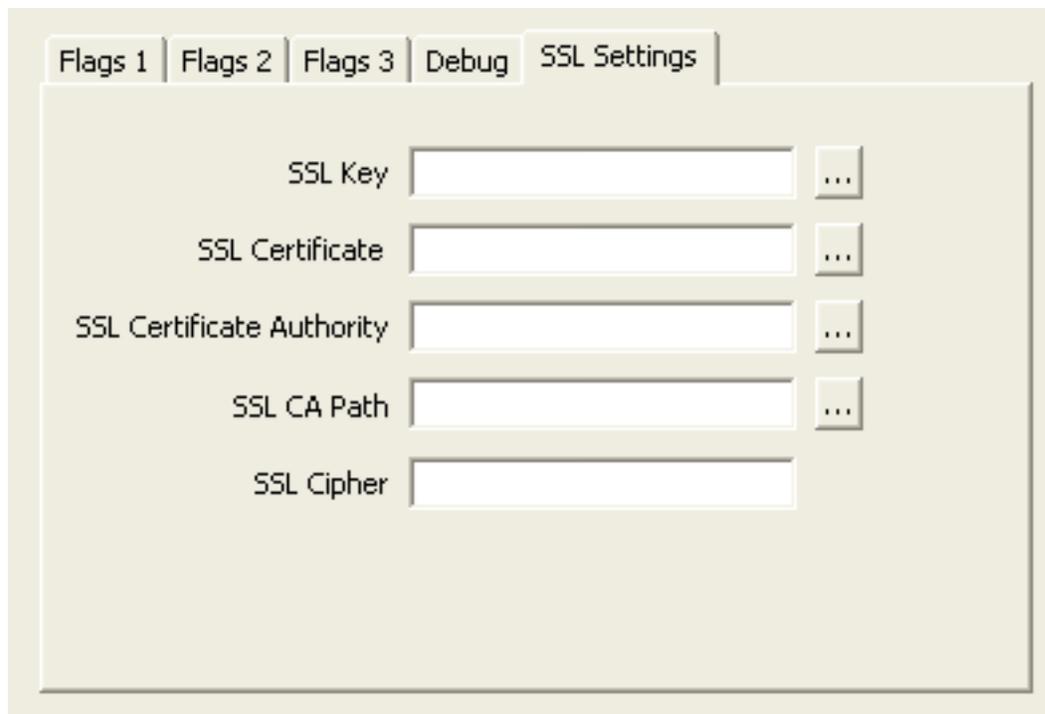
Database:

Flags 1 | Flags 2 | Flags 3 | Debug | SSL Settings

- Return matched rows instead of affected rows
- Allow big result sets
- Use compression
- Treat BIGINT columns as INT columns
- Enable safe options (see documentation)
- Enable automatic reconnect
- Enable SQL_AUTO_IS_NULL

The **Details** button opens a tabbed display where you set additional options:

- **Flags 1**, **Flags 2**, and **Flags 3** enable you to select the additional flags for the DSN connection. For more information on these flags, see [Section 22.1.4.2, “Connector/ODBC Connection Parameters”](#).
- **Debug** lets you turn on ODBC debugging to record the queries you execute through the DSN to the `myodbc.sql` file. For more information, see [Section 22.1.4.8, “Getting an ODBC Trace File”](#).
- **SSL Settings** configures the additional options required for using the Secure Sockets Layer (SSL) when communicating with MySQL server. Note that you must also enable SSL and configure the MySQL server with suitable certificates to communicate over SSL.



The Advanced tab lets you configure Connector/ODBC connection parameters. Refer to [Section 22.1.4.2, “Connector/ODBC Connection Parameters”](#), for information about the meaning of these options.

22.1.4.3.2. Configuring a Connector/ODBC 5.x DSN on Windows, Using the Command Line

Use `myodbc-installer.exe` when configuring Connector/ODBC 5.1 or later from the command-line.

Execute `myodbc-installer.exe` without arguments to view a list of available options.

22.1.4.3.3. Troubleshooting ODBC Connection Problems

This section answers Connector/ODBC connection-related questions.

- **While configuring a Connector/ODBC DSN, a `Could Not Load Translator or Setup Library` error occurs**

For more information, refer to [MS KnowledgeBase Article\(Q260558\)](#). Also, make sure you have the latest valid `ct13d32.dll` in your system directory.

- On Windows, the default `myodbc5w.dll` (Unicode) or `myodbc5a.dll` (ANSI) is compiled for optimal performance. To debug Connector/ODBC (for example, to enable tracing), instead use `myodbc5d.dll`.

To install this file, copy `myodbc5d.dll` over the installed `myodbc5w.dll` or `myodbc5a.dll` file. Make sure to revert back to the release version of the driver DLL once you are done with the debugging, because the debug version may cause performance issues.

22.1.4.4. Configuring a Connector/ODBC DSN on Mac OS X

To configure a DSN on Mac OS X, you can either use the command-line utility (`myodbc-installer` with connector/ODBC 5.1, or `myodbc3i` with connector/OCBC 3.51), edit the `odbc.ini` file within the `Library/ODBC` directory of the user, or use the ODBC Administrator GUI. If you have Mac OS X 10.2 or earlier, refer to [Section 22.1.4.5, “Configuring a Connector/ODBC DSN on Unix”](#). Select whether to create a User DSN or a System DSN. When adding a System DSN, you might need to authenticate with the system. Click the padlock and enter a user and password with administrator privileges.

For correct operation of ODBC Administrator, ensure that the `/Library/ODBC/odbc.ini` file used to set up ODBC connectivity and DSNs are writable by the `admin` group. If this file is not writable by this group, then the ODBC Administrator may fail, or may appear to work but not generate the correct entry.



Warning

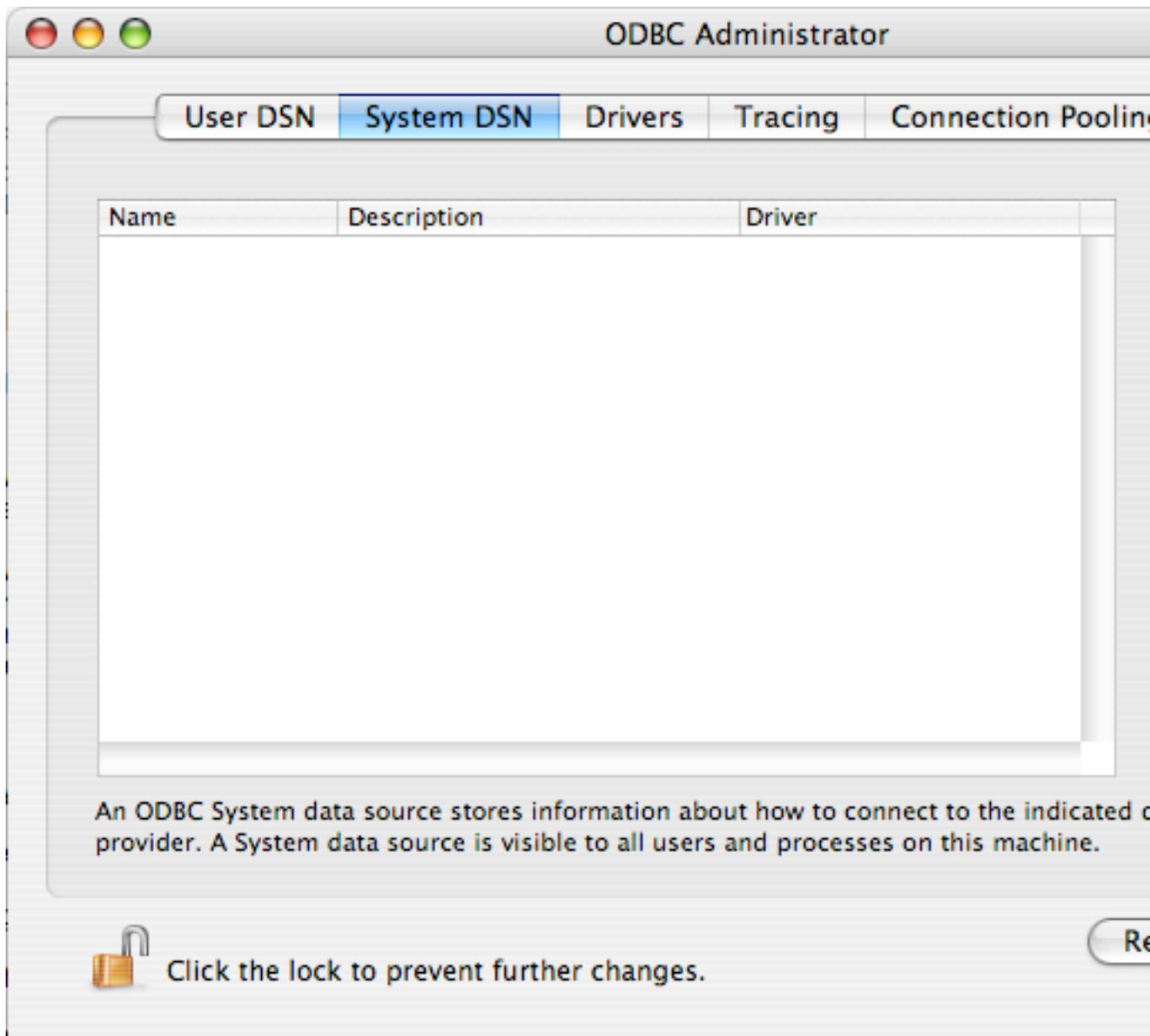
There are known issues with the OS X ODBC Administrator and Connector/ODBC that may prevent you from creating a DSN using this method. In this case, use the command line or edit the `odbc.ini` file directly. Note that existing DSNs or those that you create using the `myodbc3i` or `myodbc-installer` tool can still be checked and edited using ODBC Administrator.

To create a DSN using the `myodbc3i` utility, you need only specify the DSN type and the DSN connection string. For example:

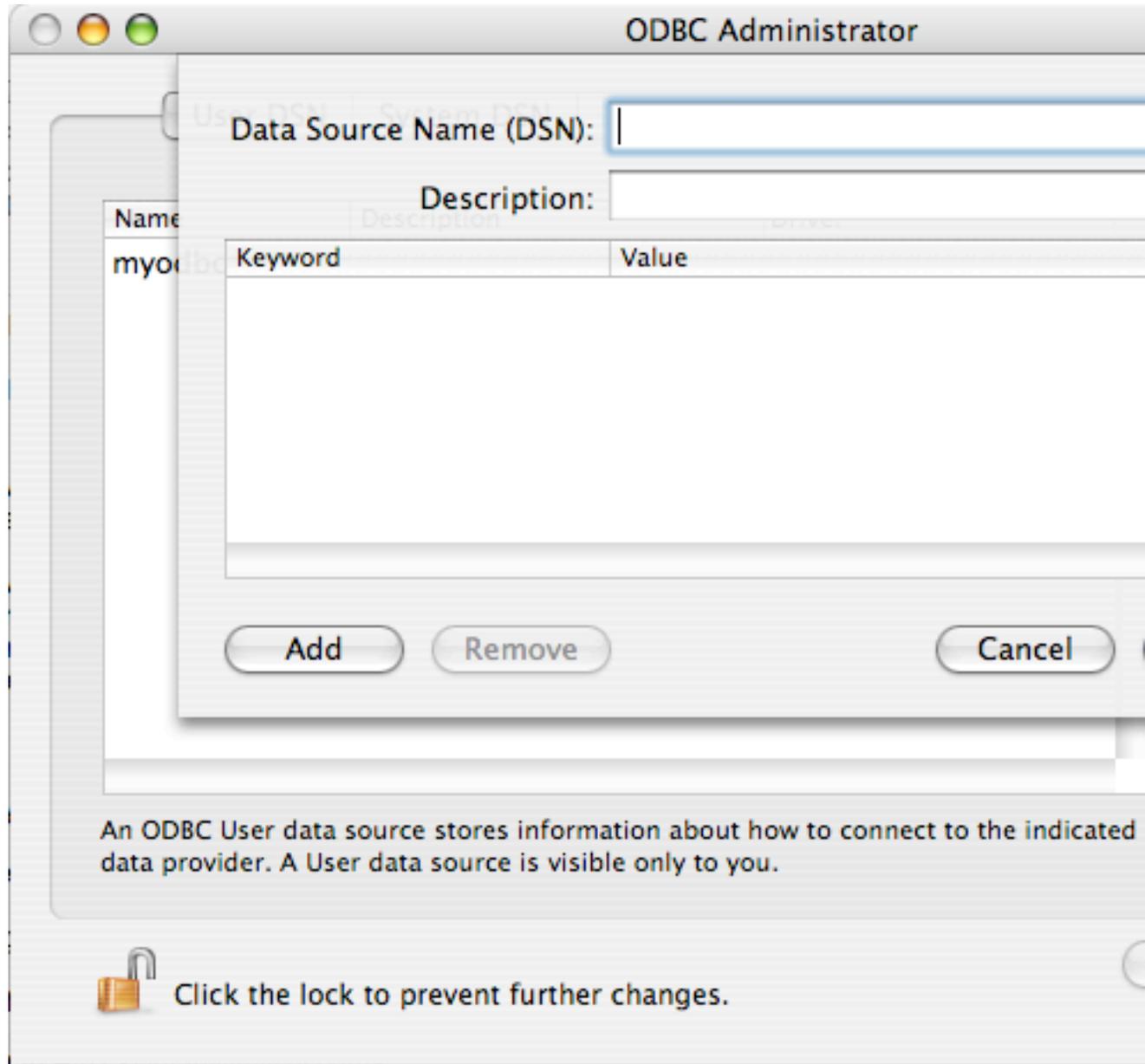
```
shell> myodbc3i -a -s -t "DSN=mydb;DRIVER=MySQL ODBC 3.51 Driver;SERVER=mysql;USER=username;PASSWORD=pass"
```

To use ODBC Administrator:

1. Open the ODBC Administrator from the `Utilities` folder in the `Applications` folder.

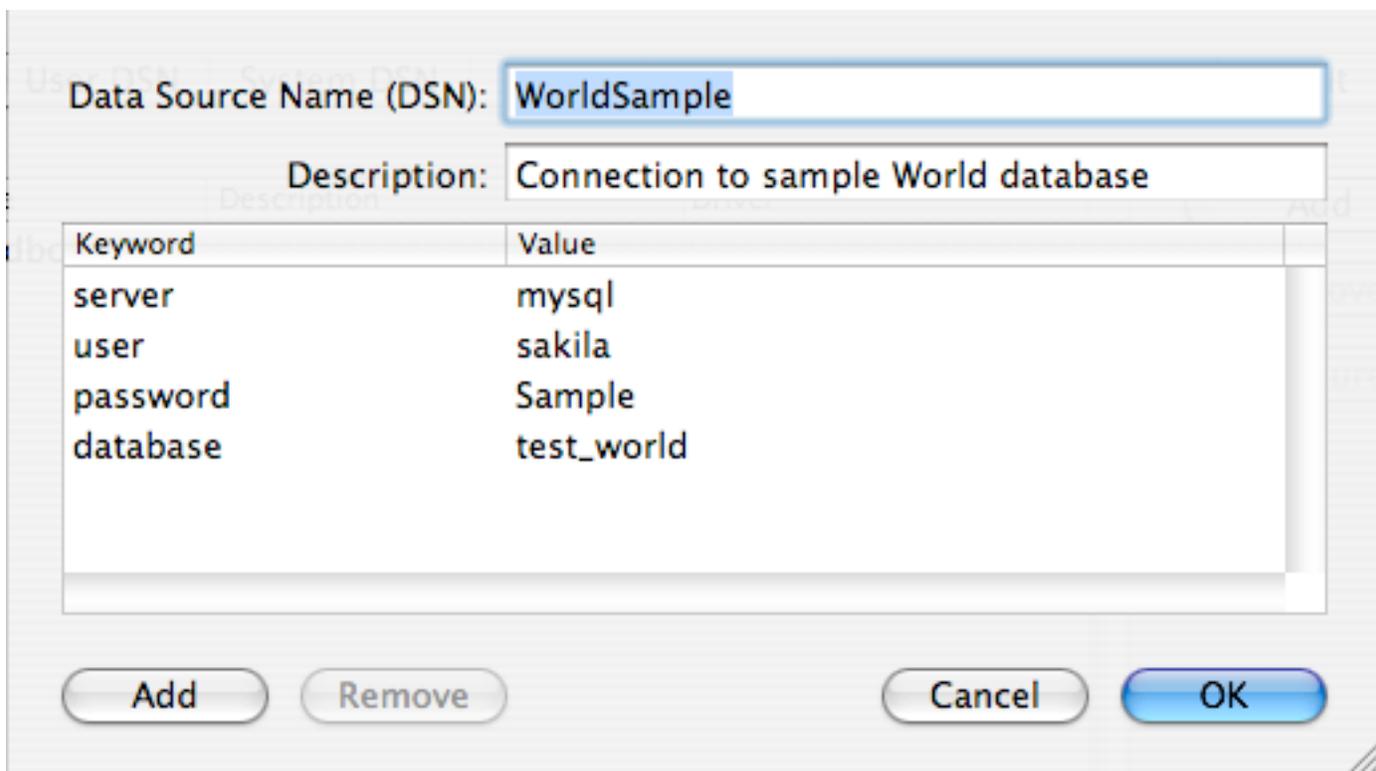


2. On the User DSN or System DSN panel, click **Add**.
3. Select the Connector/ODBC driver and click **OK**.
4. You will be presented with the **Data Source Name** dialog. Enter the **Data Source Name** and an optional **Description** for the DSN.



5. Click **Add** to add a new keyword/value pair to the panel. Configure at least four pairs to specify the [server](#), [username](#), [password](#) and [database](#) connection parameters. See [Section 22.1.4.2, "Connector/ODBC Connection Parameters"](#).
6. Click **OK** to add the DSN to the list of configured data source names.

A completed DSN configuration may look like this:



You can configure other ODBC options in your DSN by adding further keyword/value pairs and setting the corresponding values. See [Section 22.1.4.2, “Connector/ODBC Connection Parameters”](#).

22.1.4.5. Configuring a Connector/ODBC DSN on Unix

On [Unix](#), you configure DSN entries directly in the `odbc.ini` file. Here is a typical `odbc.ini` file that configures `myodbc3` as the DSN name for Connector/ODBC 3.51:

```

;
; odbc.ini configuration for Connector/ODBC and Connector/ODBC 3.51 drivers
;

[ODBC Data Sources]
myodbc3      = MyODBC 3.51 Driver DSN

[myodbc3]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = Connector/ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test
OPTION      = 3
SOCKET      =

[Default]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = Connector/ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test

```

```
OPTION      = 3
SOCKET      =
```

Refer to the [Section 22.1.4.2, “Connector/ODBC Connection Parameters”](#), for the list of connection parameters that can be supplied.



Note

If you are using `unixODBC`, you can use the following tools to set up the DSN:

- `ODBCConfig` GUI tool ([HOWTO: ODBCConfig](#))
- `odbcinst`

In some cases when using `unixODBC`, you might get this error:

```
Data source name not found and no default driver specified
```

If this happens, make sure the `ODBCINI` and `ODBCSYSINI` environment variables are pointing to the right `odbc.ini` file. For example, if your `odbc.ini` file is located in `/usr/local/etc`, set the environment variables like this:

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

22.1.4.6. Connecting Without a Predefined DSN

You can connect to the MySQL server using `SQLDriverConnect`, by specifying the `DRIVER` name field. Here are the connection strings for Connector/ODBC using DSN-Less connections:

For Connector/ODBC 3.51:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\
SERVER=localhost;\
DATABASE=test;\
USER=venu;\
PASSWORD=venu;\
OPTION=3;"
```

If your programming language converts backslash followed by whitespace to a space, it is preferable to specify the connection string as a single long string, or to use a concatenation of multiple strings that does not add spaces in between. For example:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"
"SERVER=localhost;"
"DATABASE=test;"
"USER=venu;"
"PASSWORD=venu;"
"OPTION=3;"
```

Note. On Mac OS X, you might need to specify the full path to the Connector/ODBC driver library.

Refer to [Section 22.1.4.2, “Connector/ODBC Connection Parameters”](#) for the list of connection parameters that can be supplied.

22.1.4.7. ODBC Connection Pooling

Connection pooling enables the ODBC driver to re-use existing connections to a given database from a pool of connections, instead of opening a new connection each time the database is accessed. By enabling

connection pooling you can improve the overall performance of your application by lowering the time taken to open a connection to a database in the connection pool.

For more information about connection pooling: <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169470>.

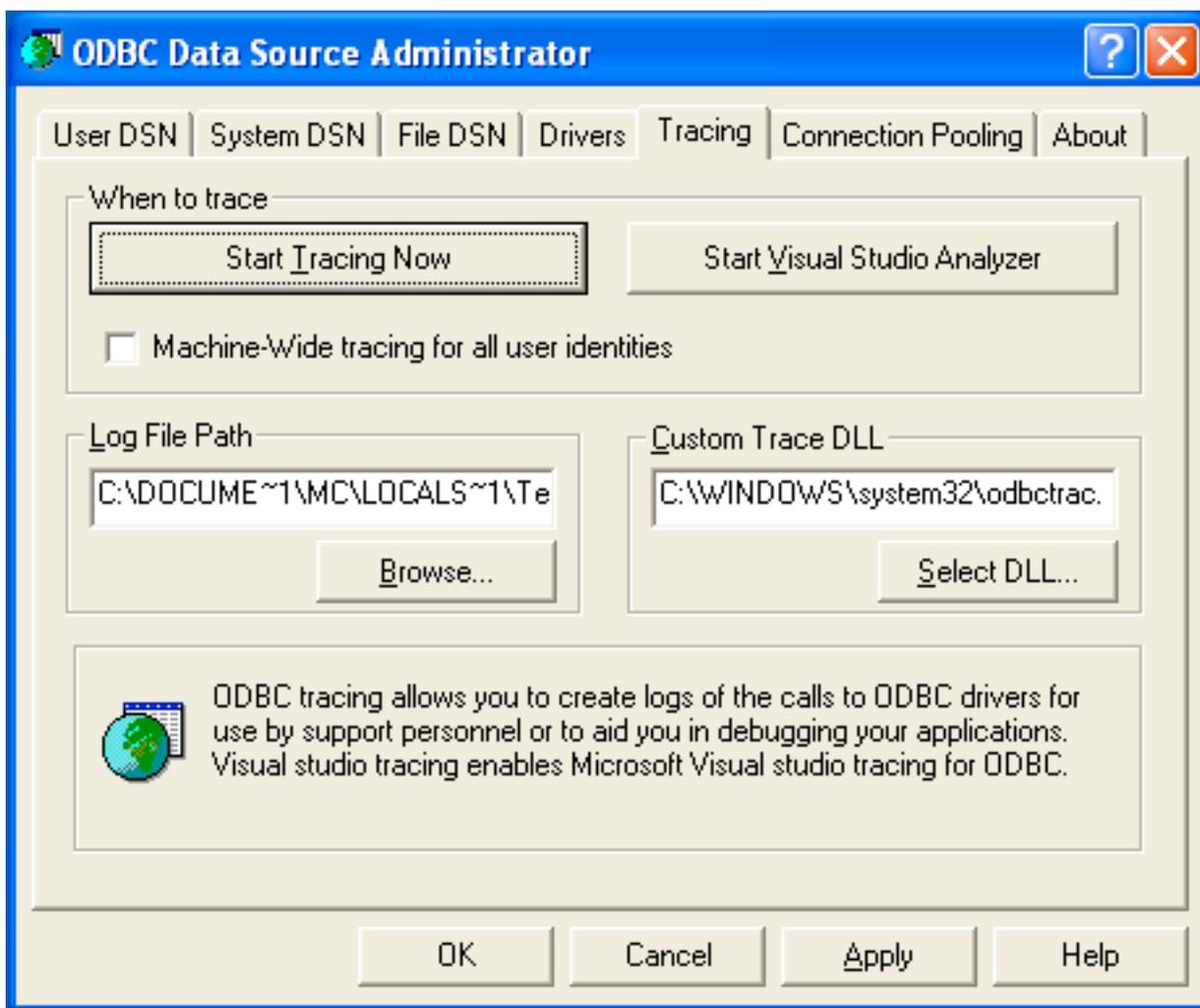
22.1.4.8. Getting an ODBC Trace File

If you encounter difficulties or problems with Connector/ODBC, start by making a log file from the [ODBC Manager](#) and Connector/ODBC. This is called *tracing*, and is enabled through the ODBC Manager. The procedure for this differs for Windows, Mac OS X and Unix.

22.1.4.8.1. Enabling ODBC Tracing on Windows

To enable the trace option on Windows:

1. The [Tracing](#) tab of the ODBC Data Source Administrator dialog box lets you configure the way ODBC function calls are traced.



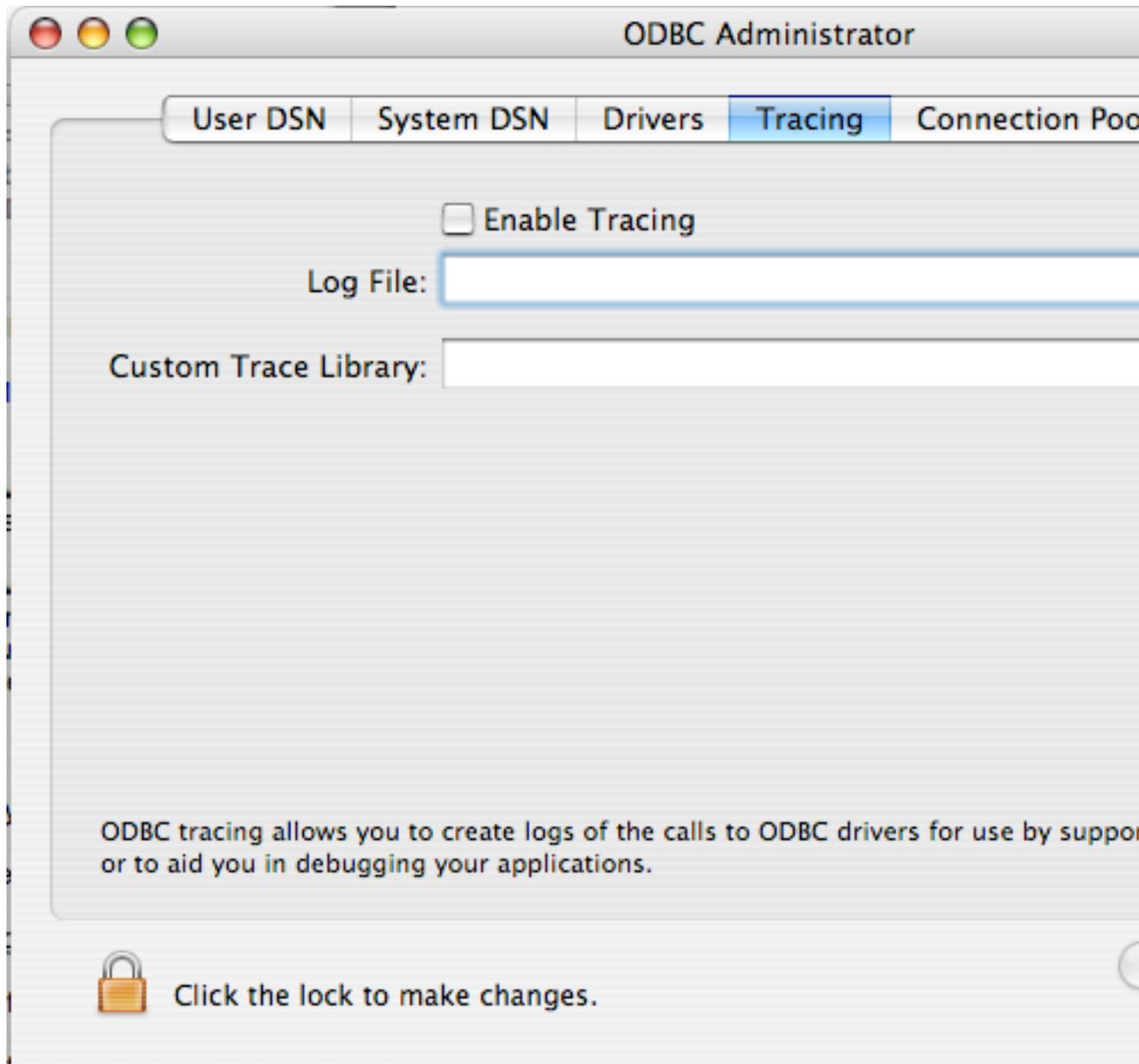
2. When you activate tracing from the [Tracing](#) tab, the [Driver Manager](#) logs all ODBC function calls for all subsequently run applications.

3. ODBC function calls from applications running before tracing is activated are not logged. ODBC function calls are recorded in a log file you specify.
4. Tracing ceases only after you click [Stop Tracing Now](#). Remember that while tracing is on, the log file continues to increase in size and that tracing affects the performance of all your ODBC applications.

22.1.4.8.2. Enabling ODBC Tracing on Mac OS X

To enable the trace option on Mac OS X 10.3 or later, use the [Tracing](#) tab within ODBC Administrator .

1. Open the ODBC Administrator.
2. Select the [Tracing](#) tab.



3. Select the [Enable Tracing](#) check box.

4. Enter the location to save the Tracing log. To append information to an existing log file, click the **Choose...** button.

22.1.4.8.3. Enabling ODBC Tracing on Unix

To enable the trace option on Mac OS X 10.2 (or earlier) or Unix, add the `trace` option to the ODBC configuration:

1. On Unix, explicitly set the `Trace` option in the `ODBC.INI` file.

Set the tracing `ON` or `OFF` by using `TraceFile` and `Trace` parameters in `odbc.ini` as shown below:

```
TraceFile = /tmp/odbc.trace
Trace     = 1
```

`TraceFile` specifies the name and full path of the trace file and `Trace` is set to `ON` or `OFF`. You can also use `1` or `YES` for `ON` and `0` or `NO` for `OFF`. If you are using `ODBCConfig` from `unixODBC`, then follow the instructions for tracing `unixODBC` calls at [HOWTO-ODBCConfig](#).

22.1.4.8.4. Enabling a Connector/ODBC Log

To generate a Connector/ODBC log, do the following:

1. Within Windows, enable the `Trace Connector/ODBC` option flag in the Connector/ODBC connect/configure screen. The log is written to file `C:\myodbc.log`. If the trace option is not remembered when you are going back to the above screen, it means that you are not using the `myodbc.dll` driver, see [Section 22.1.4.3.3, "Troubleshooting ODBC Connection Problems"](#).

On Mac OS X, Unix, or if you are using a DSN-less connection, either supply `OPTION=4` in the connection string, or set the corresponding keyword/value pair in the DSN.

2. Start your application and try to get it to fail. Then check the Connector/ODBC trace file to find out what could be wrong.

If you need help determining what is wrong, see [Section 22.1.8.1, "Connector/ODBC Community Support"](#).

22.1.5. Connector/ODBC Examples

Once you have configured a DSN to provide access to a database, how you access and use that connection is dependent on the application or programming language. As ODBC is a standardized interface, any application or language that supports ODBC can use the DSN and connect to the configured database.

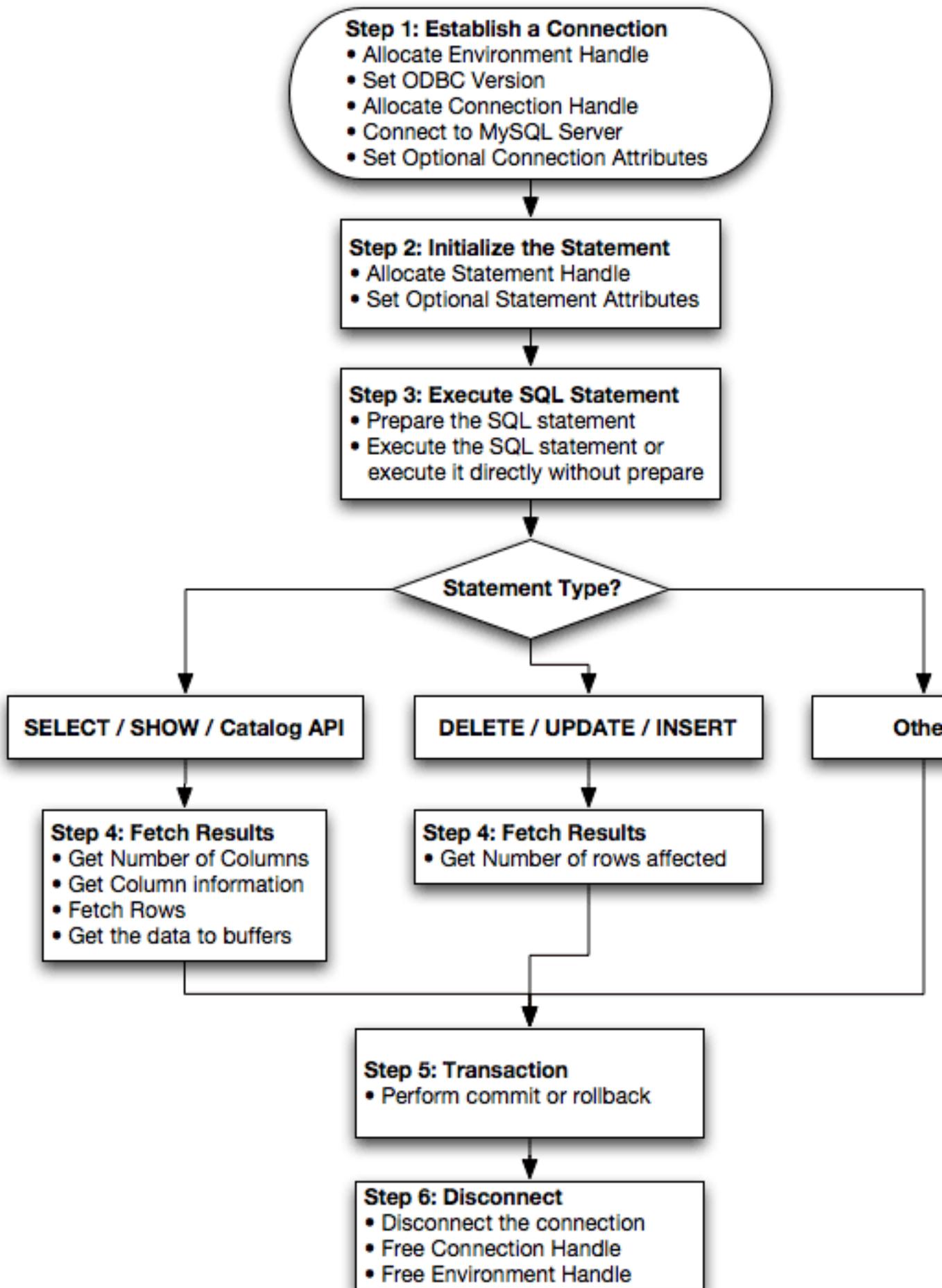
22.1.5.1. Basic Connector/ODBC Application Steps

Interacting with a MySQL server from an applications using the Connector/ODBC typically involves the following operations:

- Configure the Connector/ODBC DSN.
- Connect to MySQL server.
- Initialization operations.
- Execute SQL statements.
- Retrieve results.

- Perform [transactions](#).
- Disconnect from the server.

Most applications use some variation of these steps. The basic application steps are shown in the following diagram:



22.1.5.2. Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC

A typical situation where you would install Connector/ODBC is to access a database on a Linux or Unix host from a Windows machine.

As an example of the process required to set up access between two machines, the steps below take you through the basic steps. These instructions assume that you connect to system ALPHA from system BETA with a user name and password of `myuser` and `mypassword`.

On system ALPHA (the MySQL server) follow these steps:

1. Start the MySQL server.
2. Use `GRANT` to set up an account with a user name of `myuser` that can connect from system BETA using a password of `myuser` to the database `test`:

```
GRANT ALL ON test.* to 'myuser'@'BETA' IDENTIFIED BY 'mypassword';
```

For more information about MySQL privileges, refer to [Section 6.3, "MySQL User Account Management"](#).

On system BETA (the Connector/ODBC client), follow these steps:

1. Configure a Connector/ODBC DSN using parameters that match the server, database and authentication information that you have just configured on system ALPHA.

Parameter	Value	Comment
DSN	remote_test	A name to identify the connection.
SERVER	ALPHA	The address of the remote server.
DATABASE	test	The name of the default database.
USER	myuser	The user name configured for access to this database.
PASSWORD	mypassword	The password for <code>myuser</code> .

2. Using an ODBC-capable application, such as Microsoft Office, connect to the MySQL server using the DSN you have just created. If the connection fails, use tracing to examine the connection process. See [Section 22.1.4.8, "Getting an ODBC Trace File"](#), for more information.

22.1.5.3. Connector/ODBC and Third-Party ODBC Tools

Once you have configured your Connector/ODBC DSN, you can access your MySQL database through any application that supports the ODBC interface, including programming languages and third-party applications. This section contains guides and help on using Connector/ODBC with various ODBC-compatible tools and applications, including Microsoft Word, Microsoft Excel and Adobe/Macromedia ColdFusion.

Connector/ODBC has been tested with the following applications:

Publisher	Application	Notes
Adobe	ColdFusion	Formerly Macromedia ColdFusion
Borland	C++ Builder	
	Builder 4	
	Delphi	
Business Objects	Crystal Reports	

Publisher	Application	Notes
Claris	Filemaker Pro	
Corel	Paradox	
Computer Associates	Visual Objects	Also known as CAVO
	AllFusion ERwin Data Modeler	
Gupta	Team Developer	Previously known as Centura Team Developer; Gupta SQL/Windows
Gensym	G2-ODBC Bridge	
Inline	iHTML	
Lotus	Notes	Versions 4.5 and 4.6
Microsoft	Access	
	Excel	
	Visio Enterprise	
	Visual C++	
	Visual Basic	
	ODBC.NET	Using C#, Visual Basic, C++
	FoxPro	
Visual Interdev		
OpenOffice.org	OpenOffice.org	
Perl	DBD::ODBC	
Pervasive Software	DataJunction	
Sambar Technologies	Sambar Server	
SPSS	SPSS	
SoftVelocity	Clarion	
SQLExpress	SQLExpress for Xbase++	
Sun	StarOffice	
SunSystems	Vision	
Sybase	PowerBuilder	
	PowerDesigner	
theKompany.com	Data Architect	

If you know of any other applications that work with Connector/ODBC, please send mail to myodbc@lists.mysql.com about them.

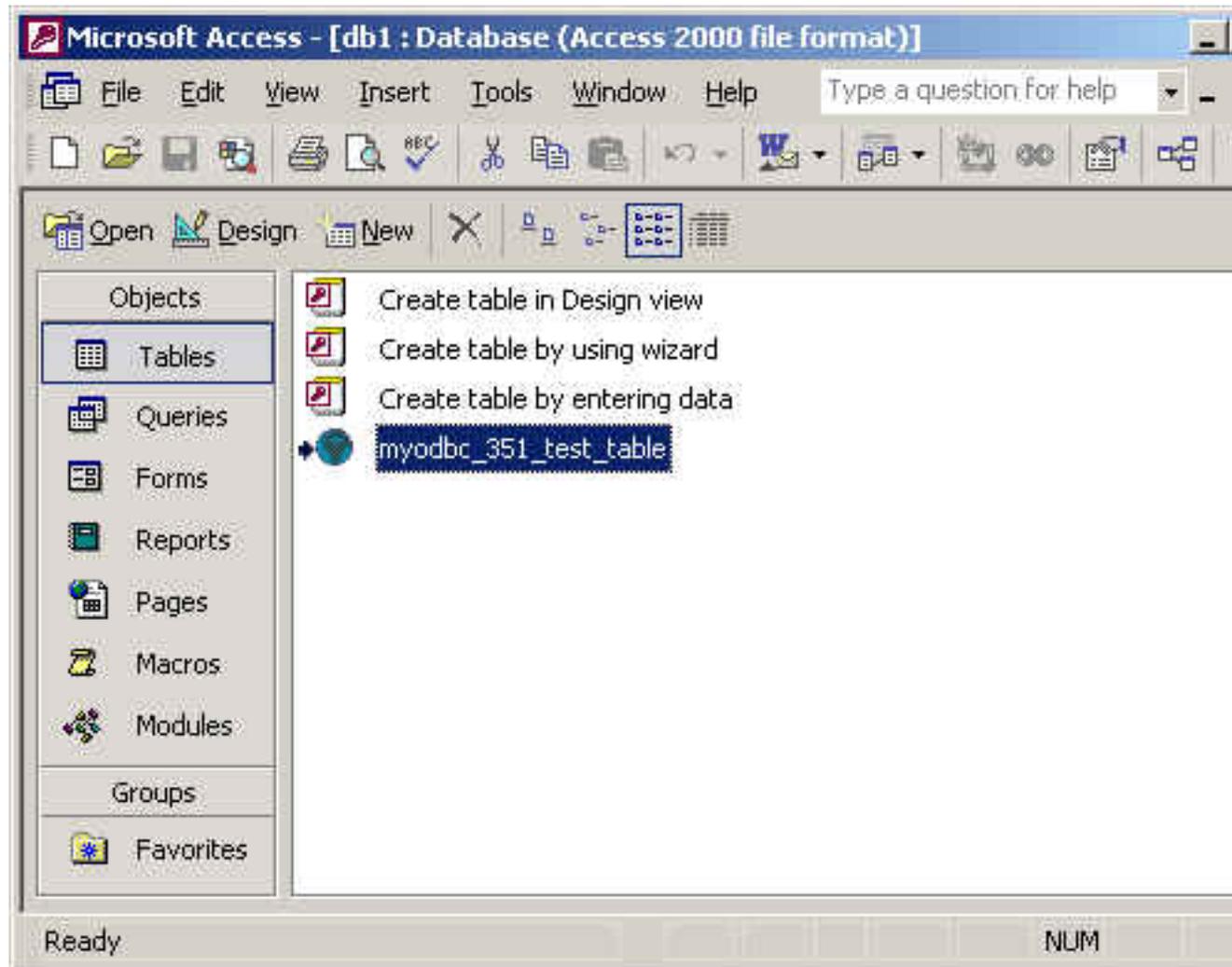
22.1.5.4. Using Connector/ODBC with Microsoft Access

You can use MySQL database with Microsoft Access using Connector/ODBC. The MySQL database can be used as an import source, an export source, or as a linked table for direct use within an Access application, so you can use Access as the front-end interface to a MySQL database.

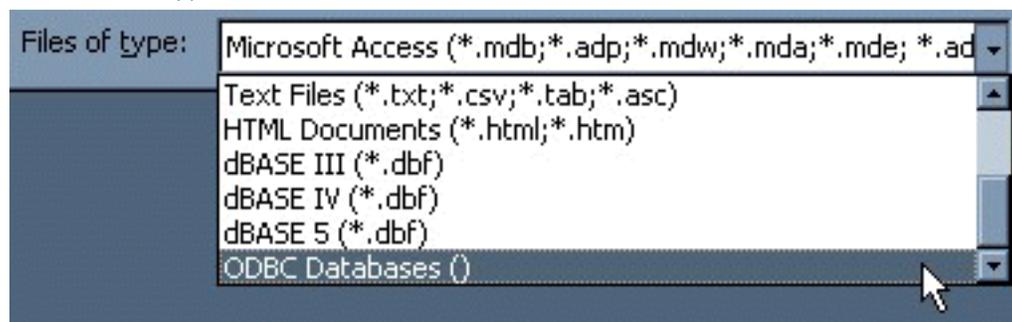
22.1.5.4.1. Exporting Access Data to MySQL

To export a table of data from an Access database to MySQL, follow these instructions:

1. When you open an Access database or an Access project, a Database window appears. It displays shortcuts for creating new database objects and opening existing objects.



2. Click the name of the [table](#) or [query](#) to export, and then in the [File](#) menu, select [Export](#).
3. In the [Export Object Type Object name To](#) dialog box, in the [Save As Type](#) box, select [ODBC Databases \(\)](#) as shown here:



4. In the [Export](#) dialog box, enter a name for the file (or use the suggested name), and then select [OK](#).

5. The Select Data Source dialog box is displayed; it lists the defined data sources for any ODBC drivers installed on your computer. Click either the File Data Source or Machine Data Source tab, and then double-click the Connector/ODBC or Connector/ODBC 3.51 data source to export to. To define a new data source for Connector/ODBC, please [Section 22.1.4.3, “Configuring a Connector/ODBC DSN on Windows”](#).



Note

Ensure that the information that you are exporting to the MySQL table is valid for the corresponding MySQL data types. Values that are outside of the supported range of the MySQL data type but valid within Access may trigger an “overflow” error during the export.

Microsoft Access connects to the MySQL Server through this data source and exports new tables and or data.

22.1.5.4.2. Importing MySQL Data to Access

To import a table or tables from MySQL to Access, follow these instructions:

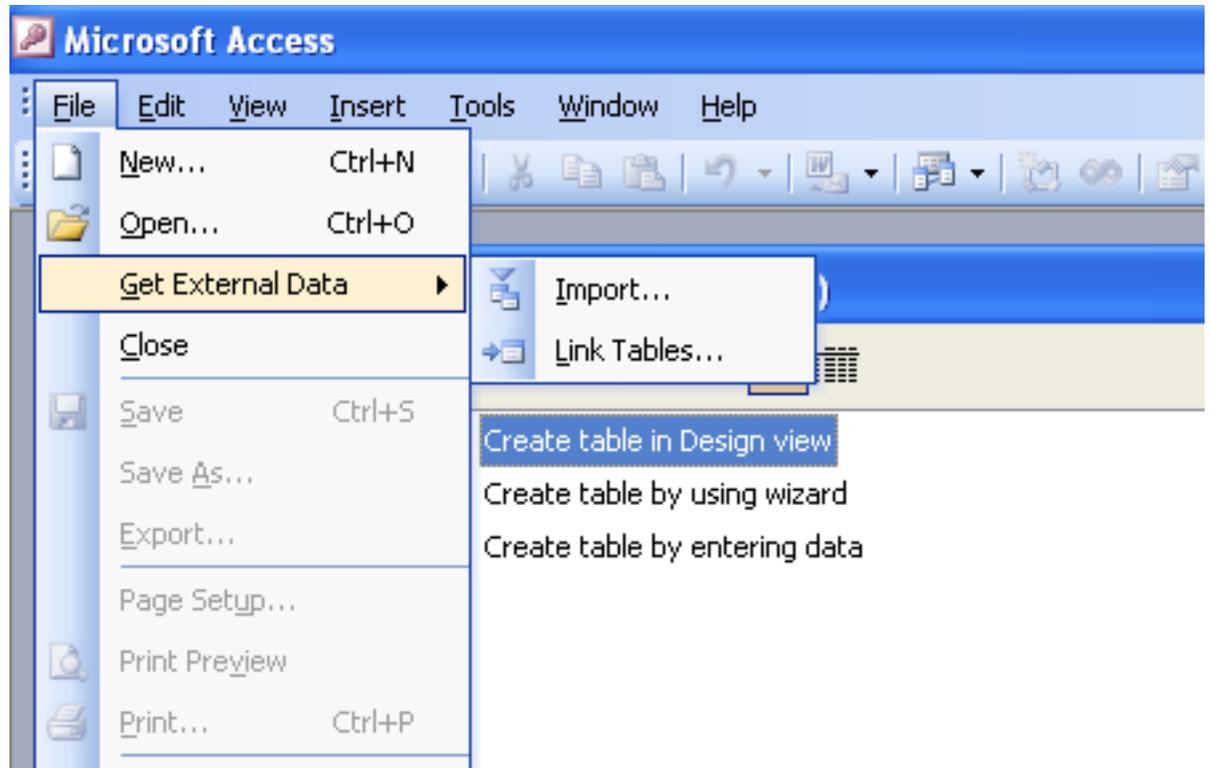
1. Open a database, or switch to the Database window for the open database.
2. To import tables, on the **File** menu, point to **Get External Data**, and then click **Import**.
3. In the **Import** dialog box, in the Files Of Type box, select **ODBC Databases ()**. The Select Data Source dialog box lists the defined data sources **The Select Data Source** dialog box is displayed; it lists the defined data source names.
4. If the ODBC data source that you selected requires you to log on, enter your login ID and password (additional information might also be required), and then click **OK**.
5. Microsoft Access connects to the MySQL server through **ODBC data source** and displays the list of tables that you can **import**.
6. Click each table to **import**, and then click **OK**.

22.1.5.4.3. Using Microsoft Access as a Front-end to MySQL

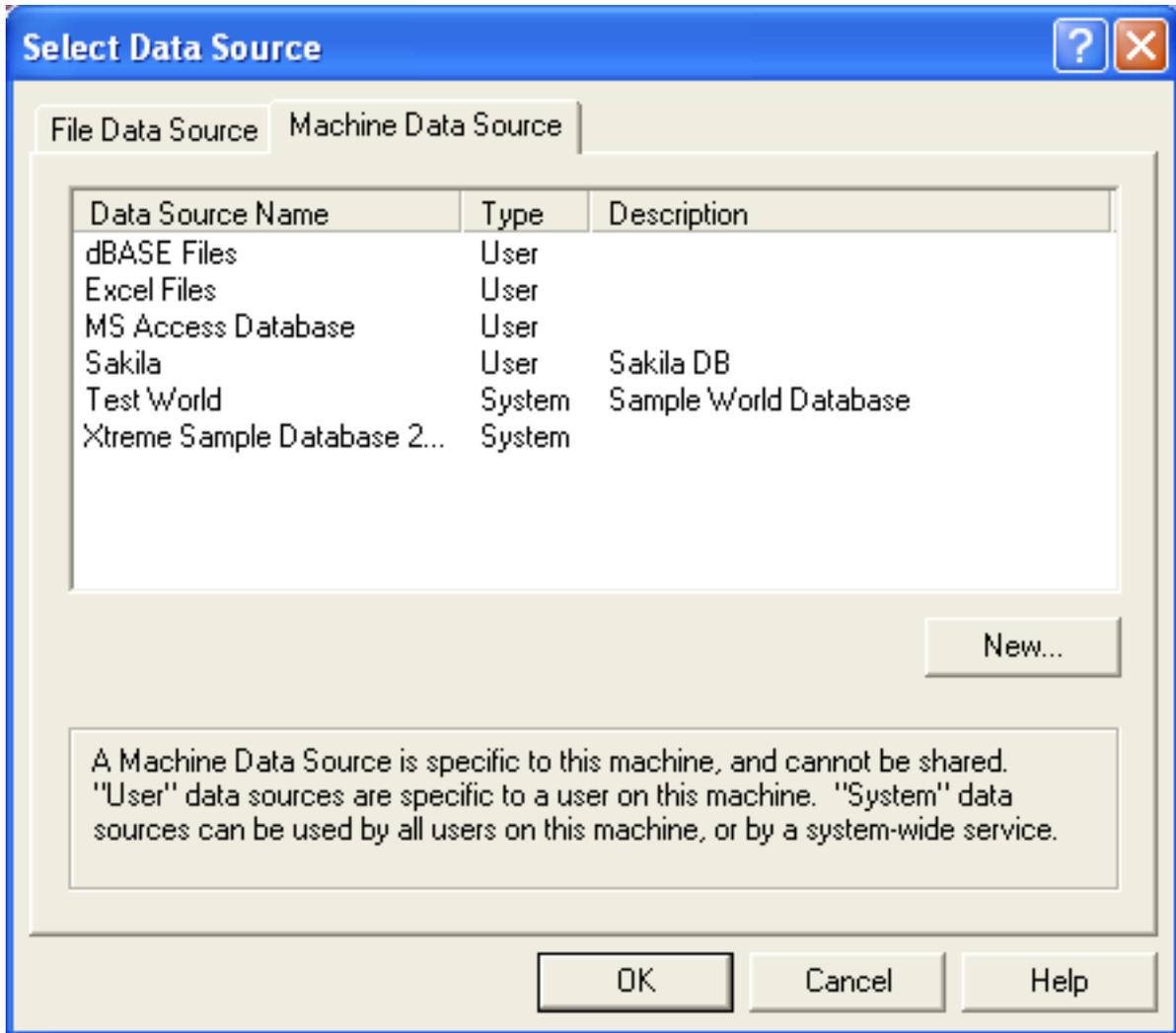
You can use Microsoft Access as a front end to a MySQL database by linking tables within your Microsoft Access database to tables that exist within your MySQL database. When a query is requested on a table within Access, ODBC is used to execute the queries on the MySQL database instead.

To create a linked table:

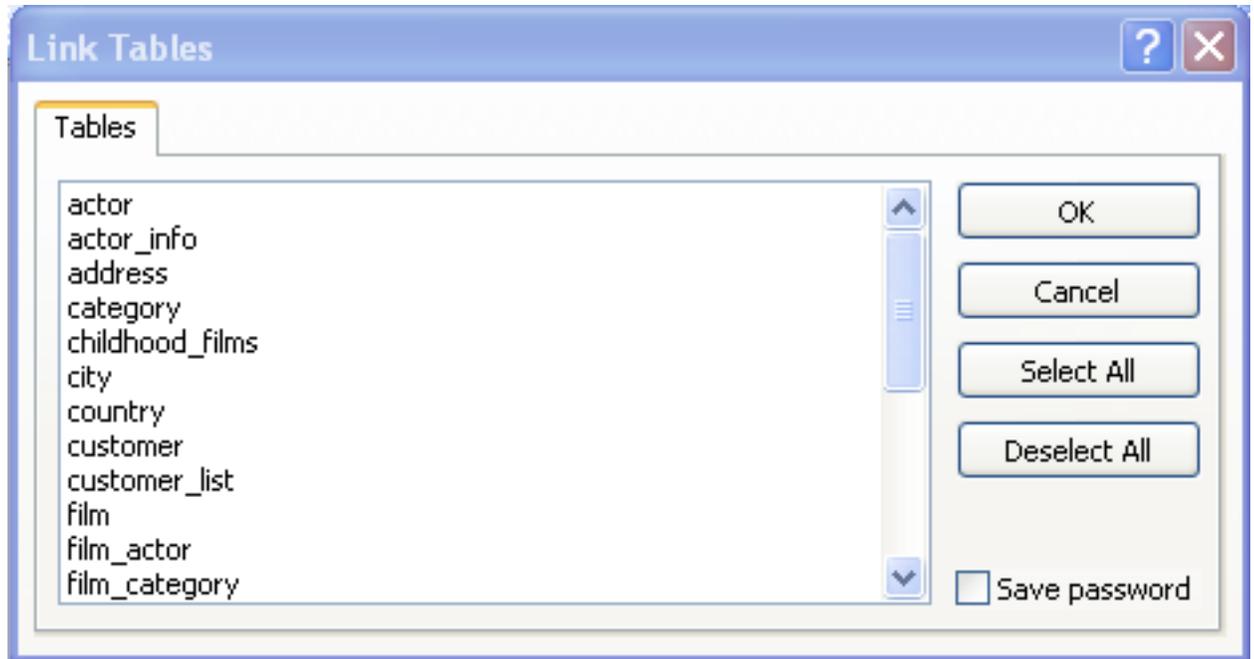
1. Open the Access database that you want to link to MySQL.
2. From the **File**, choose **Get External Data->Link Tables**.



3. From the browser, choose **ODBC Databases ()** from the **Files of type** pop-up.
4. In the **Select Data Source** window, choose an existing DSN, either from a **File Data Source** or **Machine Data Source**. You can also create a new DSN using the **New...** button. For more information on creating a DSN see [Section 22.1.4.3, "Configuring a Connector/ODBC DSN on Windows"](#).



5. In the **Link Tables** dialog, select one or more tables from the MySQL database. A link will be created to each table that you select from this list.



6. If Microsoft Access is unable to determine the unique record identifier for a table automatically then it may ask you to confirm the column, or combination of columns, to be used to uniquely identify each row from the source table. Select the columns to use and click OK.



Once the process has been completed, you can now build interfaces and queries to the linked tables just as you would for any Access database.

Use the following procedure to view or to refresh links when the structure or location of a linked table has changed. The Linked Table Manager lists the paths to all currently linked tables.

To view or refresh links:

1. Open the database that contains links to MySQL tables.
2. On the **Tools** menu, point to **Add-ins** (**Database Utilities** in Access 2000 or newer), and then click **Linked Table Manager**.
3. Select the check box for the tables whose links you want to refresh.
4. Click **OK** to refresh the links.

Microsoft Access confirms a successful refresh or, if the table wasn't found, displays the **Select New Location of <table name>** dialog box in which you can specify its the table's new location. If several selected tables have moved to the new location that you specify, the Linked Table Manager searches that location for all selected tables, and updates all links in one step.

To change the path for a set of linked tables:

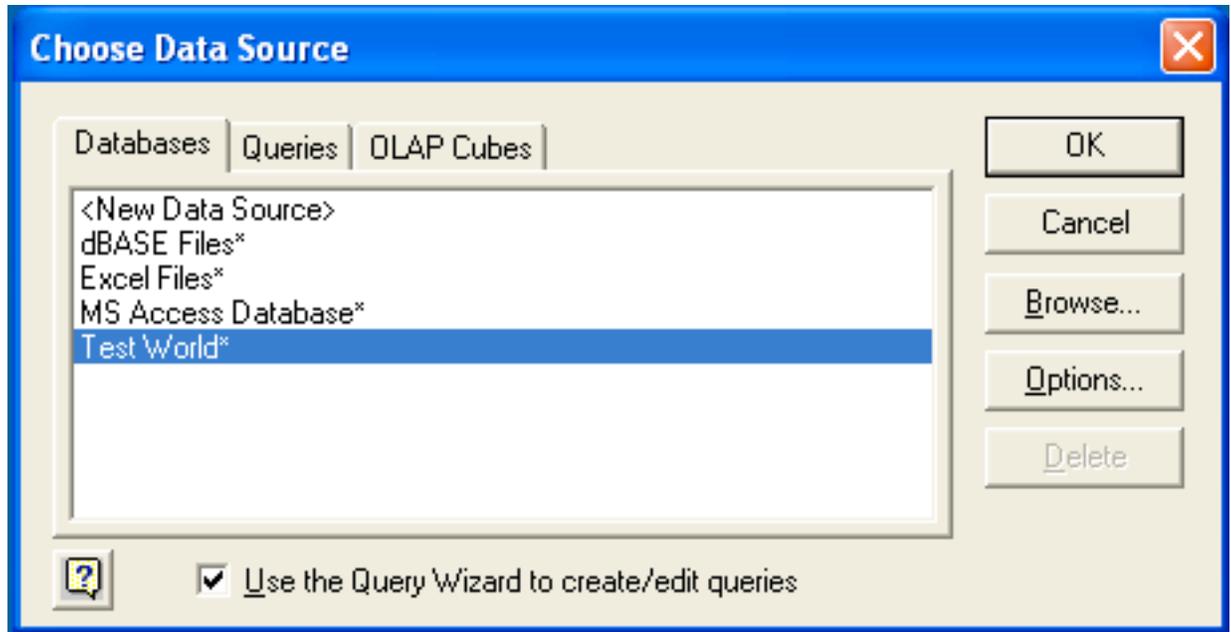
1. Open the database that contains links to tables.
2. On the **Tools** menu, point to **Add-ins** (**Database Utilities** in Access 2000 or newer), and then click **Linked Table Manager**.
3. Select the **Always Prompt For A New Location** check box.
4. Select the check box for the tables whose links you want to change, and then click **OK**.
5. In the **Select New Location of <table name>** dialog box, specify the new location, click **Open**, and then click **OK**.

22.1.5.5. Using Connector/ODBC with Microsoft Word or Excel

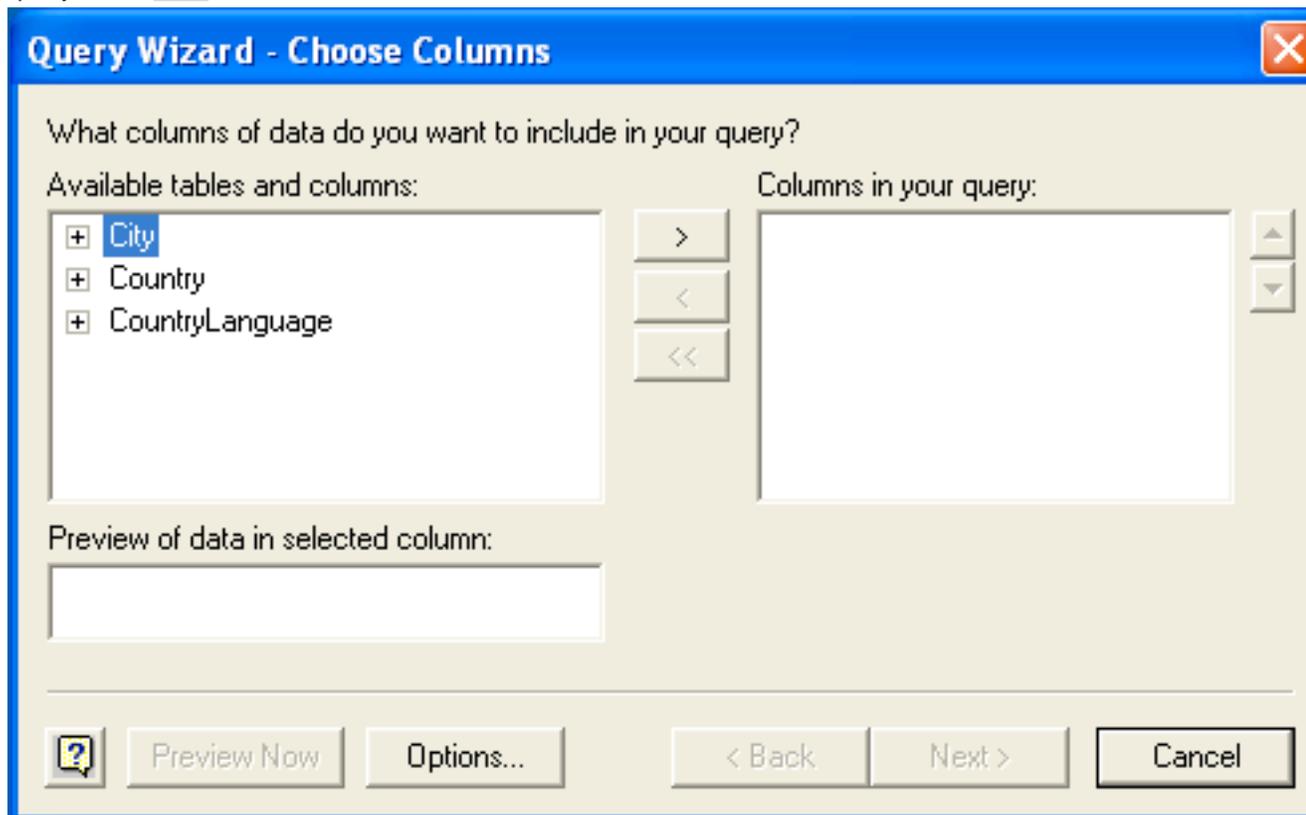
You can use Microsoft Word and Microsoft Excel to access information from a MySQL database using Connector/ODBC. Within Microsoft Word, this facility is most useful when importing data for mailmerge, or for tables and data to be included in reports. Within Microsoft Excel, you can execute queries on your MySQL server and import the data directly into an Excel Worksheet, presenting the data as a series of rows and columns.

With both applications, data is accessed and imported into the application using Microsoft Query, which lets you execute a query through an ODBC source. You use Microsoft Query to build the SQL statement to be executed, selecting the tables, fields, selection criteria and sort order. For example, to insert information from a table in the World test database into an Excel spreadsheet, using the DSN samples shown in [Section 22.1.4, "Configuring Connector/ODBC"](#):

1. Create a new Worksheet.
2. From the **Data** menu, choose **Import External Data**, and then select **New Database Query**.
3. Microsoft Query will start. First, you need to choose the data source, by selecting an existing Data Source Name.



4. Within the [Query Wizard](#), choose the columns to import. The list of tables available to the user configured through the DSN is shown on the left, the columns that will be added to your query are shown on the right. The columns you choose are equivalent to those in the first section of a [SELECT](#) query. Click Next to continue.



5. You can filter rows from the query (the equivalent of a [WHERE](#) clause) using the [Filter Data](#) dialog. Click Next to continue.

Query Wizard - Filter Data

Filter the data to specify which rows to include in your query.
If you don't want to filter the data, click Next.

Column to filter:

- Name
- CountryCode
- District
- Population

Only include rows where:

Name

And Or

And Or

And Or

6. Select an (optional) sort order for the data. This is equivalent to using a `ORDER BY` clause in your SQL query. You can select up to three fields for sorting the information returned by the query. Click Next to continue.

Query Wizard - Sort Order

Specify how you want your data sorted.
If you don't want to sort the data, click Next.

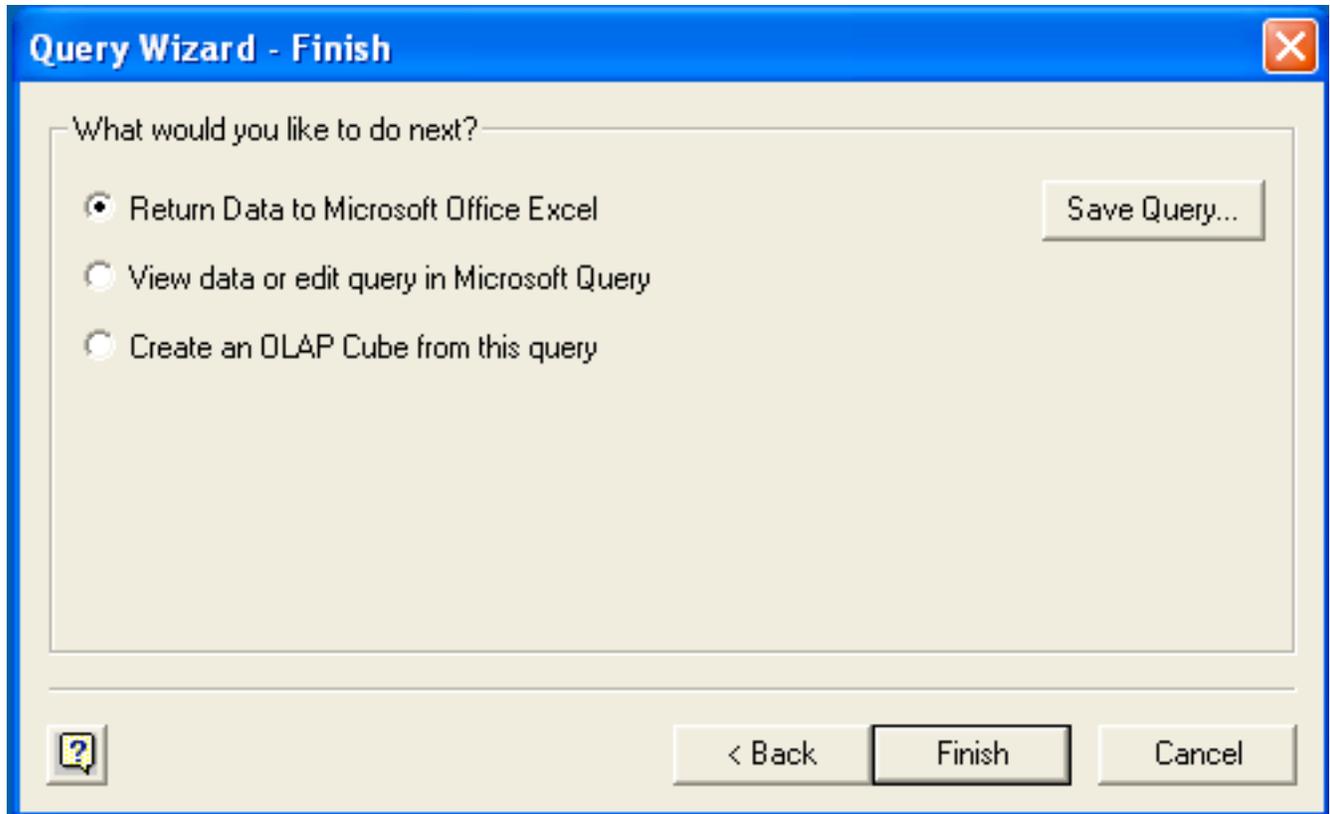
Sort by
Name Ascending
 Descending

Then by
 Ascending
 Descending

Then by
 Ascending
 Descending

< Back Next > Cancel

7. Select the destination for your query. You can select to return the data Microsoft Excel, where you can choose a worksheet and cell where the data will be inserted; you can continue to view the query and results within Microsoft Query, where you can edit the SQL query and further filter and sort the information returned; or you can create an OLAP Cube from the query, which can then be used directly within Microsoft Excel. Click **Finish**.



The same process can be used to import data into a Word document, where the data will be inserted as a table. This can be used for mail merge purposes (where the field data is read from a Word table), or where you want to include data and reports within a report or other document.

22.1.5.6. Using Connector/ODBC with Crystal Reports

Crystal Reports can use an ODBC DSN to connect to a database from which you to extract data and information for reporting purposes.



Note

There is a known issue with certain versions of Crystal Reports where the application is unable to open and browse tables and fields through an ODBC connection. Before using Crystal Reports with MySQL, please ensure that you have update to the latest version, including any outstanding service packs and hotfixes. For more information on this issue, see the [Business\) Objects Knowledgebase](#) for more information.

For example, to create a simple crosstab report within Crystal Reports XI, follow these steps:

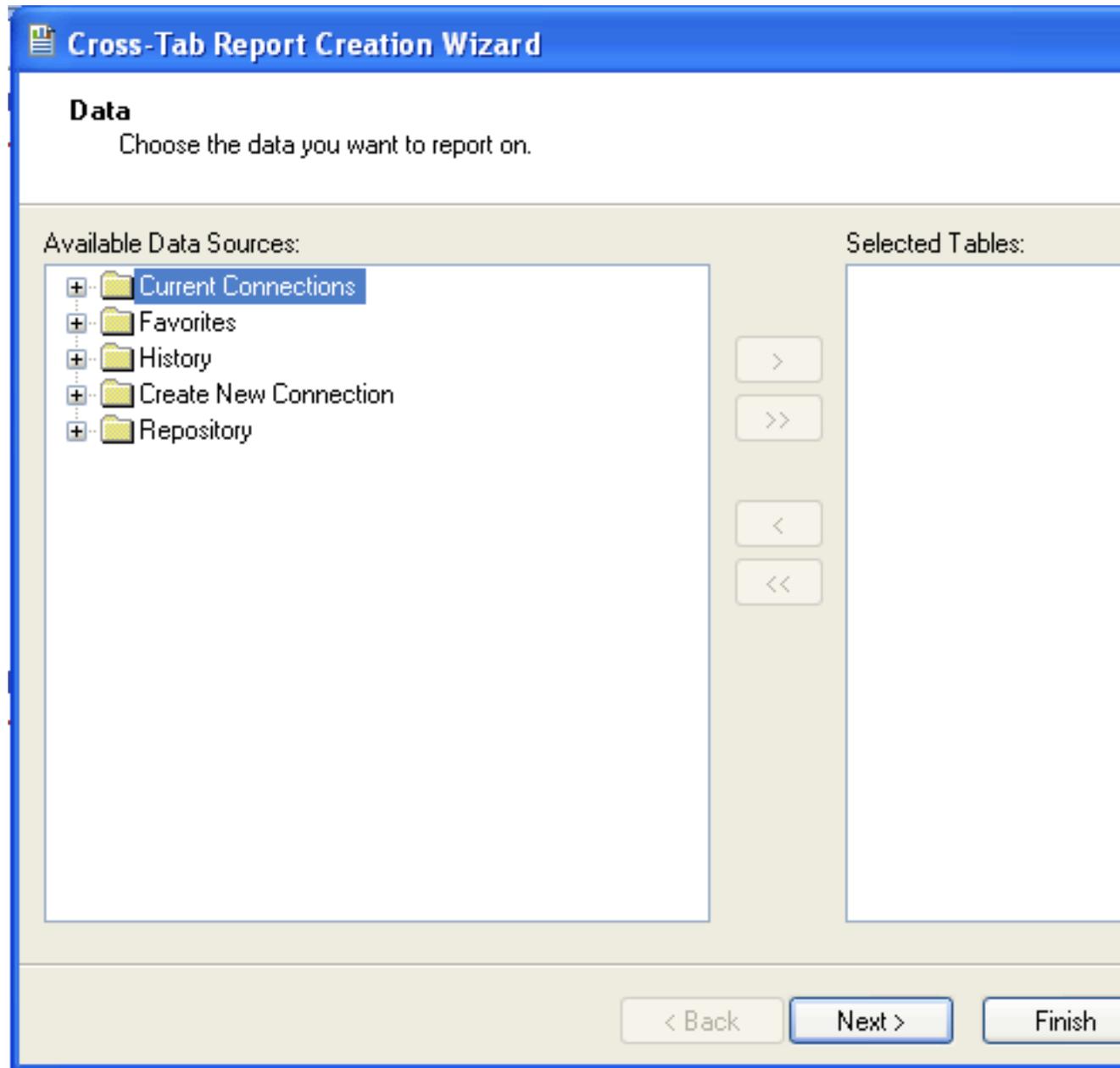
1. Create a DSN using the [Data Sources \(ODBC\)](#) tool. You can either specify a complete database, including user name and password, or you can build a basic DSN and use Crystal Reports to set the user name and password.

For the purposes of this example, a DSN that provides a connection to an instance of the MySQL Sakila sample database has been created.

2. Open Crystal Reports and create a new project, or an open an existing reporting project into which you want to insert data from your MySQL data source.

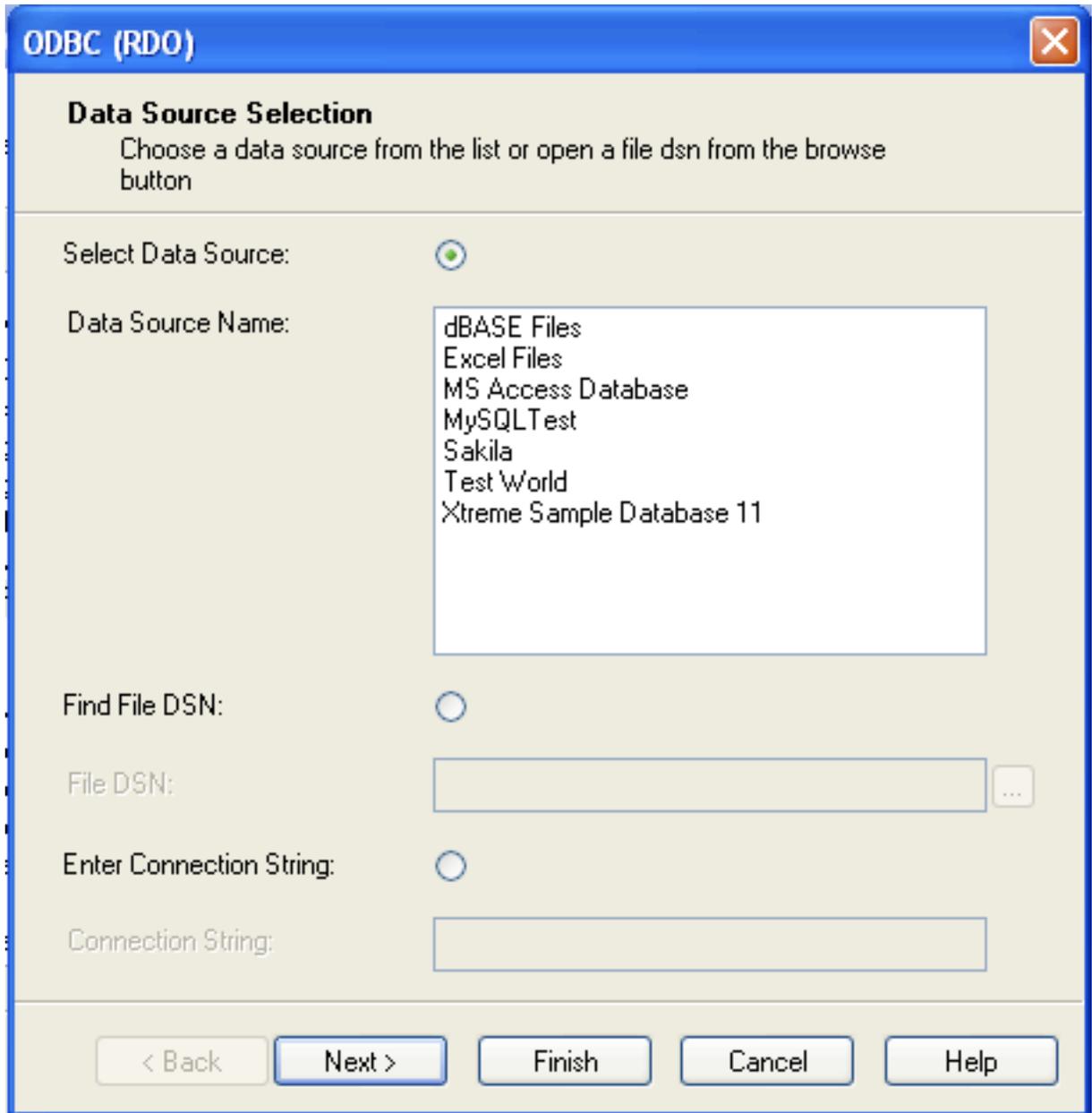
3. Start the Cross-Tab Report Wizard, either by clicking the option on the Start Page. Expand the **Create New Connection** folder, then expand the **ODBC (RDO)** folder to obtain a list of ODBC data sources.

You will be asked to select a data source.



4. When you first expand the **ODBC (RDO)** folder you will be presented the Data Source Selection screen. From here you can select either a pre-configured DSN, open a file-based DSN or enter and manual connection string. For this example, the **Sakila** DSN will be used.

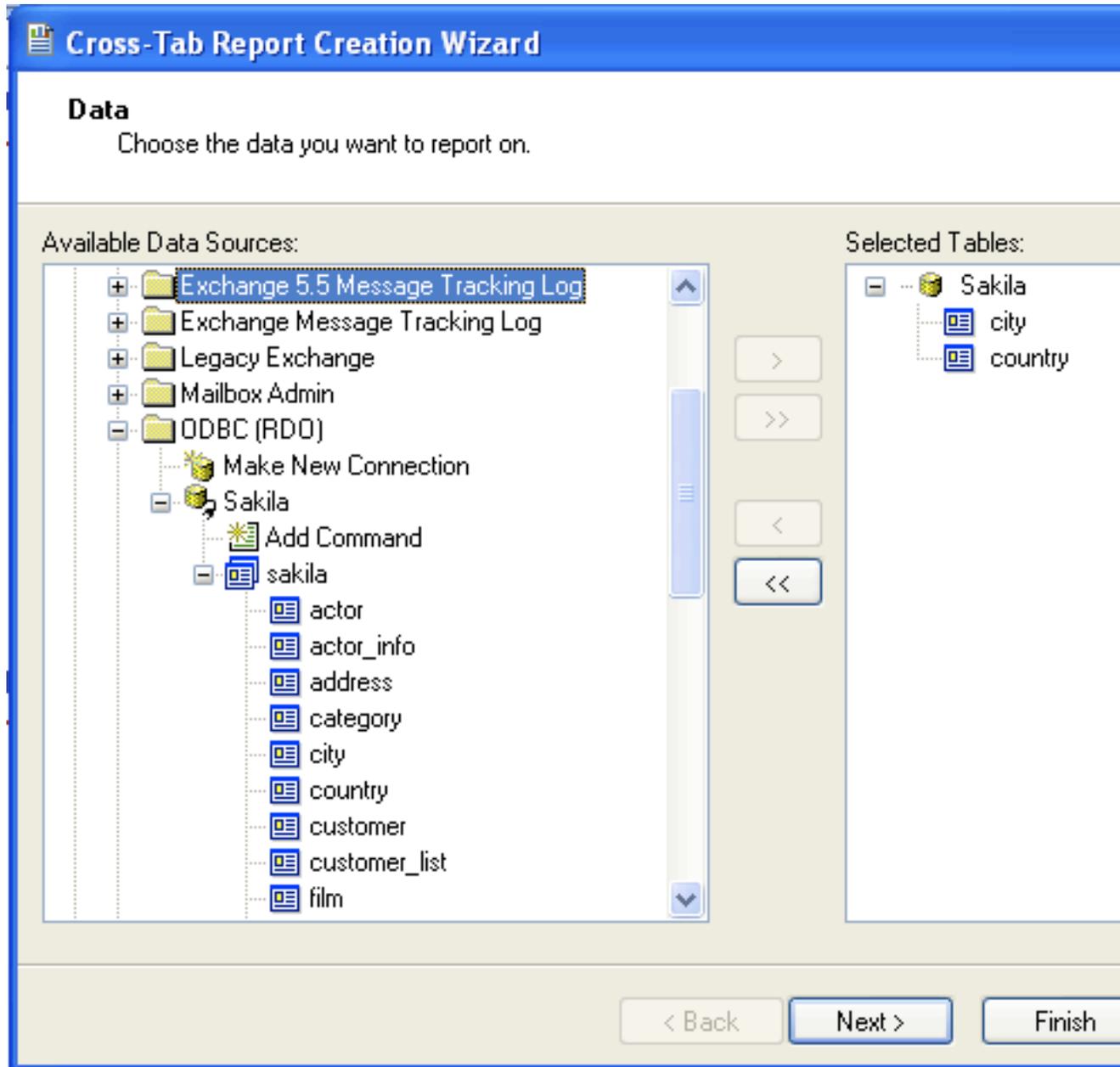
If the DSN contains a user name/password combination, or you want to use different authentication credentials, click **Next** to enter the user name and password that you want to use. Otherwise, click **Finish** to continue the data source selection wizard.



5. You will be returned the Cross-Tab Report Creation Wizard. You now need to select the database and tables that you want to include in your report. For our example, we will expand the selected Sakila database. Click the `city` table and use the `>` button to add the table to the report. Then repeat the action with the `country` table. Alternatively you can select multiple tables and add them to the report.

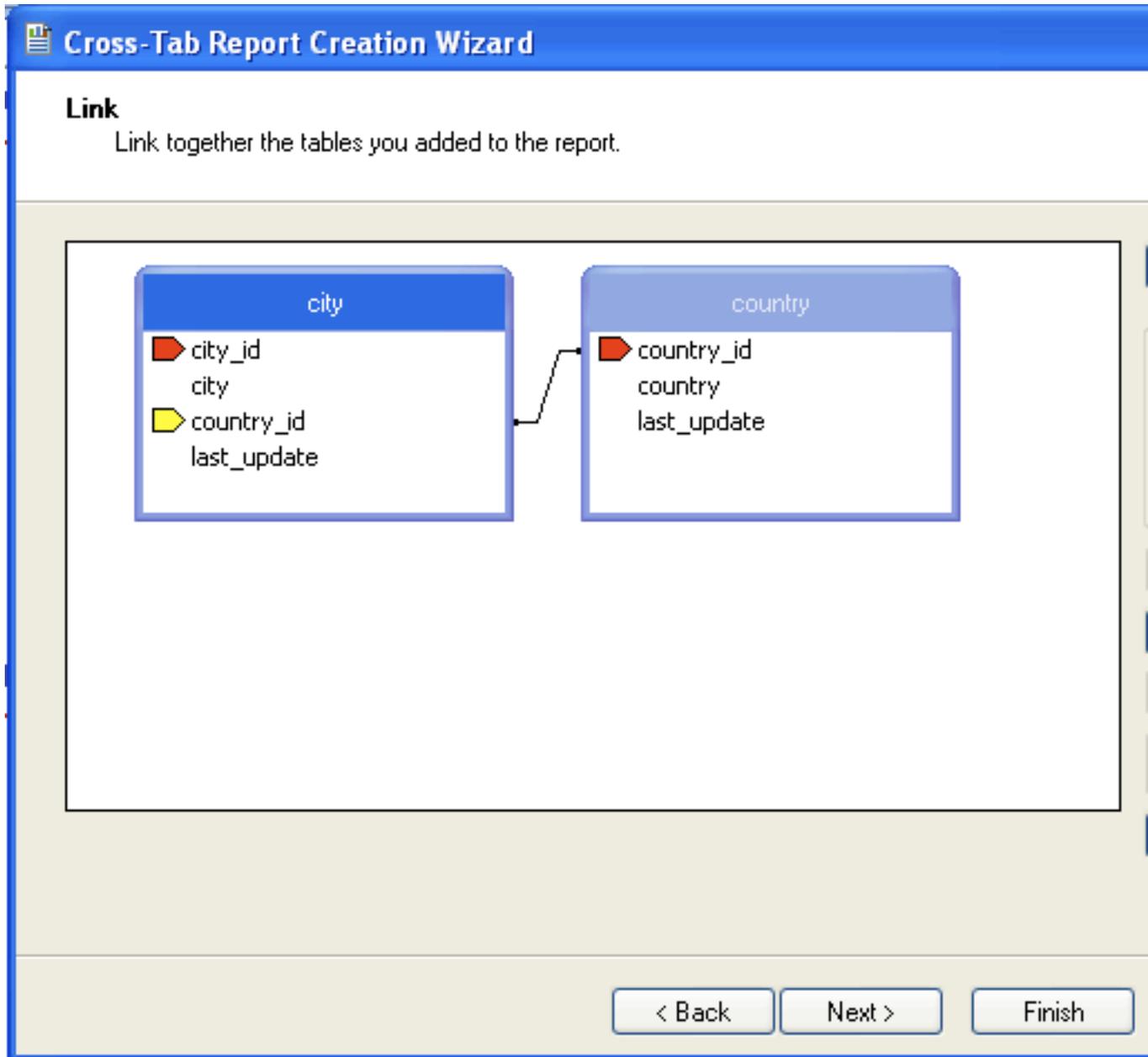
Finally, you can select the parent **Sakila** resource and add of the tables to the report.

Once you have selected the tables you want to include, click **Next** to continue.



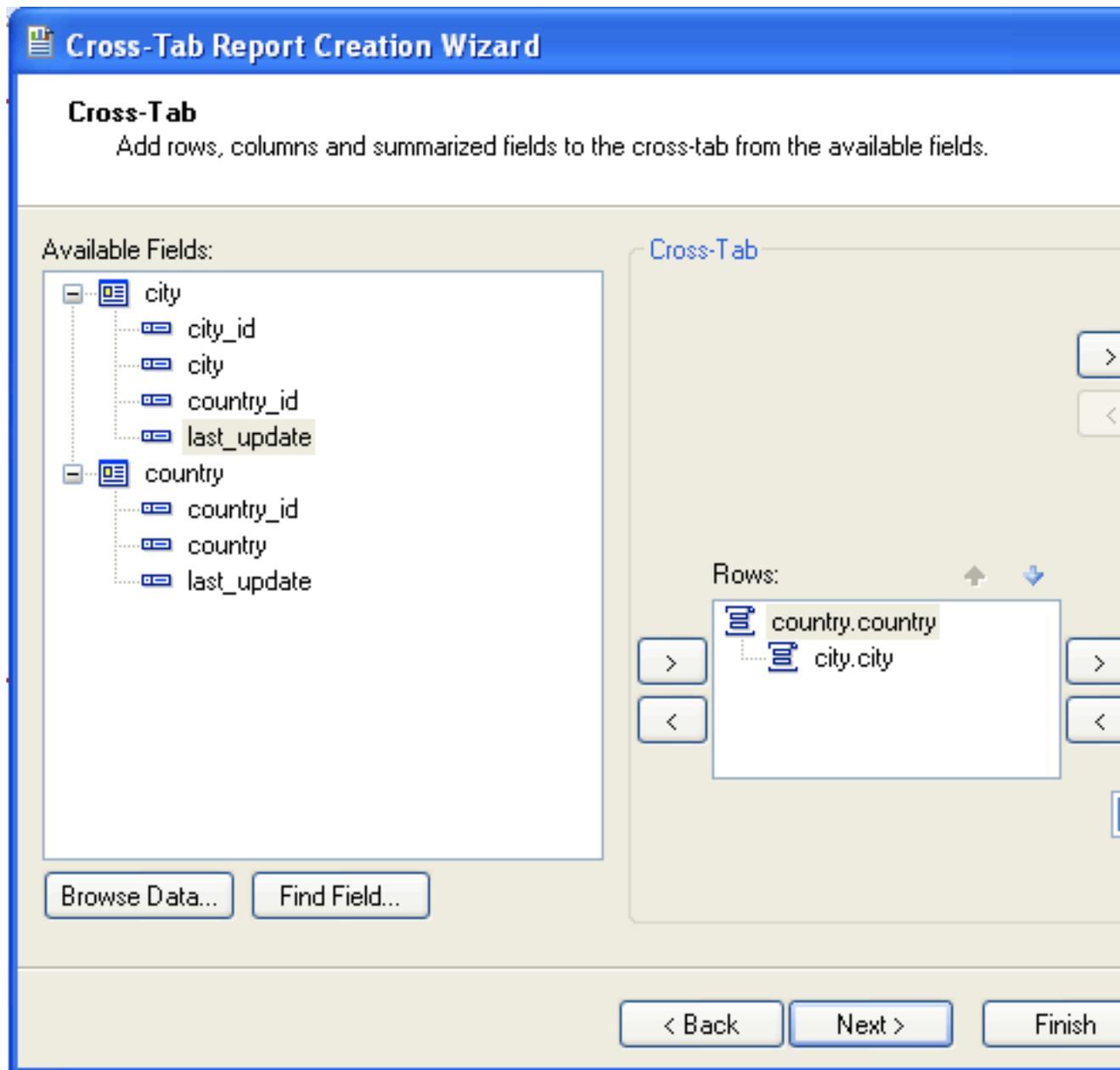
6. Crystal Reports will now read the table definitions and automatically identify the links between the tables. The identification of links between tables enables Crystal Reports to automatically lookup and summarize information based on all the tables in the database according to your query. If Crystal Reports is unable to perform the linking itself, you can manually create the links between fields in the tables you have selected.

Click **Next** to continue the process.



7. You can now select the columns and rows that to include within the Cross-Tab report. Drag and drop or use the **>** buttons to add fields to each area of the report. In the example shown, we will report on cities, organized by country, incorporating a count of the number of cities within each country. If you want to browse the data, select a field and click the **Browse Data...** button.

Click **Next** to create a graph of the results. Since we are not creating a graph from this data, click **Finish** to generate the report.



8. The finished report will be shown, a sample of the output from the Sakila sample database is shown below.

		Total
Total		600
Afghanistan	Total	1
	Kabul	1
Algeria	Total	3
	Batna	1
	Bchar	1
	Skikda	1
American Samoa	Total	1
	Tafuna	1
Angola	Total	2
	Benguela	1
	Namibe	1
Anguilla	Total	1
	South Hill	1
Argentina	Total	13
	Almirante Brow	1

Once the ODBC connection has been opened within Crystal Reports, you can browse and add any fields within the available tables into your reports.

22.1.5.7. Connector/ODBC Programming

With a suitable ODBC Manager and the Connector/ODBC driver installed, any programming language or environment that can support ODBC can connect to a MySQL database through Connector/ODBC.

This includes, but is not limited to, Microsoft support languages (including Visual Basic, C# and interfaces such as ODBC.NET), Perl (through the DBI module, and the DBD::ODBC driver).

22.1.5.7.1. Using Connector/ODBC with Visual Basic Using ADO, DAO and RDO

This section contains simple examples of the use of Connector/ODBC with ADO, DAO and RDO.

22.1.5.7.1.1. ADO: `rs.addNew`, `rs.delete`, and `rs.update`

The following ADO (ActiveX Data Objects) example creates a table `my_ado` and demonstrates the use of `rs.addNew`, `rs.delete`, and `rs.update`.

```
Private Sub myodbc_ado_Click()

Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim fld As ADODB.Field
Dim sql As String

'connect to MySQL server using Connector/ODBC
Set conn = New ADODB.Connection
conn.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"

conn.Open

'create table
conn.Execute "DROP TABLE IF EXISTS my_ado"
conn.Execute "CREATE TABLE my_ado(id int not null primary key, name varchar(20)," _
& "txt text, dt date, tm time, ts timestamp)"

'direct insert
conn.Execute "INSERT INTO my_ado(id,name,txt) values(1,100,'venu')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(2,200,'MySQL')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(3,300,'Delete')"

Set rs = New ADODB.Recordset
rs.CursorLocation = adUseServer

'fetch the initial table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Initial my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close

'rs insert
rs.Open "select * from my_ado", conn, adOpenDynamic, adLockOptimistic
rs.AddNew
rs!Name = "Monty"
rs!txt = "Insert row"
rs.Update
rs.Close

'rs update
```

```

rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-row"
rs.Update
rs.Close

'rs update second time..
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-second-time"
rs.Update
rs.Close

'rs delete
rs.Open "SELECT * FROM my_ado"
rs.MoveNext
rs.MoveNext
rs.Delete
rs.Close

'fetch the updated table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Updated my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close
conn.Close
End Sub

```

22.1.5.7.1.2. DAO: `rs.addNew`, `rs.update`, and Scrolling

The following DAO (Data Access Objects) example creates a table `my_dao` and demonstrates the use of `rs.addNew`, `rs.update`, and result set scrolling.

```

Private Sub myodbc_dao_Click()

Dim ws As Workspace
Dim conn As Connection
Dim queryDef As queryDef
Dim str As String

'connect to MySQL using MySQL ODBC 3.51 Driver
Set ws = DBEngine.CreateWorkspace("", "venu", "venu", dbUseODBC)
str = "odbc;DRIVER={MySQL ODBC 3.51 Driver};_"
& "SERVER=localhost;_"
& " DATABASE=test;_"
& "UID=venu;PWD=venu; OPTION=3"
Set conn = ws.OpenConnection("test", dbDriverNoPrompt, False, str)

'Create table my_dao
Set queryDef = conn.CreateQueryDef("", "drop table if exists my_dao")
queryDef.Execute

Set queryDef = conn.CreateQueryDef("", "create table my_dao(Id INT AUTO_INCREMENT PRIMARY KEY, " _
& "Ts TIMESTAMP(14) NOT NULL, Name varchar(20), Id2 INT)")

```

```

queryDef.Execute

'Insert new records using rs.addNew
Set rs = conn.OpenRecordset("my_dao")
Dim i As Integer

For i = 10 To 15
rs.AddNew
rs!Name = "insert record" & i
rs!Id2 = i
rs.Update
Next i
rs.Close

'rs update..
Set rs = conn.OpenRecordset("my_dao")
rs.Edit
rs!Name = "updated-string"
rs.Update
rs.Close

'fetch the table back...
Set rs = conn.OpenRecordset("my_dao", dbOpenDynamic)
str = "Results:"
rs.MoveFirst
While Not rs.EOF
str = " " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print "DATA:" & str
rs.MoveNext
Wend

'rs Scrolling
rs.MoveFirst
str = " FIRST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

rs.MoveLast
str = " LAST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

rs.MovePrevious
str = " LAST-1 ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

'free all resources
rs.Close
queryDef.Close
conn.Close
ws.Close

End Sub

```

22.1.5.7.1.3. RDO: `rs.addNew` and `rs.update`

The following RDO (Remote Data Objects) example creates a table `my_rdo` and demonstrates the use of `rs.addNew` and `rs.update`.

```

Dim rs As rdoResultset
Dim cn As New rdoConnection
Dim cl As rdoColumn
Dim SQL As String

'cn.Connect = "DSN=test;"
cn.Connect = "DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& "DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"

```

```
cn.CursorDriver = rdUseOdbc
cn.EstablishConnection rdDriverPrompt

'drop table my_rdo
SQL = "drop table if exists my_rdo"
cn.Execute SQL, rdExecDirect

'create table my_rdo
SQL = "create table my_rdo(id int, name varchar(20))"
cn.Execute SQL, rdExecDirect

'insert - direct
SQL = "insert into my_rdo values (100,'venu')"
cn.Execute SQL, rdExecDirect

SQL = "insert into my_rdo values (200,'MySQL')"
cn.Execute SQL, rdExecDirect

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 300
rs!Name = "Insert1"
rs.Update
rs.Close

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 400
rs!Name = "Insert 2"
rs.Update
rs.Close

'rs update
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.Edit
rs!id = 999
rs!Name = "updated"
rs.Update
rs.Close

'fetch back...
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
Do Until rs.EOF
For Each cl In rs.rdoColumns
Debug.Print cl.Value,
Next
rs.MoveNext
Debug.Print
Loop
Debug.Print "Row count="; rs.RowCount

'close
rs.Close
cn.Close

End Sub
```

22.1.5.7.2. Using Connector/ODBC with .NET

This section contains simple examples that demonstrate the use of Connector/ODBC drivers with ODBC.NET.

22.1.5.7.2.1. Using Connector/ODBC with ODBC.NET and C# (C sharp)

The following sample creates a table `my_odbc_net` and demonstrates its use in C#.

```
/**
 * @sample      : mycon.cs
 * @purpose     : Demo sample for ODBC.NET using Connector/ODBC
 * @author      : Venu, <myodbc@lists.mysql.com>
 *
 * (C) Copyright MySQL AB, 1995-2006
 *
 **/

/* build command
 *
 * csc /t:exe
 *      /out:mycon.exe mycon.cs
 *      /r:Microsoft.Data.Odbc.dll
 */

using Console = System.Console;
using Microsoft.Data.Odbc;

namespace myodbc3
{
    class mycon
    {
        static void Main(string[] args)
        {
            try
            {
                //Connection string for Connector/ODBC 3.51
                string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
                    "SERVER=localhost;" +
                    "DATABASE=test;" +
                    "UID=venu;" +
                    "PASSWORD=venu;" +
                    "OPTION=3";

                //Connect to MySQL using Connector/ODBC
                OdbcConnection MyConnection = new OdbcConnection(MyConString);
                MyConnection.Open();

                Console.WriteLine("\n !!! success, connected successfully !!!\n");

                //Display connection information
                Console.WriteLine("Connection Information:");
                Console.WriteLine("\tConnection String:" +
                    MyConnection.ConnectionString);
                Console.WriteLine("\tConnection Timeout:" +
                    MyConnection.ConnectionTimeout);
                Console.WriteLine("\tDatabase:" +
                    MyConnection.Database);
                Console.WriteLine("\tDataSource:" +
                    MyConnection.DataSource);
                Console.WriteLine("\tDriver:" +
                    MyConnection.Driver);
                Console.WriteLine("\tServerVersion:" +
                    MyConnection.ServerVersion);

                //Create a sample table
                OdbcCommand MyCommand =
                    new OdbcCommand("DROP TABLE IF EXISTS my_odbc_net",
                        MyConnection);
            }
        }
    }
}
```

Connector/ODBC Examples

```
MyCommand.ExecuteNonQuery();
MyCommand.CommandText =
    "CREATE TABLE my_odbc_net(id int, name varchar(20), idb bigint)";
MyCommand.ExecuteNonQuery();

//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(10,'venu', 300)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());

//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(20,'mysql',400)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());

//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(20,'mysql',500)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());

//Update
MyCommand.CommandText =
    "UPDATE my_odbc_net SET id=999 WHERE id=20";
Console.WriteLine("Update, Total rows affected:" +
    MyCommand.ExecuteNonQuery());

//COUNT(*)
MyCommand.CommandText =
    "SELECT COUNT(*) as TRows FROM my_odbc_net";
Console.WriteLine("Total Rows:" +
    MyCommand.ExecuteScalar());

//Fetch
MyCommand.CommandText = "SELECT * FROM my_odbc_net";
OdbcDataReader MyDataReader;
MyDataReader = MyCommand.ExecuteReader();
while (MyDataReader.Read())
{
    if(string.Compare(MyConnection.Driver,"myodbc3.dll") == 0) {
        //Supported only by Connector/ODBC 3.51
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
            MyDataReader.GetString(1) + " " +
            MyDataReader.GetInt64(2));
    }
    else {
        //BIGINTs not supported by Connector/ODBC
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
            MyDataReader.GetString(1) + " " +
            MyDataReader.GetInt32(2));
    }
}

//Close all resources
MyDataReader.Close();
MyConnection.Close();
}
catch (OdbcException MyOdbcException) //Catch any ODBC exception ..
{
    for (int i=0; i < MyOdbcException.Errors.Count; i++)
    {
        Console.Write("ERROR #" + i + "\n" +
            "Message: " +
            MyOdbcException.Errors[i].Message + "\n" +
            "Native: " +
```



```

MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(10,'venu')"
Console.WriteLine("INSERT, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'Insert
MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
Console.WriteLine("INSERT, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'Insert
MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
Console.WriteLine("INSERT, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'Insert
MyCommand.CommandText = "INSERT INTO my_vb_net(id) VALUES(30)"
Console.WriteLine("INSERT, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'Update
MyCommand.CommandText = "UPDATE my_vb_net SET id=999 WHERE id=20"
Console.WriteLine("Update, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'COUNT(*)
MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_vb_net"
Console.WriteLine("Total Rows:" & MyCommand.ExecuteScalar())

'Select
Console.WriteLine("Select * FROM my_vb_net")
MyCommand.CommandText = "SELECT * FROM my_vb_net"
Dim MyDataReader As OdbcDataReader
MyDataReader = MyCommand.ExecuteReader
While MyDataReader.Read
    If MyDataReader("name") Is DBNull.Value Then
        Console.WriteLine("id = " & _
CStr(MyDataReader("id")) & " name = " & _
"NULL")
    Else
        Console.WriteLine("id = " & _
CStr(MyDataReader("id")) & " name = " & _
CStr(MyDataReader("name")))
    End If
End While

'Catch ODBC Exception
Catch MyOdbcException As OdbcException
Dim i As Integer
Console.WriteLine(MyOdbcException.ToString)

'Catch program exception
Catch MyException As Exception
Console.WriteLine(MyException.ToString)
End Try
End Sub

```

22.1.6. Connector/ODBC Reference

This section provides reference material for the Connector/ODBC API, showing supported functions and methods, supported MySQL column types and the corresponding native type in Connector/ODBC, and the error codes returned by Connector/ODBC when a fault occurs.

22.1.6.1. Connector/ODBC API Reference

This section summarizes ODBC routines, categorized by functionality.

For the complete ODBC API reference, please refer to the ODBC Programmer's Reference at <http://msdn.microsoft.com/en-us/library/ms714177.aspx>.

An application can call `SQLGetInfo` function to obtain conformance information about Connector/ODBC. To obtain information about support for a specific function in the driver, an application can call `SQLGetFunctions`.

**Note**

For backward compatibility, the Connector/ODBC driver supports all deprecated functions.

The following tables list Connector/ODBC API calls grouped by task:

Table 22.6. ODBC API Calls for Connecting to a Data Source

Function Name	Connector/ODBC Supports?	Standard	Purpose
<code>SQLAllocHandle</code>	Yes	ISO 92	Obtains an environment, connection, statement, or descriptor handle.
<code>SQLConnect</code>	Yes	ISO 92	Connects to a specific driver by data source name, user ID, and password.
<code>SQLDriverConnect</code>	Yes	ODBC	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
<code>SQLAllocEnv</code>	Yes	Deprecated	Obtains an environment handle allocated from driver.
<code>SQLAllocConnect</code>	Yes	Deprecated	Obtains a connection handle

Table 22.7. ODBC API Calls for Obtaining Information about a Driver and Data Source

Function Name	Connector/ODBC Supports?	Standard	Purpose
<code>SQLDataSources</code>	No	ISO 92	Returns the list of available data sources, handled by the Driver Manager
<code>SQLDrivers</code>	No	ODBC	Returns the list of installed drivers and their attributes, handles by Driver Manager
<code>SQLGetInfo</code>	Yes	ISO 92	Returns information about a specific driver and data source.
<code>SQLGetFunctions</code>	Yes	ISO 92	Returns supported driver functions.
<code>SQLGetTypeInfo</code>	Yes	ISO 92	Returns information about supported data types.

Table 22.8. ODBC API Calls for Setting and Retrieving Driver Attributes

Function Name	Connector/ODBC Supports?	Standard	Purpose
<code>SQLSetConnectAttr</code>	Yes	ISO 92	Sets a connection attribute.
<code>SQLGetConnectAttr</code>	Yes	ISO 92	Returns the value of a connection attribute.

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLSetConnectOption	Yes	Deprecated	Sets a connection option
SQLGetConnectOption	Yes	Deprecated	Returns the value of a connection option
SQLSetEnvAttr	Yes	ISO 92	Sets an environment attribute.
SQLGetEnvAttr	Yes	ISO 92	Returns the value of an environment attribute.
SQLSetStmtAttr	Yes	ISO 92	Sets a statement attribute.
SQLGetStmtAttr	Yes	ISO 92	Returns the value of a statement attribute.
SQLSetStmtOption	Yes	Deprecated	Sets a statement option
SQLGetStmtOption	Yes	Deprecated	Returns the value of a statement option

Table 22.9. ODBC API Calls for Preparing SQL Requests

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLAllocStmt	Yes	Deprecated	Allocates a statement handle
SQLPrepare	Yes	ISO 92	Prepares an SQL statement for later execution.
SQLBindParameter	Yes	ODBC	Assigns storage for a parameter in an SQL statement. Connector/ODBC 5.2 adds support for “out” and “inout” parameters, through the SQL_PARAM_OUTPUT or SQL_PARAM_INPUT_OUTPUT type specifiers. (“Out” and “inout” parameters are not supported for LONGTEXT and LONGBLOB columns.)
SQLGetCursorName	Yes	ISO 92	Returns the cursor name associated with a statement handle.
SQLSetCursorName	Yes	ISO 92	Specifies a cursor name.
SQLSetScrollOptions	Yes	ODBC	Sets options that control cursor behavior.

Table 22.10. ODBC API Calls for Submitting Requests

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLExecute	Yes	ISO 92	Executes a prepared statement.
SQLExecDirect	Yes	ISO 92	Executes a statement
SQLNativeSql	Yes	ODBC	Returns the text of an SQL statement as translated by the driver.
SQLDescribeParam	Yes	ODBC	Returns the description for a specific parameter in a statement.
SQLNumParams	Yes	ISO 92	Returns the number of parameters in a statement.
SQLParamData	Yes	ISO 92	Used in conjunction with SQLPutData to supply parameter data at execution time. (Useful for long data values.)

Function Name	Connector/ ODBC Supports?	Standard	Purpose
<code>SQLPutData</code>	Yes	ISO 92	Sends part or all of a data value for a parameter. (Useful for long data values.)

Table 22.11. ODBC API Calls for Retrieving Results and Information about Results

Function Name	Connector/ ODBC Supports?	Standard	Purpose
<code>SQLRowCount</code>	Yes	ISO 92	Returns the number of rows affected by an insert, update, or delete request.
<code>SQLNumResultCols</code>	Yes	ISO 92	Returns the number of columns in the result set.
<code>SQLDescribeCol</code>	Yes	ISO 92	Describes a column in the result set.
<code>SQLColAttribute</code>	Yes	ISO 92	Describes attributes of a column in the result set.
<code>SQLColAttributes</code>	Yes	Deprecated	Describes attributes of a column in the result set.
<code>SQLFetch</code>	Yes	ISO 92	Returns multiple result rows.
<code>SQLFetchScroll</code>	Yes	ISO 92	Returns scrollable result rows.
<code>SQLExtendedFetch</code>	Yes	Deprecated	Returns scrollable result rows.
<code>SQLSetPos</code>	Yes	ODBC	Positions a cursor within a fetched block of data and enables an application to refresh data in the rowset or to update or delete data in the result set.
<code>SQLBulkOperations</code>	Yes	ODBC	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.

Table 22.12. ODBC API Calls for Retrieving Error or Diagnostic Information

Function Name	Connector/ ODBC Supports?	Standard	Purpose
<code>SQLError</code>	Yes	Deprecated	Returns additional error or status information
<code>SQLGetDiagField</code>	Yes	ISO 92	Returns additional diagnostic information (a single field of the diagnostic data structure).
<code>SQLGetDiagRec</code>	Yes	ISO 92	Returns additional diagnostic information (multiple fields of the diagnostic data structure).

Table 22.13. ODBC API Calls for Obtaining Information about the Data Source's System Tables (Catalog Functions) Item

Function Name	Connector/ ODBC Supports?	Standard	Purpose
<code>SQLColumnPrivileges</code>	Yes	ODBC	Returns a list of columns and associated privileges for one or more tables.
<code>SQLColumns</code>	Yes	X/Open	Returns the list of column names in specified tables.

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLForeignKeys	Yes	ODBC	Returns a list of column names that make up foreign keys, if they exist for a specified table.
SQLPrimaryKeys	Yes	ODBC	Returns the list of column names that make up the primary key for a table.
SQLSpecialColumns	Yes	X/Open	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	Yes	ISO 92	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges	Yes	ODBC	Returns a list of tables and the privileges associated with each table.
SQLTables	Yes	X/Open	Returns the list of table names stored in a specific data source.

Table 22.14. ODBC API Calls for Performing Transactions

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLTransact	Yes	Deprecated	Commits or rolls back a transaction
SQLEndTran	Yes	ISO 92	Commits or rolls back a transaction .

Table 22.15. ODBC API Calls for Terminating a Statement

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLFreeStmt	Yes	ISO 92	Ends statement processing, discards pending results, and, optionally, frees all resources associated with the statement handle.
SQLCloseCursor	Yes	ISO 92	Closes a cursor that has been opened on a statement handle.
SQLCancel	Yes	ISO 92	Cancels an SQL statement.

Table 22.16. ODBC API Calls for Terminating a Connection

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLDisconnect	Yes	ISO 92	Closes the connection.
SQLFreeHandle	Yes	ISO 92	Releases an environment, connection, statement, or descriptor handle.
SQLFreeConnect	Yes	Deprecated	Releases connection handle.
SQLFreeEnv	Yes	Deprecated	Releases an environment handle.

22.1.6.2. Connector/ODBC Data Types

The following table illustrates how Connector/ODBC maps the server data types to default SQL and C data types.

Table 22.17. How Connector/ODBC Maps MySQL Data Types to SQL and C Data Types

Native Value	SQL Type	C Type
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bit	SQL_BIT	SQL_C_BIT
bit	SQL_CHAR	SQL_C_CHAR
blob	SQL_LONGVARBINARY	SQL_C_BINARY
bool	SQL_CHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
date	SQL_DATE	SQL_C_DATE
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
decimal	SQL_DECIMAL	SQL_C_CHAR
double precision	SQL_DOUBLE	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
enum	SQL_VARCHAR	SQL_C_CHAR
float	SQL_REAL	SQL_C_FLOAT
int unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
integer unsigned	SQL_INTEGER	SQL_C_ULONG
integer	SQL_INTEGER	SQL_C_SLONG
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
longblob	SQL_LONGVARBINARY	SQL_C_BINARY
longtext	SQL_LONGVARCHAR	SQL_C_CHAR
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
mediumint	SQL_INTEGER	SQL_C_SLONG
mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR
real	SQL_FLOAT	SQL_C_DOUBLE
set	SQL_VARCHAR	SQL_C_CHAR
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT
smallint	SQL_SMALLINT	SQL_C_SSHORT
text	SQL_LONGVARCHAR	SQL_C_CHAR
time	SQL_TIME	SQL_C_TIME
timestamp [567]	SQL_TIMESTAMP	SQL_C_TIMESTAMP

Native Value	SQL Type	C Type
<code>tinyblob</code>	<code>SQL_LONGVARBINARY</code>	<code>SQL_C_BINARY</code>
<code>tinyint unsigned</code>	<code>SQL_TINYINT</code>	<code>SQL_C_UTINYINT</code>
<code>tinyint</code>	<code>SQL_TINYINT</code>	<code>SQL_C_STINYINT</code>
<code>tinytext</code>	<code>SQL_LONGVARCHAR</code>	<code>SQL_C_CHAR</code>
<code>varchar</code>	<code>SQL_VARCHAR</code>	<code>SQL_C_CHAR</code>
<code>year</code>	<code>SQL_SMALLINT</code>	<code>SQL_C_SHORT</code>

22.1.6.3. Connector/ODBC Error Codes

The following tables lists the error codes returned by Connector/ODBC apart from the server errors.

Table 22.18. Special Error Codes Returned by Connector/ODBC

Native Code	SQLSTATE 2	SQLSTATE 3	Error Message
500	01000	01000	General warning
501	01004	01004	String data, right truncated
502	01S02	01S02	Option value changed
503	01S03	01S03	No rows updated/deleted
504	01S04	01S04	More than one row updated/deleted
505	01S06	01S06	Attempt to fetch before the result set returned the first row set
506	07001	07002	<code>SQLBindParameter</code> not used for all parameters
507	07005	07005	Prepared statement not a cursor-specification
508	07009	07009	Invalid descriptor index
509	08002	08002	Connection name in use
510	08003	08003	Connection does not exist
511	24000	24000	Invalid cursor state
512	25000	25000	Invalid transaction state
513	25S01	25S01	Transaction state unknown
514	34000	34000	Invalid cursor name
515	S1000	HY000	General driver defined error
516	S1001	HY001	Memory allocation error
517	S1002	HY002	Invalid column number
518	S1003	HY003	Invalid application buffer type
519	S1004	HY004	Invalid SQL data type
520	S1009	HY009	Invalid use of null pointer
521	S1010	HY010	Function sequence error
522	S1011	HY011	Attribute can not be set now
523	S1012	HY012	Invalid transaction operation code
524	S1013	HY013	Memory management error

Native Code	SQLSTATE 2	SQLSTATE 3	Error Message
525	S1015	HY015	No cursor name available
526	S1024	HY024	Invalid attribute value
527	S1090	HY090	Invalid string or buffer length
528	S1091	HY091	Invalid descriptor field identifier
529	S1092	HY092	Invalid attribute/option identifier
530	S1093	HY093	Invalid parameter number
531	S1095	HY095	Function type out of range
532	S1106	HY106	Fetch type out of range
533	S1117	HY117	Row value out of range
534	S1109	HY109	Invalid cursor position
535	S1C00	HYC00	Optional feature not implemented
0	21S01	21S01	Column count does not match value count
0	23000	23000	Integrity constraint violation
0	42000	42000	Syntax error or access violation
0	42S02	42S02	Base table or view not found
0	42S12	42S12	Index not found
0	42S21	42S21	Column already exists
0	42S22	42S22	Column not found
0	08S01	08S01	Communication link failure

22.1.7. Connector/ODBC Notes and Tips

Here are some common notes and tips for using Connector/ODBC within different environments, applications and tools. The notes provided here are based on the experiences of Connector/ODBC developers and users.

22.1.7.1. Connector/ODBC General Functionality

This section provides help with common queries and areas of functionality in MySQL and how to use them with Connector/ODBC.

22.1.7.1.1. Obtaining Auto-Increment Values

Obtaining the value of column that uses `AUTO_INCREMENT` after an `INSERT` statement can be achieved in a number of different ways. To obtain the value immediately after an `INSERT`, use a `SELECT` query with the `LAST_INSERT_ID()` [1194] function.

For example, using Connector/ODBC you would execute two separate statements, the `INSERT` statement and the `SELECT` query to obtain the auto-increment value.

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

If you do not require the value within your application, but do require the value as part of another `INSERT`, the entire process can be handled by executing the following statements:

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
INSERT INTO tbl2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

Certain ODBC applications (including Delphi and Access) may have trouble obtaining the auto-increment value using the previous examples. In this case, try the following statement as an alternative:

```
SELECT * FROM tbl WHERE auto IS NULL;
```

This alternative method requires that `sql_auto_is_null` [554] variable is not set to 0. See [Section 5.1.4, “Server System Variables”](#).

See also [Section 22.8.11.3, “How to Get the Unique ID for the Last Inserted Row”](#).

22.1.7.1.2. Dynamic Cursor Support

Support for the `dynamic cursor` is provided in Connector/ODBC 3.51, but dynamic cursors are not enabled by default. You can enable this function within Windows by selecting the `Enable Dynamic Cursor` check box within the ODBC Data Source Administrator.

On other platforms, you can enable the dynamic cursor by adding `32` to the `OPTION` value when creating the DSN.

22.1.7.1.3. Connector/ODBC Performance

The Connector/ODBC driver has been optimized to provide very fast performance. If you experience problems with the performance of Connector/ODBC, or notice a large amount of disk activity for simple queries, there are a number of aspects to check:

- Ensure that `ODBC Tracing` is not enabled. With tracing enabled, a lot of information is recorded in the tracing file by the ODBC Manager. You can check, and disable, tracing within Windows using the `Tracing` panel of the ODBC Data Source Administrator. Within Mac OS X, check the `Tracing` panel of ODBC Administrator. See [Section 22.1.4.8, “Getting an ODBC Trace File”](#).
- Make sure you are using the standard version of the driver, and not the debug version. The debug version includes additional checks and reporting measures.
- Disable the Connector/ODBC driver trace and query logs. These options are enabled for each DSN, so make sure to examine only the DSN that you are using in your application. Within Windows, you can disable the Connector/ODBC and query logs by modifying the DSN configuration. Within Mac OS X and Unix, ensure that the driver trace (option value 4) and query logging (option value 524288) are not enabled.

22.1.7.1.4. Setting ODBC Query Timeout in Windows

For more information on how to set the query timeout on Microsoft Windows when executing queries through an ODBC connection, read the Microsoft knowledgebase document at <http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B153756>.

22.1.7.2. Connector/ODBC Application-Specific Tips

Most programs should work with Connector/ODBC, but for each of those listed here, there are specific notes and tips to improve or enhance the way you work with Connector/ODBC and these applications.

With all applications, ensure that you are using the latest Connector/ODBC drivers, ODBC Manager and any supporting libraries and interfaces used by your application. For example, on Windows, using the latest version of Microsoft Data Access Components (MDAC) will improve the compatibility with ODBC in general, and with the Connector/ODBC driver.

22.1.7.2.1. Using Connector/ODBC with Microsoft Applications

The majority of Microsoft applications have been tested with Connector/ODBC, including Microsoft Office, Microsoft Access and the various programming languages supported within ASP and Microsoft Visual Studio.

22.1.7.2.1.1. Microsoft Access

To improve the integration between Microsoft Access and MySQL through Connector/ODBC:

- For all versions of Access, enable the Connector/ODBC `Return matching rows` option. For Access 2.0, also enable the `Simulate ODBC 1.0` option.
- Include a `TIMESTAMP` column in all tables that you want to be able to update. For maximum portability, do not use a length specification in the column declaration (which is unsupported within MySQL in versions earlier than 4.1).
- Include a `primary key` in each MySQL table you want to use with Access. If not, new or updated rows may show up as `#DELETED#`.
- Use only `DOUBLE` float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as `#DELETED#` or that you cannot find or update rows.
- If you are using Connector/ODBC to link to a table that has a `BIGINT` column, the results are displayed as `#DELETED#`. The work around solution is:
 - Have one more dummy column with `TIMESTAMP` as the data type.
 - Select the `Change BIGINT columns to INT` option in the connection dialog in ODBC DSN Administrator.
 - Delete the table link from Access and re-create it.

Old records may still display as `#DELETED#`, but newly added/updated records are displayed properly.

- If you still get the error `Another user has changed your data` after adding a `TIMESTAMP` column, the following trick may help you:

Do not use a `table` data sheet view. Instead, create a form with the fields you want, and use that `form` data sheet view. Set the `DefaultValue` property for the `TIMESTAMP` column to `NOW()` [1143]. Consider hiding the `TIMESTAMP` column from view so your users are not confused.

- In some cases, Access may generate SQL statements that MySQL cannot understand. You can fix this by selecting `"Query | SQLSpecific | Pass-Through"` from the Access menu.
- On Windows NT, Access reports `BLOB` columns as `OLE OBJECTS`. If you want to have `MEMO` columns instead, change `BLOB` columns to `TEXT` with `ALTER TABLE`.
- Access cannot always handle the MySQL `DATE` column properly. If you have a problem with these, change the columns to `DATETIME`.
- If you have in Access a column defined as `BYTE`, Access tries to export this as `TINYINT` instead of `TINYINT UNSIGNED`. This gives you problems if you have values larger than 127 in the column.
- If you have very large (long) tables in Access, it might take a very long time to open them. Or you might run low on virtual memory and eventually get an `ODBC Query Failed` error and the table cannot open. To deal with this, select the following options:

- Return Matching Rows (2)
- Allow BIG Results (8).

These add up to a value of 10 (`OPTION=10`).

Some external articles and tips that may be useful when using Access, ODBC and Connector/ODBC:

- Read [How to Trap ODBC Login Error Messages in Access](#)
- Optimizing Access ODBC Applications
 - [Optimizing for Client/Server Performance](#)
 - [Tips for Converting Applications to Using ODBCDirect](#)
 - [Tips for Optimizing Queries on Attached SQL Tables](#)
- For a list of tools that can be used with Access and ODBC data sources, refer to <http://www.mysql.com/portal/software/convertors/> section for list of available tools.

22.1.7.2.1.2. Microsoft Excel and Column Types

If you have problems importing data into Microsoft Excel, particularly numeric, date, and time values, this is probably because of a bug in Excel, where the column type of the source data is used to determine the data type when that data is inserted into a cell within the worksheet. The result is that Excel incorrectly identifies the content and this affects both the display format and the data when it is used within calculations.

To address this issue, use the `CONCAT () [1102]` function in your queries. The use of `CONCAT () [1102]` forces Excel to treat the value as a string, which Excel will then parse and usually correctly identify the embedded information.

However, even with this option, some data may be incorrectly formatted, even though the source data remains unchanged. Use the `Format Cells` option within Excel to change the format of the displayed information.

22.1.7.2.1.3. Microsoft Visual Basic

To be able to update a table, you must define a `primary key` for the table.

Visual Basic with ADO cannot handle big integers. This means that some queries like `SHOW PROCESSLIST` do not work properly. The fix is to use `OPTION=16384` in the ODBC connect string or to select the `Change BIGINT columns to INT` option in the Connector/ODBC connect screen. You may also want to select the `Return matching rows` option.

22.1.7.2.1.4. Microsoft Visual InterDev

If you have a `BIGINT` in your result, you may get the error `[Microsoft][ODBC Driver Manager] Driver does not support this parameter`. Try selecting the `Change BIGINT columns to INT` option in the Connector/ODBC connect screen.

22.1.7.2.1.5. Visual Objects

Select the `Don't optimize column widths` option.

22.1.7.2.1.6. Microsoft ADO

When you are coding with the ADO API and Connector/ODBC, you need to pay attention to some default properties that aren't supported by the MySQL server. For example, using the [CursorLocation Property](#) as `adUseServer` returns a result of `-1` for the [RecordCount Property](#). To have the right value, you need to set this property to `adUseClient`, as shown in the VB code here:

```
Dim myconn As New ADODB.Connection
Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long

myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount

myrs.Close
myconn.Close
```

Another workaround is to use a `SELECT COUNT(*)` statement for a similar query to get the correct row count.

To find the number of rows affected by a specific SQL statement in ADO, use the [RecordsAffected](#) property in the ADO execute method. For more information on the usage of execute method, refer to <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/hm/mdmthcnnextecute.asp>.

For information, see [ActiveX Data Objects\(ADO\) Frequently Asked Questions](#).

22.1.7.2.1.7. Using Connector/ODBC with Active Server Pages (ASP)

Select the [Return matching rows](#) option in the DSN.

For more information about how to access MySQL through ASP using Connector/ODBC, refer to the following articles:

- [Using MyODBC To Access Your MySQL Database Via ASP](#)
- [ASP and MySQL at DWAM.NT](#)

A Frequently Asked Questions list for ASP can be found at <http://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp>.

22.1.7.2.1.8. Using Connector/ODBC with Visual Basic (ADO, DAO and RDO) and ASP

Some articles that may help with Visual Basic and ASP:

- [MySQL BLOB columns and Visual Basic 6](#) by Mike Hillyer (<mike@openwin.org>).
- [How to map Visual basic data type to MySQL types](#) by Mike Hillyer (<mike@openwin.org>).

22.1.7.2.2. Using Connector/ODBC with Borland Applications

With all Borland applications where the Borland Database Engine (BDE) is used, follow these steps to improve compatibility:

- Update to BDE 3.2 or newer.
- Enable the [Don't optimize column widths](#) option in the DSN.

- Enabled the [Return matching rows](#) option in the DSN.

22.1.7.2.2.1. Using Connector/ODBC with Borland Builder 4

When you start a query, you can use the [Active](#) property or the [Open](#) method. Note that [Active](#) starts by automatically issuing a `SELECT * FROM ...` query. That may not be a good thing if your tables are large.

22.1.7.2.2.2. Using Connector/ODBC with Delphi

Also, here is some potentially useful Delphi code that sets up both an ODBC entry and a BDE entry for Connector/ODBC. The BDE entry requires a BDE Alias Editor that is free at a Delphi Super Page near you. (Thanks to Bryan Brunton <bryan@flesherfab.com> for this):

```
fReg:= TRegistry.Create;
fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
fReg.WriteString('Database', 'Documents');
fReg.WriteString('Description', ' ');
fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
fReg.WriteString('Flag', '1');
fReg.WriteString('Password', '');
fReg.WriteString('Port', ' ');
fReg.WriteString('Server', 'xmark');
fReg.WriteString('User', 'winuser');
fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
fReg.WriteString('DocumentsFab', 'MySQL');
fReg.CloseKey;
fReg.Free;

Memo1.Lines.Add('DATABASE NAME=');
Memo1.Lines.Add('USER NAME=');
Memo1.Lines.Add('ODBC DSN=DocumentsFab');
Memo1.Lines.Add('OPEN MODE=READ/WRITE');
Memo1.Lines.Add('BATCH COUNT=200');
Memo1.Lines.Add('LANGDRIVER=');
Memo1.Lines.Add('MAX ROWS=-1');
Memo1.Lines.Add('SCHEMA CACHE DIR=');
Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');

AliasEditor.Add('DocumentsFab', 'MySQL', Memo1.Lines);
```

22.1.7.2.2.3. Using Connector/ODBC with C++ Builder

Tested with BDE 3.0. The only known problem is that when the table schema changes, query fields are not updated. BDE, however, does not seem to recognize primary keys, only the index named [PRIMARY](#), although this has not been a problem.

22.1.7.2.3. Using Connector/ODBC with ColdFusion

The following information is taken from the ColdFusion documentation:

Use the following information to configure ColdFusion Server for Linux to use the [unixODBC](#) driver with Connector/ODBC for MySQL data sources. You can download Connector/ODBC at <http://dev.mysql.com/downloads/connector/odbc/>.

ColdFusion version 4.5.1 lets you use the ColdFusion Administrator to add the MySQL data source. However, the driver is not included with ColdFusion version 4.5.1. Before the MySQL driver appears in the ODBC data sources drop-down list, build and copy the Connector/ODBC driver to `/opt/coldfusion/lib/libmyodbc.so`.

The Contrib directory contains the program `mydsn-xxx.zip` which lets you build and remove the DSN registry file for the Connector/ODBC driver on ColdFusion applications.

For more information and guides on using ColdFusion and Connector/ODBC, see the following external sites:

- [Troubleshooting Data Sources and Database Connectivity for Unix Platforms](#).

22.1.7.2.4. Using Connector/ODBC with OpenOffice.org

Open Office (<http://www.openoffice.org>) [How-to: MySQL + OpenOffice](#). [How-to: OpenOffice + MyODBC + unixODBC](#).

22.1.7.2.5. Using Connector/ODBC with Sambar Server

Sambar Server (<http://www.sambarserver.info>) [How-to: MyODBC + SambarServer + MySQL](#).

22.1.7.2.6. Using Connector/ODBC with Pervasive Software DataJunction

You have to change it to output `VARCHAR` rather than `ENUM`, as it exports the latter in a manner that causes MySQL problems.

22.1.7.2.7. Using Connector/ODBC with SunSystems Vision

Select the `Return matching rows` option.

22.1.7.3. Connector/ODBC Errors and Resolutions (FAQ)

The following section details some common errors and their suggested fix or alternative solution. If you are still experiencing problems, use the Connector/ODBC mailing list; see [Section 22.1.8.1, "Connector/ODBC Community Support"](#).

Many problems can be resolved by upgrading your Connector/ODBC drivers to the latest available release. On Windows, make sure that you have the latest versions of the Microsoft Data Access Components (MDAC) installed.

64-Bit Windows and ODBC Data Source Administrator

I have installed Connector/ODBC on Windows XP x64 Edition or Windows Server 2003 R2 x64. The installation completed successfully, but the Connector/ODBC driver does not appear in [ODBC Data Source Administrator](#).

This is not a bug, but is related to the way Windows x64 editions operate with the ODBC driver. On Windows x64 editions, the Connector/ODBC driver is installed in the `%SystemRoot%\SysWOW64` folder. However, the default [ODBC Data Source Administrator](#) that is available through the [Administrative Tools](#) or [Control Panel](#) in Windows x64 Editions is located in the `%SystemRoot%\system32` folder, and only searches this folder for ODBC drivers.

On Windows x64 editions, use the ODBC administration tool located at `%SystemRoot%\SysWOW64\odbcad32.exe`, this will correctly locate the installed Connector/ODBC drivers and enable you to create a Connector/ODBC DSN.

This issue was originally reported as Bug #20301.

Error 10061 (Cannot connect to server)

When connecting or using the **Test** button in [ODBC Data Source Administrator](#) I get error 10061 (Cannot connect to server)

This error can be raised by a number of different issues, including server problems, network problems, and firewall and port blocking problems. For more information, see [Section C.5.2.2, "Can't connect to \[local\] MySQL server"](#).

"Transactions are not enabled" Error

The following error is reported when using transactions: `Transactions are not enabled`

This error indicates that you are trying to use [transactions](#) with a MySQL table that does not support transactions. Transactions are supported within MySQL when using the [InnoDB](#) database engine, which is the default storage engine in MySQL 5.5 and higher. In versions of MySQL before MySQL 5.1, you may also use the [BDB](#) engine.

Check the following before continuing:

- Verify that your MySQL server supports a transactional database engine. Use [SHOW ENGINES](#) to obtain a list of the available engine types.
- Verify that the tables you are updating use a transactional database engine.
- Ensure that you have not enabled the [disable transactions](#) option in your DSN.

#DELETED# Records Reported by Access

Access reports records as `#DELETED#` when inserting or updating records in linked tables.

If the inserted or updated records are shown as `#DELETED#` in Access, then:

- If you are using Access 2000, get and install the newest (version 2.6 or higher) Microsoft MDAC ([Microsoft Data Access Components](#)) from <http://support.microsoft.com/kb/110093>. This fixes a bug in Access that when you export data to MySQL, the table and column names aren't specified.

Also, get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5), which can be found at <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114>. This fixes some cases where columns are marked as `#DELETED#` in Access.
- For all versions of Access, enable the Connector/ODBC [Return matching rows](#) option. For Access 2.0, also enable the [Simulate ODBC 1.0](#) option.
- Include a [TIMESTAMP](#) in all tables that you want to be able to update.
- Include a [primary key](#) in the table. If not, new or updated rows may show up as `#DELETED#`.
- Use only [DOUBLE](#) float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as `#DELETED#` or that you cannot find or update rows.
- If you are using Connector/ODBC to link to a table that has a [BIGINT](#) column, the results are displayed as `#DELETED`. The work around solution is:
 - Have one more dummy column with [TIMESTAMP](#) as the data type.
 - Select the [Change BIGINT columns to INT](#) option in the connection dialog in ODBC DSN Administrator.

- Delete the table link from Access and re-create it.

Old records still display as #DELETED#, but newly added/updated records are displayed properly.

Write Conflicts or Row Location Errors

How do I handle Write Conflicts or Row Location errors?

If you see the following errors, select the [Return Matching Rows](#) option in the DSN configuration dialog, or specify `OPTION=2`, as the connection parameter:

```
Write Conflict. Another user has changed your data.  
  
Row cannot be located for updating. Some values may have been changed  
since it was last read.
```

Importing from Access 97

Exporting data from Access 97 to MySQL reports a [Syntax Error](#).

This error is specific to Access 97 and versions of Connector/ODBC earlier than 3.51.02. Update to the latest version of the Connector/ODBC driver to resolve this problem.

Importing from Microsoft DTS

Exporting data from Microsoft DTS to MySQL reports a [Syntax Error](#).

This error occurs only with MySQL tables using the [TEXT](#) or [VARCHAR](#) data types. You can fix this error by upgrading your Connector/ODBC driver to version 3.51.02 or higher.

SQL_NO_DATA Exception from ODBC.NET

Using ODBC.NET with Connector/ODBC, while fetching empty string (0 length), it starts giving the [SQL_NO_DATA](#) exception.

You can get the patch that addresses this problem from <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q319243>.

Error with SELECT COUNT(*)

Using `SELECT COUNT(*) FROM tbl_name` within Visual Basic and ASP returns an error.

This error occurs because the `COUNT(*) [1204]` expression is returning a [BIGINT](#), and ADO cannot make sense of a number this big. Select the [Change BIGINT columns to INT](#) option (option value 16384).

Multiple-Step Operation Error

Using the `AppendChunk()` or `GetChunk()` ADO methods, the [Multiple-step operation generated errors. Check each status value](#) error is returned.

The `GetChunk()` and `AppendChunk()` methods from ADO do not work as expected when the cursor location is specified as `adUseServer`. On the other hand, you can overcome this error by using `adUseClient`.

A simple example can be found from http://www.dwam.net/iishelp/ado/docs/adomth02_4.htm

Modified Record Error

Access returns [Another user had modified the record that you have modified while editing records on a Linked Table](#).

In most cases, this can be solved by doing one of the following things:

- Add a [primary key](#) for the table if one doesn't exist.
- Add a timestamp column if one doesn't exist.
- Only use double-precision float fields. Some programs may fail when they compare single-precision floats.

If these strategies do not help, start by making a log file from the ODBC manager (the log you get when requesting logs from ODBCADMIN) and a Connector/ODBC log to help you figure out why things go wrong. For instructions, see [Section 22.1.4.8, "Getting an ODBC Trace File"](#).

Direct Application Linking Under Unix or Linux

When linking an application directly to the Connector/ODBC library under Unix or Linux, the application crashes.

Connector/ODBC under Unix or Linux is not compatible with direct application linking. To connect to an ODBC source, use a driver manager, such as [iODBC](#) or [unixODBC](#).

Microsoft Office and DATE or TIMESTAMP Columns

Applications in the Microsoft Office suite cannot update tables that have [DATE](#) or [TIMESTAMP](#) columns.

This is a known issue with Connector/ODBC. Ensure that the field has a default value (rather than [NULL](#)) and that the default value is nonzero (that is, something other than [0000-00-00 00:00:00](#)).

INFORMATION_SCHEMA Database

When connecting Connector/ODBC 5.x to a MySQL 4.x server, the error [1044 Access denied for user 'xxx'@'%' to database 'information_schema'](#) is returned.

Connector/ODBC 5.x is designed to work with MySQL 5.0 or later, taking advantage of the [INFORMATION_SCHEMA](#) database to determine data definition information. Support for MySQL 4.1 is planned for the final release.

S1T00 Error

When calling [SQLTables](#), the error [S1T00](#) is returned, but I cannot find this in the list of error numbers for Connector/ODBC.

The [S1T00](#) error indicates that a general timeout has occurred within the ODBC system and is not a MySQL error. Typically it indicates that the connection you are using is stale, the server is too busy to accept your request or that the server has gone away.

"Table does not exist" Error in Access 2000

When linking to tables in Access 2000 and generating links to tables programmatically, rather than through the table designer interface, you may get errors about tables not existing.

There is a known issue with a specific version of the [msjet40.dll](#) that exhibits this issue. The version affected is 4.0.9025.0. Reverting to an older version will enable you to create the links. If you have recently updated your version, check your [WINDOWS](#) directory for the older version of the file and copy it to the drivers directory.

Batched Statements

When I try to use batched statements, the execution of the batched statements fails.

Batched statement support was added in 3.51.18. Support for batched statements is not enabled by default. Enable option `FLAG_MULTI_STATEMENTS`, value 67108864, or select the **Allow multiple statements** flag within a GUI configuration.

Packet Errors with ADODB and Excel

When connecting to a MySQL server using ADODB and Excel, occasionally the application fails to communicate with the server and the error `Got an error reading communication packets` appears in the error log.

This error may be related to Keyboard Logger 1.1 from PanteraSoft.com, which is known to interfere with the network communication between MySQL Connector/ODBC and MySQL.

Outer Join Error

When using some applications to access a MySQL server using Connector/ODBC and outer joins, an error is reported regarding the Outer Join Escape Sequence.

This is a known issue with MySQL Connector/ODBC which is not correctly parsing the "Outer Join Escape Sequence", as per the specs at [Microsoft ODBC Specs](#). Currently, Connector/ODBC will return a value > 0 when asked for `SQL_OJ_CAPABILITIES` even though no parsing takes place in the driver to handle the outer join escape sequence.

Hebrew/CJK Characters

I can correctly store extended characters in the database (Hebrew/CJK) using Connector/ODBC 5.1, but when I retrieve the data, the text is not formatted correctly and I get garbled characters.

When using ASP and UTF8 characters, add the following to your ASP files to ensure that the data returned is correctly encoded:

```
Response.CodePage = 65001
Response.CharSet = "utf-8"
```

Duplicate Entry in Installed Programs List

I have a duplicate MySQL Connector/ODBC entry within my **Installed Programs** list, but I cannot delete one of them.

This problem can occur when you upgrade an existing Connector/ODBC installation, rather than removing and then installing the updated version.



Warning

To fix the problem, use any working uninstallers to remove existing installations; then may have to edit the contents of the registry. Make sure you have a backup of your registry information before attempting any editing of the registry contents.

Values Truncated to 255 Characters

When submitting queries with parameter binding using `UPDATE`, my field values are being truncated to 255 characters.

Ensure that the `FLAG_BIG_PACKETS` option is set for your connection. This removes the 255 character limitation on bound parameters.

Disabling Data-At-Execution

Is it possible to disable data-at-execution using a flag?

If you do not want to use data-at-execution, remove the corresponding calls. For example:

```
SQLLEN ylen = SQL_LEN_DATA_AT_EXEC(10);
SQLBindCol(hstmt,2,SQL_C_BINARY, buf, 10, &ylen);
```

Would become:

```
SQLBindCol(hstmt,2,SQL_C_BINARY, buf, 10, NULL);
```

Note that in the call to `SQLBindCol()`, `&ylen` has been replaced by `NULL`.

For further information, refer to the [MSDN documentation](#) for `SQLBindCol()`.

NULLABLE Attribute for AUTO_INCREMENT Columns

When you call `SQLColumns()` for a table column that is `AUTO_INCREMENT`, the `NULLABLE` column of the result set is always `SQL_NULLABLE (1)`.

This is because MySQL reports the `DEFAULT` value for such a column as `NULL`. It means, if you insert a `NULL` value into the column, you will get the next integer value for the table's `auto_increment` counter.

22.1.8. Connector/ODBC Support

There are many different places where you can get support for using Connector/ODBC. Always try the Connector/ODBC Mailing List or Connector/ODBC Forum. See [Section 22.1.8.1, “Connector/ODBC Community Support”](#), for help before reporting a specific bug or issue to MySQL.

22.1.8.1. Connector/ODBC Community Support

Oracle provides assistance to the user community by means of its mailing lists. For Connector/ODBC-related issues, you can get help from experienced users by using the `<myodbc@lists.mysql.com>` mailing list. Archives are available online at <http://lists.mysql.com/myodbc>.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [Section 1.6.1, “MySQL Mailing Lists”](#).

Community support from experienced users is also available through the [ODBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [Section 1.6.2, “MySQL Community Support at the MySQL Forums”](#).

22.1.8.2. How to Report Connector/ODBC Problems or Bugs

If you encounter difficulties or problems with Connector/ODBC, start by making a log file from the [ODBC Manager](#) (the log you get when requesting logs from `ODBC ADMIN`) and Connector/ODBC. The procedure for doing this is described in [Section 22.1.4.8, “Getting an ODBC Trace File”](#).

Check the Connector/ODBC trace file to find out what could be wrong. Determine what statements were issued by searching for the string `>mysql_real_query` in the `myodbc.log` file.

Also, try issuing the statements from the `mysql` client program or from `admindemo`. This helps you determine whether the error is in Connector/ODBC or MySQL.

If you find out something is wrong, please only send the relevant rows (maximum 40 rows) to the `myodbc` mailing list. See [Section 1.6.1, “MySQL Mailing Lists”](#). Please never send the whole Connector/ODBC or ODBC log file!

Ideally, include the following information with the email:

- Operating system and version
- Connector/ODBC version
- ODBC Driver Manager type and version
- MySQL server version
- ODBC trace from Driver Manager
- Connector/ODBC log file from Connector/ODBC driver
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem!

Also, before posting the bug, check the MyODBC mailing list archive at <http://lists.mysql.com/myodbc>.

If you are unable to find out what is wrong, the last option is to create an archive in `tar` or `zip` format that contains a Connector/ODBC trace file, the ODBC log file, and a `README` file that explains the problem. Initiate a bug report for our bugs database at <http://bugs.mysql.com/>, then click the Files tab in the bug report for instructions on uploading the archive to the bugs database. Only MySQL engineers have access to the files you upload, and we are very discreet with the data.

If you can create a program that also demonstrates the problem, please include it in the archive as well.

If the program works with another SQL server, include an ODBC log file where you perform exactly the same SQL statements so that we can compare the results between the two systems.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

22.1.8.3. How to Submit a Connector/ODBC Patch

You can send a patch or suggest a better solution for any existing code or problems by sending a mail message to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com).

22.2. MySQL Connector/Net

Connector/Net lets you easily develop .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET-aware tools. Developers can build applications using their choice of .NET languages. Connector/Net is a fully managed ADO.NET driver written in 100% pure C#.

Connector/Net includes full support for:

- Features provided by MySQL Server up to and including MySQL Server version 5.5.
- Large-packet support for sending and receiving rows and BLOBs up to 2 gigabytes in size.
- Protocol compression, which enables compressing the data stream between the client and server.
- Connections using TCP/IP sockets, named pipes, or shared memory on Windows.
- Connections using TCP/IP sockets or Unix sockets on Unix.
- The Open Source Mono framework developed by Novell.

- Fully managed, does not utilize the MySQL client library.

This document is intended as a user's guide to Connector/Net and includes a full syntax reference. Syntax information is also included within the [Documentation.chm](#) file included with the Connector/Net distribution.

If you are using MySQL 5.0 or later, and Visual Studio as your development environment, you can also use the MySQL Visual Studio Plugin. The plugin acts as a DDEX (Data Designer Extensibility) provider: you can use the data design tools within Visual Studio to manipulate the schema and objects within a MySQL database. For more information, see [Section 22.2.3, "Connector/Net Visual Studio Integration"](#).



Note

Connector/Net 5.1.2 and later include the Visual Studio Plugin by default.

MySQL Connector/Net supports full versions of Visual Studio 2005, 2008, and 2010, although certain features are only available in Visual Studio 2010 when using MySQL Connector/Net version 6.3.2 and later. Note that MySQL Connector/Net does not currently support Express versions of Microsoft products, including Microsoft Visual Web Developer.

Key topics:

- For connection string properties when using the `MySqlConnection` class, see [Section 22.2.6, "Connector/Net Connection String Options Reference"](#).

For release notes detailing the changes in each release of Connector/Net, see [MySQL Connector/Net Release Notes](#).

As of Connector/Net 5.1.2, the Visual Studio Plugin is part of the main Connector/Net package. However, even for versions of Connector/Net older than 5.1.2, the Visual Studio Plugin release notes can be found at [MySQL Connector/Net Release Notes](#).

22.2.1. Connector/Net Versions

There are several versions of Connector/Net available:

- Connector/Net 6.6 includes support for MySQL Server 5.6, 5.5, 5.1, and 5.0. Important new features include stored procedure debugging in Microsoft Visual Studio, support for pluggable authentication including the ability to write your own authentication plugin, Entity Framework 4.3 Code First support, and enhancements to partial trust support to allow hosting services to deploy applications without installing the Connector/Net library in the GAC.
- Connector/Net 6.5 includes support for MySQL Server 5.6, 5.5, 5.1, and 5.0. Important new features include interceptor classes for exceptions and commands, support for the MySQL 5.6 fractional seconds feature, better partial-trust support, and better IntelliSense, including auto-completion when editing stored procedures or `.mysql` files.
- Connector/Net 6.4 includes support for MySQL Server 5.5, 5.1, and 5.0. Important new features include support for Windows authentication (when connecting to MySQL Server 5.5+), table caching on the client side, simple connection fail-over support, and improved SQL generation from the Entity Framework provider.
- Connector/Net 6.3 includes support for MySQL Server 5.5, 5.1, and 5.0. Important new features include integration with Visual Studio 2010, such as availability of DDL T4 template for Entity Framework, and a custom MySQL SQL Editor. Other features include refactored transaction scope: Connector/Net now supports nested transactions in a scope where they use the same connection string.

- Connector/Net 6.2 includes support for MySQL Server 5.5, 5.1, 5.0, and 4.1. Important new features include a new logging system and client SSL certificates.
- Connector/Net 6.1 includes support for MySQL Server 5.5, 5.1, 5.0, and 4.1. Important new features include the MySQL Website Configuration Tool and a Session State Provider.
- Connector/Net 6.0 includes support for MySQL Server 5.5, 5.1, 5.0, and 4.1.

This version of Connector/Net is no longer supported.

- Connector/Net 5.2 includes support for MySQL Server 5.5, 5.1, 5.0, and 4.1 features. Connector/Net 5.2 also includes support for a new membership/role provider, Compact Framework 2.0, a new stored procedure parser and improvements to [GetSchema](#). Connector/Net 5.2 also includes the Visual Studio Plugin as a standard installable component.

This version of Connector/Net is no longer supported.

- Connector/Net 5.1 includes support for MySQL Server 5.5, 5.1, 5.0, 4.1, and 4.0 features. Connector/Net 5.1 also includes support for a new membership/role provider, Compact Framework 2.0, a new stored procedure parser and improvements to [GetSchema](#). Connector/Net 5.1 also includes the Visual Studio Plugin as a standard installable component.

This version of Connector/Net is no longer supported.

- Connector/Net 5.0 includes support for MySQL Server 5.1, 5.0, 4.1 and 4.0 features. Connector/Net 5.0 also includes full support for the ADO.NET 2.0 interfaces and subclasses, includes support for the usage advisor and performance monitor (PerfMon) hooks.

This version of Connector/Net is no longer supported.

- Connector/Net 1.0 includes support for MySQL Server 5.0, 4.1, and 4.0 features, and full compatibility with the ADO.NET driver interface.

This version of Connector/Net is no longer supported.

The latest source code for Connector/Net can be downloaded from the MySQL public Subversion server. For further details, see [Section 22.2.2.3, “Installing Connector/Net from the Source Code”](#).

The following table shows the .NET Framework version required, and the MySQL Server version supported by Connector/Net:

Table 22.19. Connector/Net Requirements for Related Products

Connector/Net version	ADO.NET version supported	.NET Framework version required	MySQL Server version supported	Currently supported
6.6	2.x+	2.x+, 4.x+ for VS 2010 support	5.6, 5.5, 5.1, 5.0	Yes
6.5	2.x+	2.x+, 4.x+ for VS 2010 support	5.6, 5.5, 5.1, 5.0	Yes
6.4	2.x+	2.x+, 4.x+ for VS 2010 support	5.6, 5.5, 5.1, 5.0	Yes
6.3	2.x+	2.x+, 4.x+ for VS 2010 support	5.6, 5.5, 5.1, 5.0	Yes
6.2	2.x+	2.x+	5.6, 5.5, 5.1, 5.0, 4.1	Yes

Connector/Net version	ADO.NET version supported	.NET Framework version required	MySQL Server version supported	Currently supported
6.1	2.x+	2.x+	5.6, 5.5, 5.1, 5.0, 4.1	Yes
6.0	2.x+	2.x+	5.5, 5.1, 5.0, 4.1	Critical issues only
5.2	2.x+	2.x+	5.5, 5.1, 5.0, 4.1	No
5.1	2.x+	2.x+	5.5, 5.1, 5.0, 4.1, 4.0	No
5.0	2.x+	2.x+	5.0, 4.1, 4.0	No
1.0	1.x	1.x	5.0, 4.1, 4.0	No



Note

Version numbers for MySQL products are formatted as X.Y.Z, where Z=0 indicates alpha, Z=1 indicates beta, and Z>=2 indicates GA. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.YY.ZZ. For example, the official MySQL formatted version number 5.0.9 may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display format is different.

22.2.2. Installing Connector/Net

Connector/Net runs on any platform that supports the .NET framework. The .NET framework is primarily supported on recent versions of Microsoft Windows, and is supported on Linux through the Open Source Mono framework (see <http://www.mono-project.com>).

Connector/Net is available for download from <http://dev.mysql.com/downloads/connector/net/>.

22.2.2.1. Installing Connector/Net on Windows

On Windows, you can install either through a binary installation process or by downloading a zip file with the Connector/Net components.

Before installing, ensure that your system is up to date, including installing the latest version of the .NET Framework.

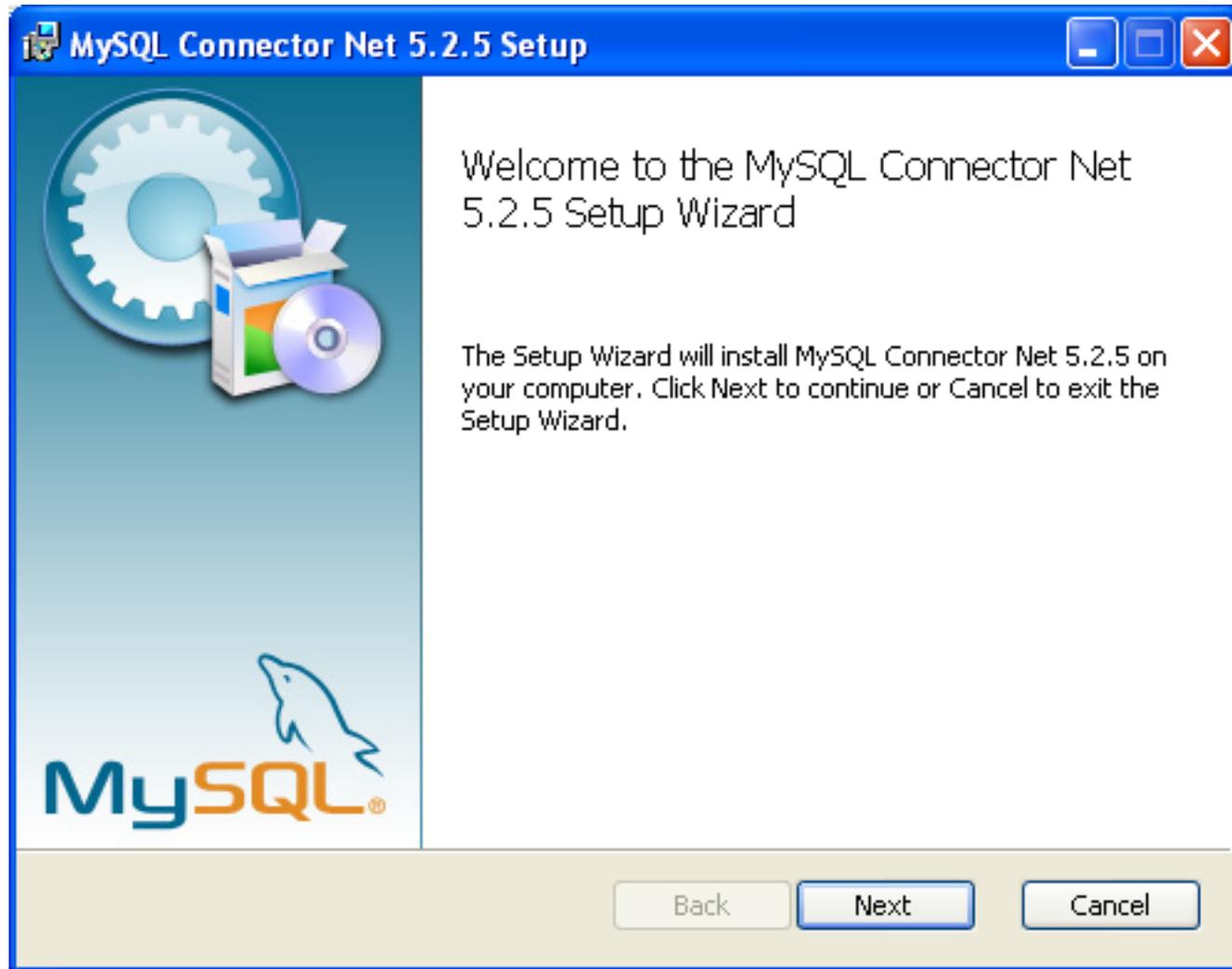
22.2.2.1.1. Installing Connector/Net using the Installer

Using the installer is the most straightforward method of installing Connector/Net on Windows and the installed components include the source code, test code and full reference documentation.

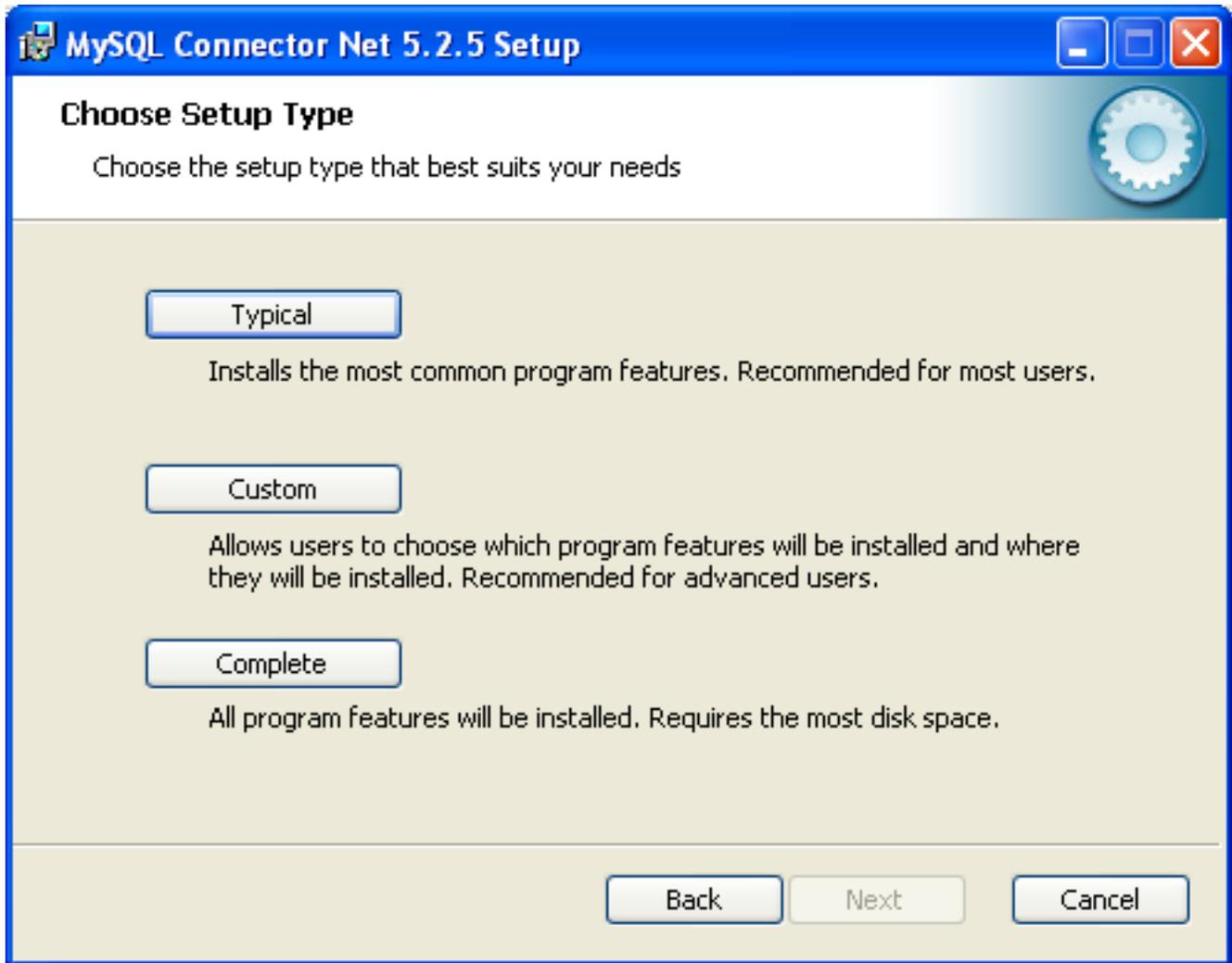
You install Connector/Net through a Windows Installer (.msi) installation package, which can install Connector/Net on all Windows operating systems. The MSI package is contained within a zip archive named `mysql-connector-net-version.zip`, where `version` indicates the Connector/Net version.

To install Connector/Net:

1. Double-click the MSI installer file extracted from the zip you downloaded. Click **Next** to start the installation.



2. You must choose the type of installation to perform.



For most situations, the Typical installation is suitable. Click the **Typical** button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the **Complete** button and proceed to step 5. To customize your installation, including choosing the components to install and some installation options, click the **Custom** button and proceed to Step 3.

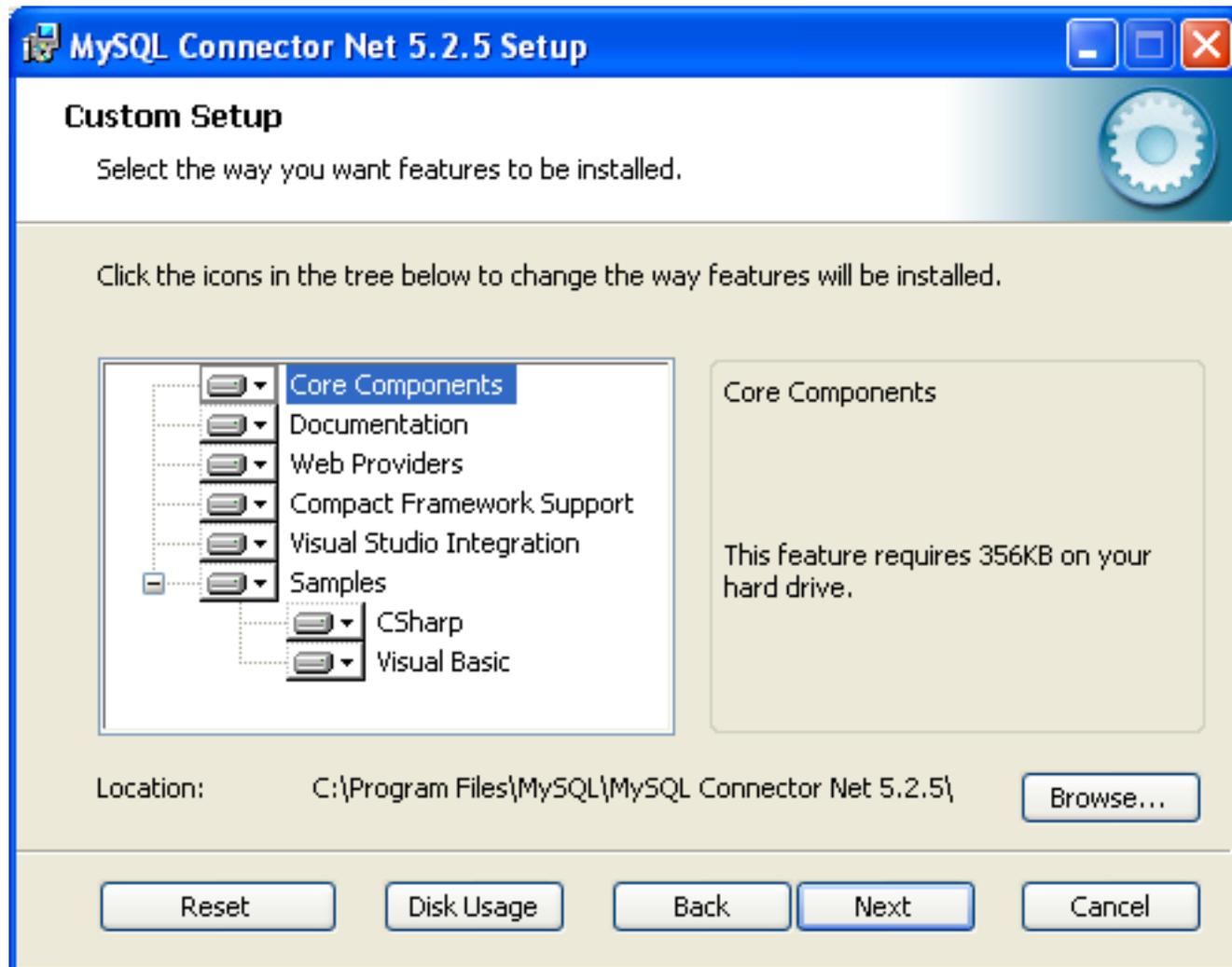
The Connector/Net installer will register the connector within the Global Assembly Cache (GAC) - this will make the Connector/Net component available to all applications, not just those where you explicitly reference the Connector/Net component. The installer will also create the necessary links in the Start menu to the documentation and release notes.

3. If you have chosen a custom installation, you can select the individual components to install, including the core interface component, supporting documentation (a CHM file) samples and examples, and the source code. Select the items, and their installation level, and then click **Next** to continue the installation.

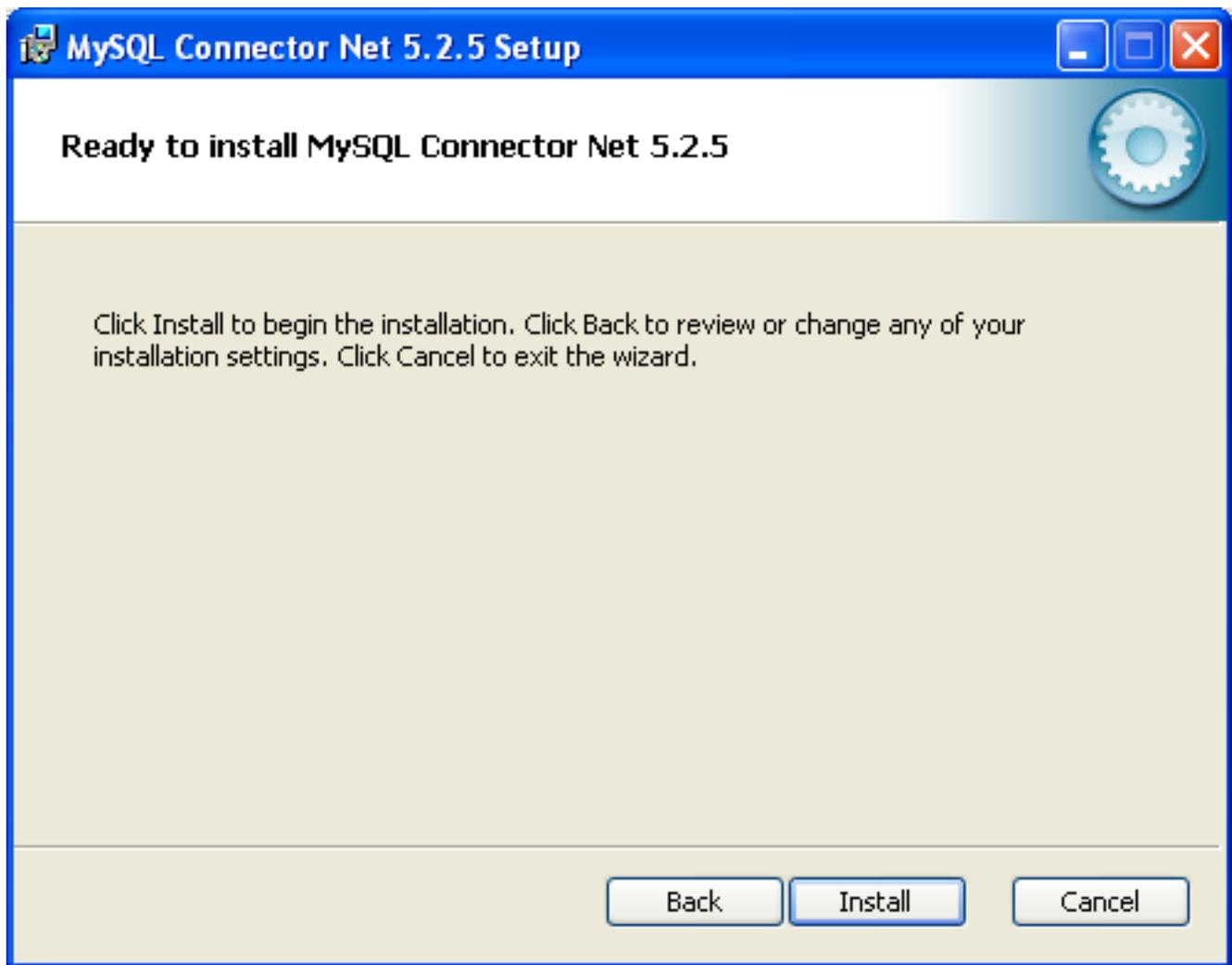
**Note**

For Connector/Net 1.0.8 or lower and Connector 5.0.4 and lower the installer will attempt to install binaries for both 1.x and 2.x of the .NET Framework. If you only have one version of the framework installed, the connector installation

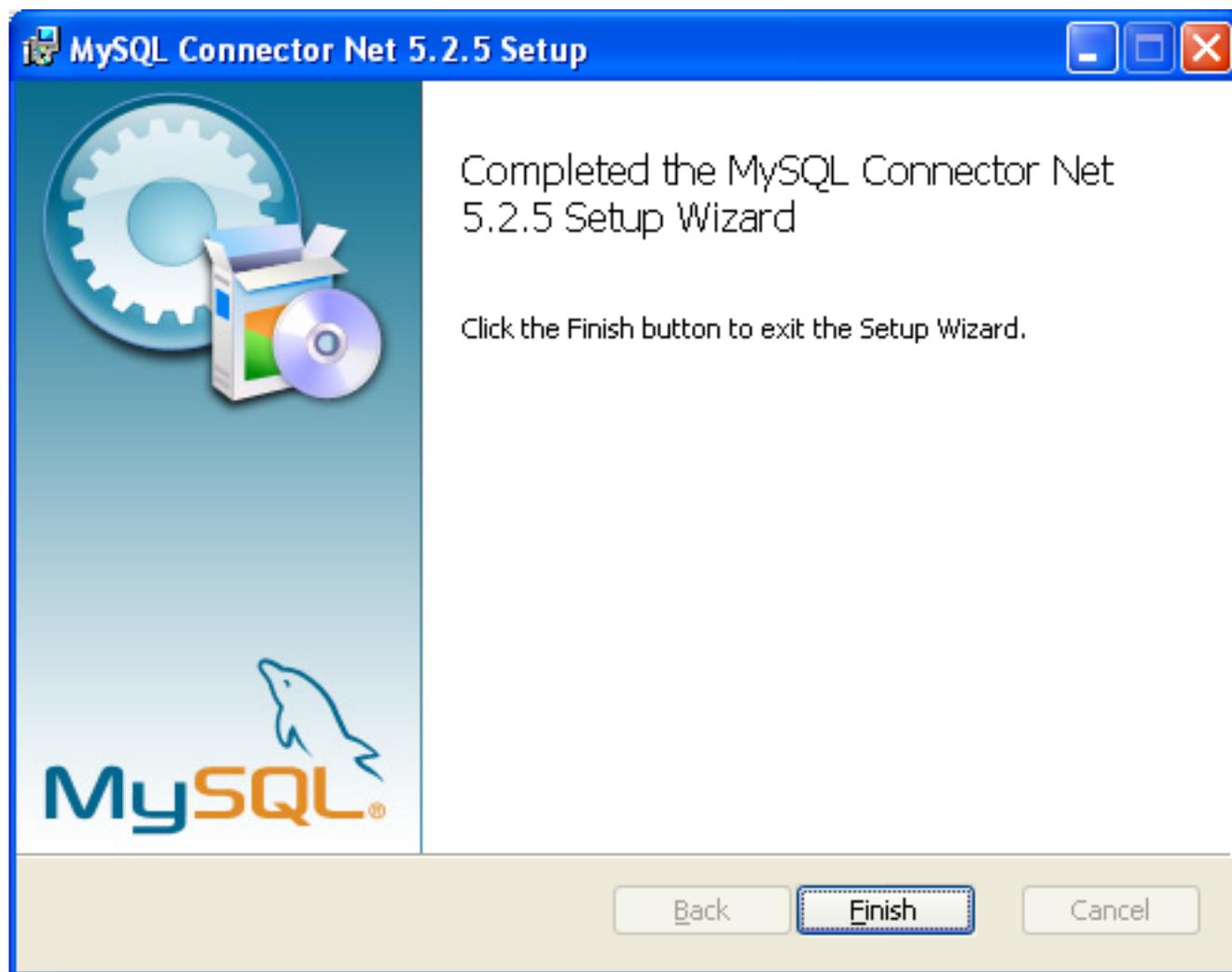
may fail. If this happens, you can choose the framework version to be installed through the custom installation step.



4. You will be given a final opportunity to confirm the installation. Click **Install** to copy and install the files onto your machine.



5. Once the installation has been completed, click **Finish** to exit the installer.



Unless you choose otherwise, Connector/Net is installed in `C:\Program Files\MySQL\MySQL Connector Net X.X.X`, where `X.X.X` is replaced with the version of Connector/Net you are installing. New installations do not overwrite existing versions of Connector/Net.

Depending on your installation type, the installed components will include some or all of the following components:

- `bin`: Connector/Net MySQL libraries for different versions of the .NET environment.
- `docs`: Connector/Net documentation in CHM format.
- `samples`: Sample code and applications that use the Connector/Net component.
- `src`: The source code for the Connector/Net component.

You may also use the `/quiet` or `/q` command-line option with the `msiexec` tool to install the Connector/Net package automatically (using the default options) with no notification to the user. Using this method the user cannot select options. Additionally, no prompts, messages or dialog boxes will be displayed.

```
C:\> msiexec /package connector-net.msi /quiet
```

To provide a progress bar to the user during automatic installation, use the `/passive` option.

22.2.2.1.2. Installing Connector/Net Using the Zip Packages

If you have problems running the installer, you can download a zip file without an installer as an alternative. That file is called `mysql-connector-net-version-noinstall.zip`. After downloading the zip archive, extract the files to a location of your choice.

The file contains the following directories:

- `bin`: Connector/Net MySQL libraries for different versions of the .NET environment.
- `Docs`: Connector/Net documentation in CHM format.
- `Samples`: Sample code and applications that use the Connector/Net component.

Connector/Net 6.0.x has a different directory structure:

- `Assemblies`: A collection of DLLs that make up the connector functionality.
- `Documentation`: Connector/Net documentation in CHM format.
- `Samples`: sample code and applications that use the Connector/Net component.

Another zip file available for download contains the source code distribution. This file is named `mysql-connector-net-version-src.zip`.

The file contains the following directories:

- `Documentation`: Source files to build the documentation into the compiled HTML (CHM) format.
- `Installer`: Source files to build the Connector/Net installer program.
- `MySQL.Data`: Source files for the core data provider.
- `MySQL.VisualStudio`: Source files for the Microsoft Visual Studio extensions.
- `MySQL.Web`: Source files for the web providers. This includes code for the membership provider, role provider and profile provider. These are used in ASP.NET web sites.
- `Samples`: Source files for several example applications.
- `Tests`: A spreadsheet listing test cases.
- `VisualStudio`: Resources used by the Visual Studio plugin.

Finally, ensure that `MySQL.Data.dll` is accessible to your program at build time (and run time). If using Microsoft Visual Studio, add `MySQL.Data` as a Reference to your project.



Note

If using MySQL Connector/Net 6.3.5 and above, the `MySQL.Data.dll` file provided will work with both .NET Framework 2.x and 4.x.

22.2.2.2. Installing Connector/Net on Unix with Mono

There is no installer available for installing the Connector/Net component on your Unix installation. Before installing, ensure that you have a working Mono project installation. To test whether your system has Mono installed, enter:

```
shell> mono --version
```

The version of the Mono JIT compiler is displayed.

To compile C# source code, make sure a Mono C# compiler is installed. Note that there are two Mono C# compilers available, `mcs`, which accesses the 1.0-profile libraries, and `gmcs`, which accesses the 2.0-profile libraries.

To install Connector/Net on Unix/Mono:

1. Download the `mysql-connector-net-version-noinstall.zip` and extract the contents to a directory of your choice, for example: `~/connector-net/`.
2. In the directory where you unzipped the connector to, change into the `bin` subdirectory. Ensure the file `MySQL.Data.dll` is present. This filename is case-sensitive.
3. You must register the Connector/Net component, `MySQL.Data`, in the Global Assembly Cache (GAC). In the current directory enter the `gacutil` command:

```
root-shell> gacutil /i MySQL.Data.dll
```

This will register `MySQL.Data` into the GAC. You can check this by listing the contents of `/usr/lib/mono/gac`, where you will find `MySQL.Data` if the registration has been successful.

You are now ready to compile your application. You must ensure that when you compile your application you include the Connector/Net component using the `-r:` command-line option. For example:

```
shell> gmcs -r:System.dll -r:System.Data.dll -r:MySQL.Data.dll HelloWorld.cs
```

Note, the assemblies that are referenced depend on the requirements of the application, but applications using Connector/Net must provide `-r:MySQL.Data` as a minimum.

You can further check your installation by running the compiled program, for example:

```
shell> mono HelloWorld.exe
```

22.2.2.3. Installing Connector/Net from the Source Code



Caution

Read this section only if you are interested in helping us test our new code. If you just want to get Connector/Net up and running on your system, use a standard release distribution.

Obtaining the Source Code

To obtain the most recent development source tree, first download and install Bazaar. You can obtain Bazaar from the [Bazaar VCS Website](#). Bazaar is supported by any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows or Mac OS X host. Instructions for downloading and installing Bazaar on the different platforms are available on the Bazaar Web site.

The most recent development source tree is available from our public Subversion trees at <http://dev.mysql.com/tech-resources/sources.html>.

To check out the Connector/Net sources, change to the directory where you want the copy of the Connector/Net tree to be stored, then use the following command:

```
shell> bzz branch lp:connectornet/trunk
```

To download a specific version of Connector/Net, specify the version number instead of `trunk`. For example, to obtain a copy of the 6.0 version of the source tree:

```
shell> bzz branch lp:connectornet/6.0
```

Source packages are also available on the [downloads page](#).

Building the Source Code on Windows

The following procedure can be used to build the connector on Microsoft Windows.

- Obtain the source code, either from the Subversion server, or through one of the prepared source code packages.
- Navigate to the root of the source code tree.
- A Microsoft Visual Studio 2005 solution file is available to build the connector, this is called `MySQL-VS2005.sln`. Click this file to load the solution into Visual Studio.
- Select **Build**, **Build Solution** from the main menu to build the solution.

Building the Source Code on Unix

Support for building Connector/Net on Mono/Unix is currently not available.

22.2.3. Connector/Net Visual Studio Integration

When MySQL Connector/Net is installed on Microsoft Windows, Visual Studio integration components are also installed and initialized. This enables the developer to work seamlessly with MySQL Connector/Net in the familiar Visual Studio environment, as described in the following sections of the manual.

MySQL Connector/Net supports Visual Studio versions 2005, 2008, and 2010. However, only MySQL Connector/Net version 6.3 and higher fully integrate with Visual Studio 2010, although applications using earlier versions of the connector can be built with the Visual Studio 2010 environment using .NET 2.x frameworks.

Visual Studio 2010 support was introduced with MySQL Connector/Net 6.3.2. From version 6.3.2 the connector ships with both .NET 2.x and .NET 4.x versions of the entity framework support files, `mysql.data.ef.dll` and `mysql.visualstudio.dll`. The .NET 4.x versions are required to enable new integration features supported in Visual Studio 2010, including:

- New DDL T4 template for the Entity Framework (EF) - This enables developers to design an EF model from scratch and use the native Visual Studio 2010 facility to generate MySQL DDL from that model. This is done by creating the model, and with the model open, choosing the SSDLToMySQL template in the properties window. The correct DDL is then generated. The developer can then save this code as a `.mysql` file in their project and execute it against the MySQL server.
- New SQL Editor - A new SQL editor lets you connect to a MySQL server to execute SQL. This is activated by creating a new file with a `.mysql` extension. A new template lets you create files with this file type using the Visual Studio 2010 main menu item **File**, **New**. Note that the MySQL SQL Editor is also available in Visual Studio 2005 and 2008.

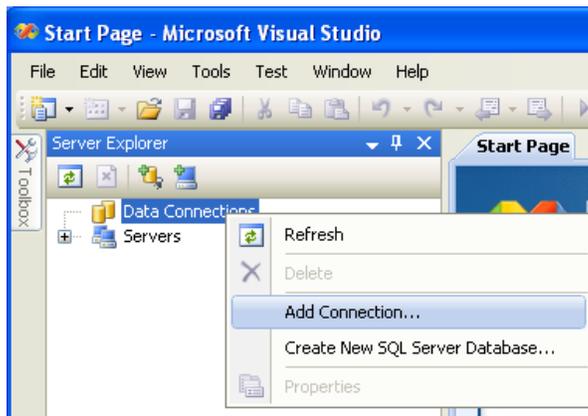
As of Connector/Net 5.1.2, the Visual Studio Plugin is part of the main Connector/Net package. However, even for versions of Connector/Net older than 5.1.2, the Visual Studio Plugin release notes can be found at [MySQL Connector/Net Release Notes](#).

22.2.3.1. Making a Connection

Once the connector is installed, you can use it to create, modify, and delete connections to MySQL databases. To create a connection with a MySQL database, perform the following steps:

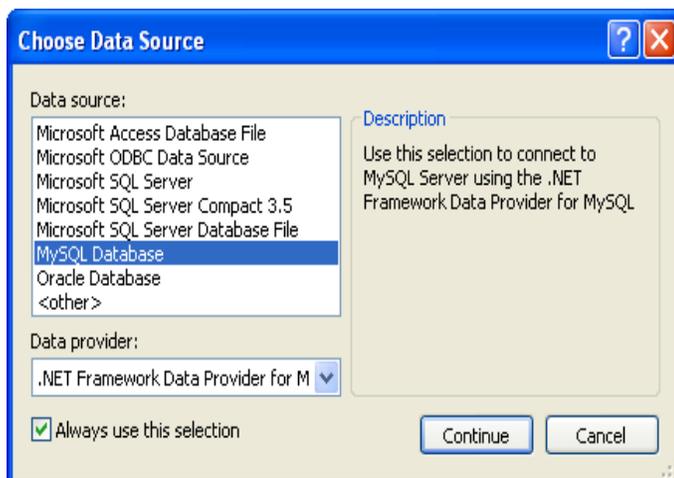
- Start Visual Studio, and open the Server Explorer window (View, Server Explorer option in the main Visual Studio menu, or **Control+W, L** keyboard shortcuts).
- Right-click the Data Connections node, and choose the **Add Connection...** menu item.
- Add Connection dialog opens. Press the **Change** button to choose MySQL Database as a data source.

Figure 22.1. Add Connection Context Menu

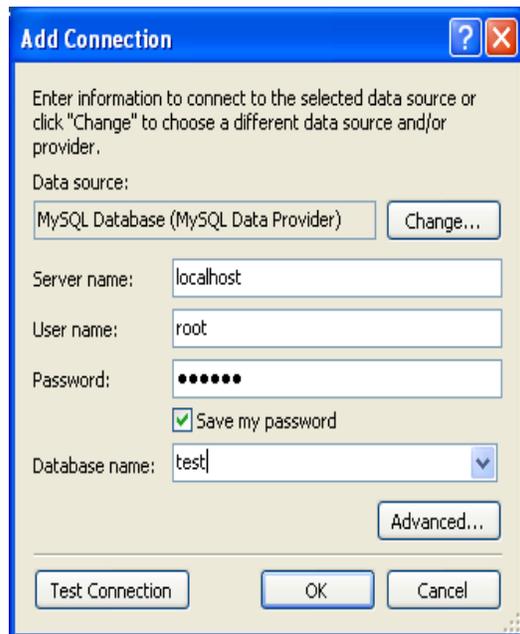


- **Change Data Source** dialog opens. Choose **MySQL Database** in the list of data sources (or the `<other>` option, if MySQL Database is absent), and then choose **.NET Framework Data Provider for MySQL** in the combo box of data providers.

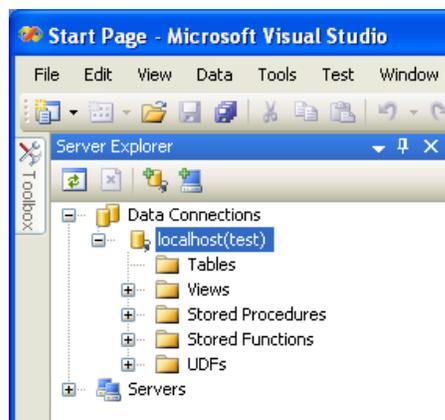
Figure 22.2. Choose Data Source



- Input the connection settings: the server host name (for example, localhost if the MySQL server is installed on the local machine), the user name, the password, and the default schema name. Note that you must specify the default schema name to open the connection.

Figure 22.3. Add Connection Dialog

- You can also set the port to connect with the MySQL server by pressing the **Advanced** button. To test connection with the MySQL server, set the server host name, the user name, and the password, and press the **Test Connection** button. If the test succeeds, the success confirmation dialog opens.
- After you set all settings and test the connection, press **OK**. The newly created connection is displayed in Server Explorer. Now you can work with the MySQL server through standard Server Explorer GUI.

Figure 22.4. New Data Connection

After the connection is successfully established, all settings are saved for future use. When you start Visual Studio for the next time, open the connection node in Server Explorer to establish a connection to the MySQL server again.

To modify and delete a connection, use the Server Explorer context menu for the corresponding node. You can modify any of the settings by overwriting the existing values with new ones. Note that the connection may be modified or deleted only if no active editor for its objects is opened: otherwise, you may lose your data.

22.2.3.2. Using IntelliSense in the SQL Editor

IntelliSense support is available starting in Connector/Net 6.5. Once you have established a connection, for example, using the **Connect to MySQL** toolbar button, you can get autocompletion as you type, or by pressing **Control+J**. Depending on the context, the autocompletion dialog can show the list of available tables, table columns, or stored procedures (with the routine's signature as a tooltip). Typing some characters before pressing **Control+J** filters the choices to those items starting with that prefix.

22.2.3.3. Editing Tables

Connector/Net contains a table editor, which enables the visual creation and modification of tables.

The Table Designer can be accessed through a mouse action on table-type node of Server Explorer. To create a new table, right-click the **Tables** node (under the connection node) and choose the **Create Table** command from the context menu.

To modify an existing table, double-click the node of the table to modify, or right-click this node and choose the **Design** item from the context menu. Either of the commands opens the Table Designer.

The table editor is implemented in the manner of the well-known Query Browser Table Editor, but with minor differences.

Figure 22.5. Editing New Table

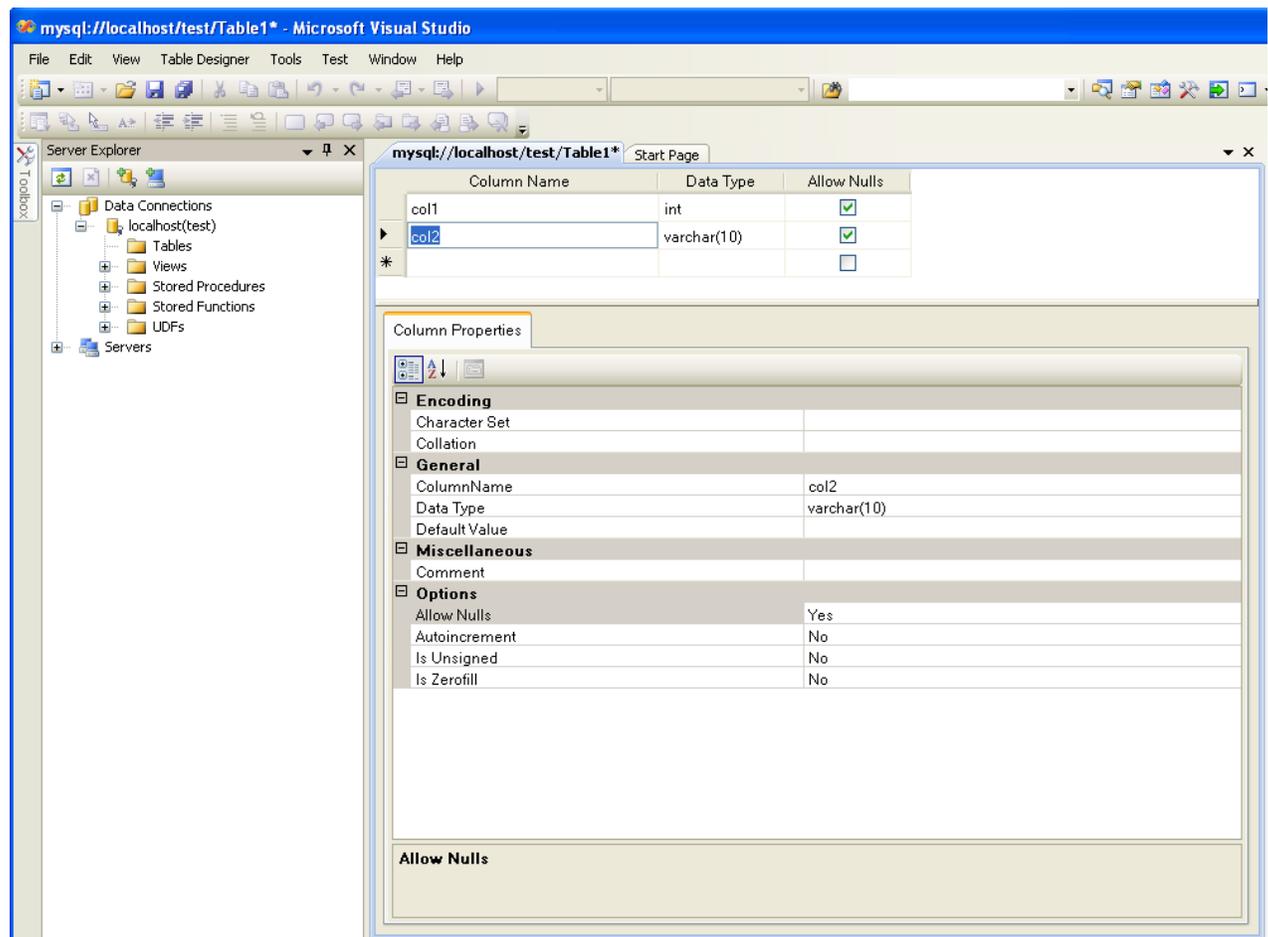


Table Designer consists of the following parts:

- Columns Editor - a data grid on top of the Table Designer. Use the Columns grid for column creation, modification, and deletion.
- Indexes tab - a tab on bottom of the Table Designer. Use the Indexes tab for indexes management.
- Foreign Keys tab - a tab on bottom of the Table Designer. Use the Foreign Keys tab for foreign keys management.
- Column Details tab - a tab on bottom of the Table Designer. Use the Column Details tab to set advanced column options.
- Properties window - a standard Visual Studio Properties window, where the properties of the edited table are displayed. Use the Properties window to set the table properties.

Each of these areas is discussed in more detail in subsequent sections.

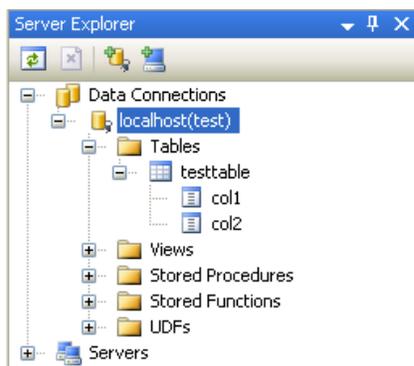
To save changes you have made in the Table Designer, use either **Save** or **Save All** button of the Visual Studio main toolbar, or press **Control+S**. If you have not already named the table, you will be prompted to do so.

Figure 22.6. Choose Table Name

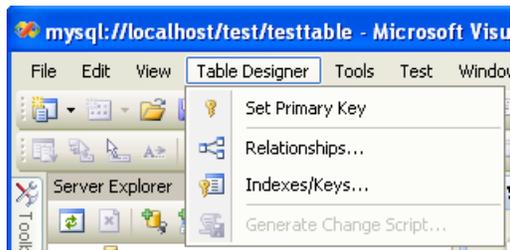


Once the table is created, you can view it in the Server Explorer.

Figure 22.7. Newly Created Table



The Table Designer main menu lets you set a [primary key](#) column, edit relationships such as [foreign keys](#), and create [indexes](#).

Figure 22.8. Table Designer Main Menu

22.2.3.3.1. Column Editor

You can use the Column Editor to set or change the name, data type, default value, and other properties of a table column. To set the focus to a needed cell of a grid, use the mouse click. Also you can move through the grid using **Tab** and **Shift+Tab** keys.

To set or change the name, data type, default value and comment of a column, activate the appropriate cell and type the desired value.

To set or unset flag-type column properties (**NOT NULL**, auto incremented, flags), check or uncheck the corresponding check boxes. Note that the set of column flags depends on its data type.

To reorder columns, index columns or foreign key columns in the Column Editor, select the whole column to reorder by clicking the selector column on the left of the column grid. Then move the column by using **Control+Up** (to move the column up) or **Control+Down** (to move the column down) keys.

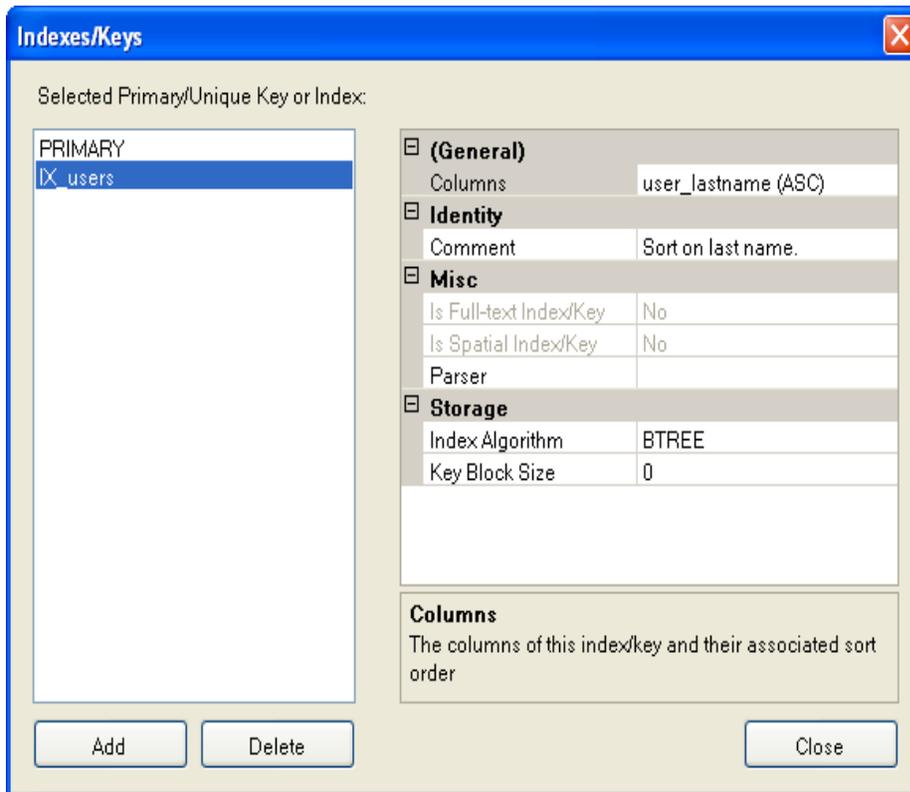
To delete a column, select it by clicking the selector column on the left of the column grid, then press the **Delete** button on a keyboard.

22.2.3.3.2. Editing Indexes

Indexes management is performed using the **Indexes/Keys** dialog.

To add an index, select **Table Designer, Indexes/Keys...** from the main menu, and click **Add** to add a new index. You can then set the index name, index kind, index type, and a set of index columns.

Figure 22.9. Indexes Dialog



To remove an index, select it in the list box on the left, and click the **Delete** button.

To change index settings, select the needed index in the list box on the left. The detailed information about the index is displayed in the panel on the right hand side. Change the desired values.

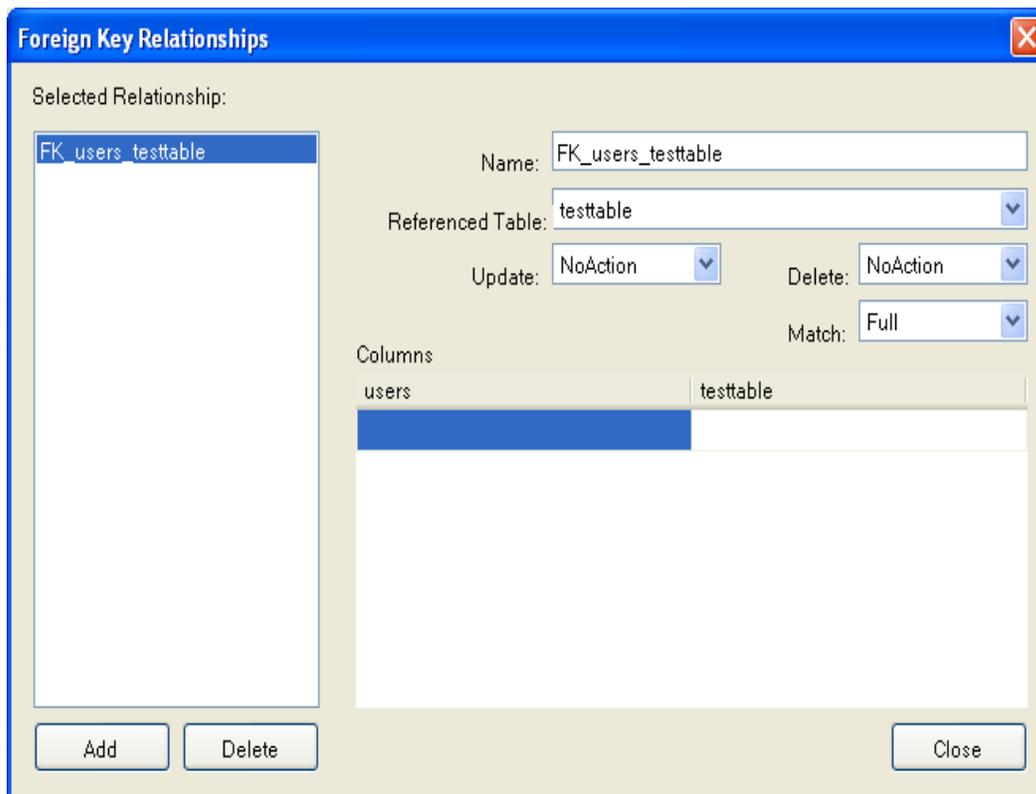
22.2.3.3.3. Editing Foreign Keys

You manage [foreign keys](#) for [InnoDB](#) tables using the **Foreign Key Relationships** dialog.

To add a foreign key, select **Table Designer, Relationships...** from the main menu. This displays the **Foreign Key Relationship** dialog. Click **Add**. You can then set the foreign key name, referenced table name, foreign key columns, and actions upon update and delete.

To remove a foreign key, select it in the list box on the left, and click the **Delete** button.

To change foreign key settings, select the required foreign key in the list box on the left. The detailed information about the foreign key is displayed in the right hand panel. Change the desired values.

Figure 22.10. Foreign Key Relationships Dialog

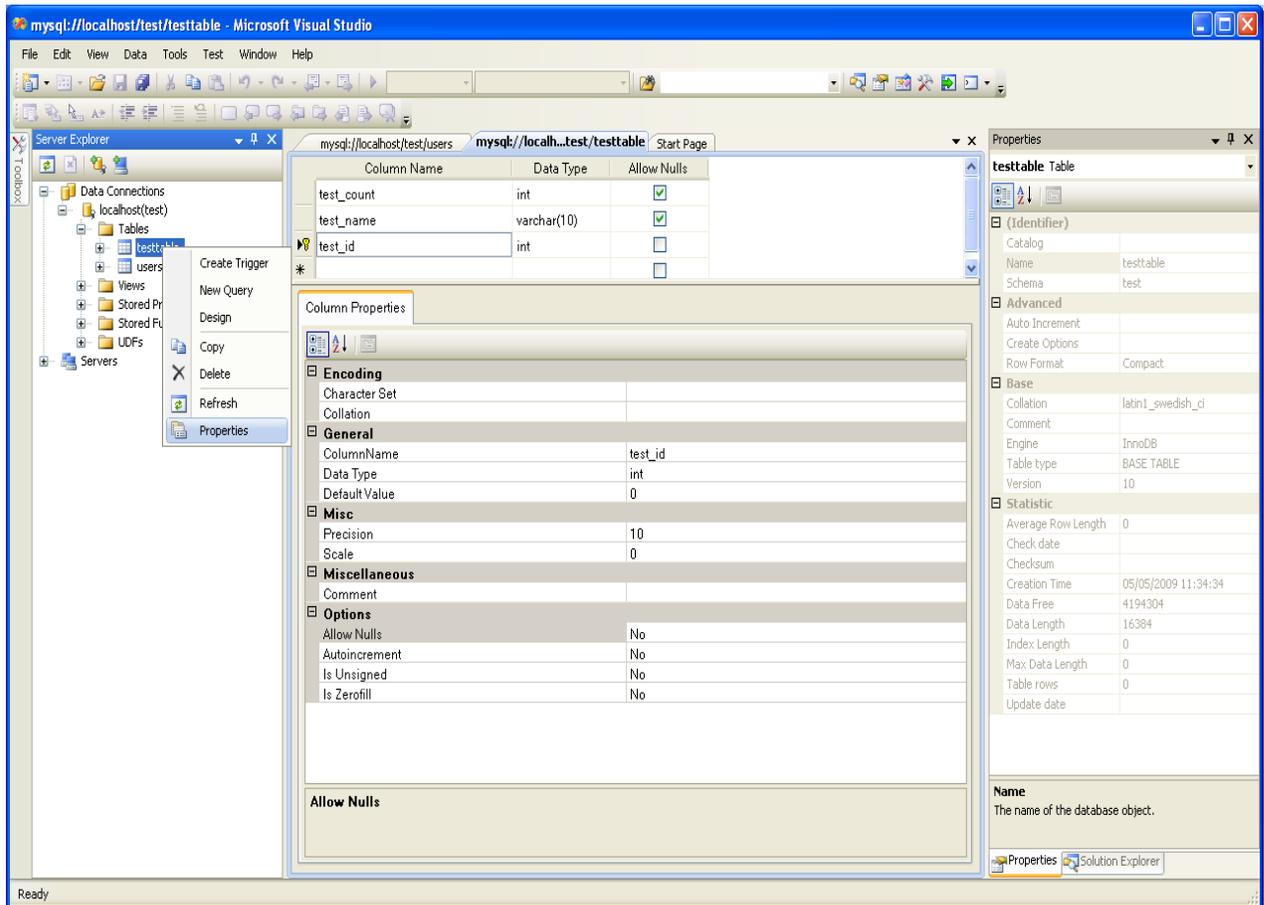
22.2.3.3.4. Column Properties

The **Column Properties** tab can be used to set column options. In addition to the general column properties presented in the Column Editor, in the **Column Properties** tab you can set additional properties such as Character Set, Collation and Precision.

22.2.3.3.5. Table Properties

To bring up Table Properties select the table and right-click to activate the context menu. Select Properties. The **Table Properties** dockable window will be displayed.

Figure 22.11. Table Properties Menu Item



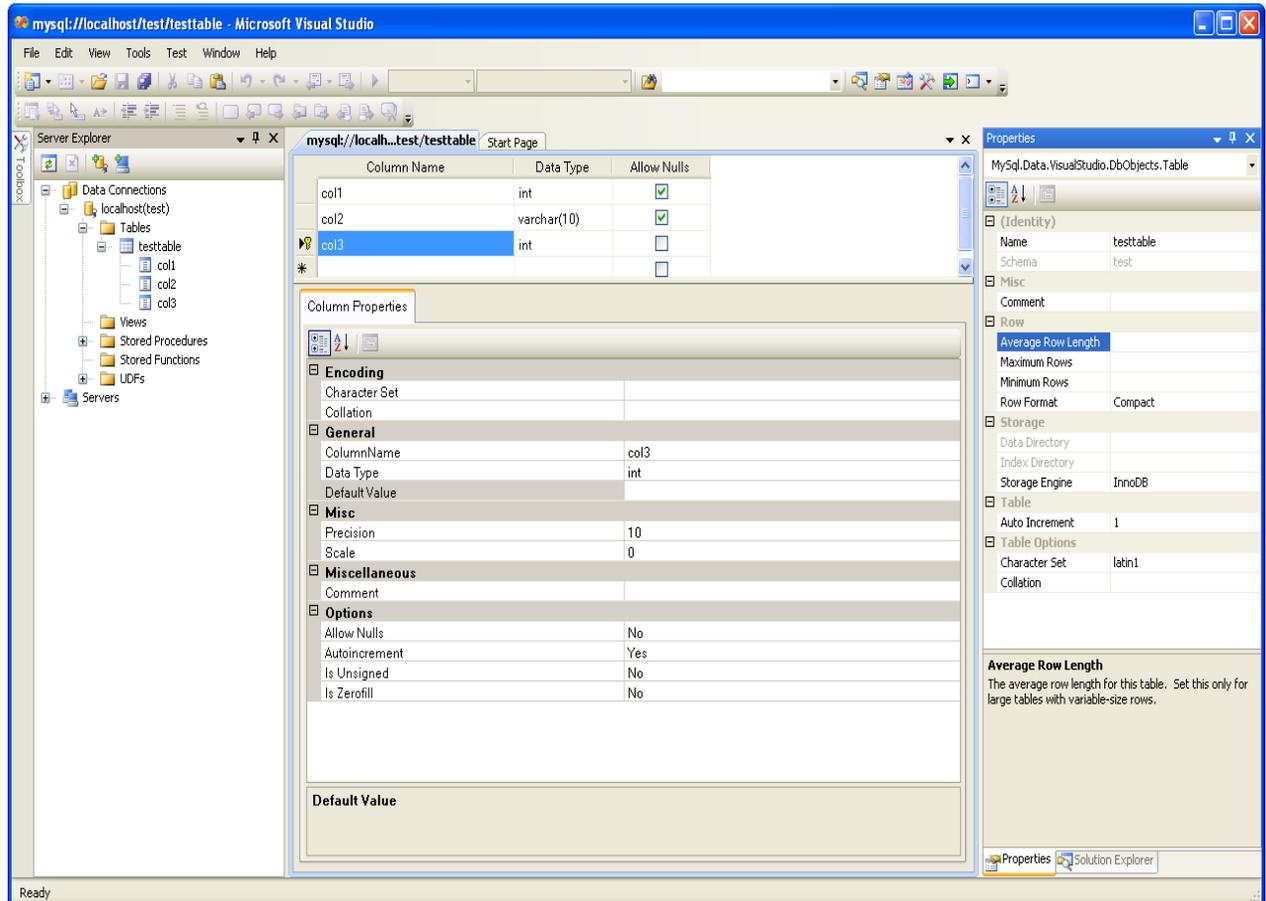
The following table properties can be set:

- Auto Increment.
- Average Row Length.
- Character Set.
- Collation.
- Comment.
- Data Directory.
- Index Directory.
- Maximum Rows.
- Minimum Rows.
- Name.
- Row Format.
- Schema.

- **Storage Engine.** Note that in MySQL 5.5 and higher, the default storage engine for new tables is [InnoDB](#). See [Section 14.3.1, “InnoDB as the Default MySQL Storage Engine”](#) for more information about the choice of storage engine, and considerations if you convert existing tables to [InnoDB](#).

The property [Schema](#) is read-only.

Figure 22.12. Table Properties



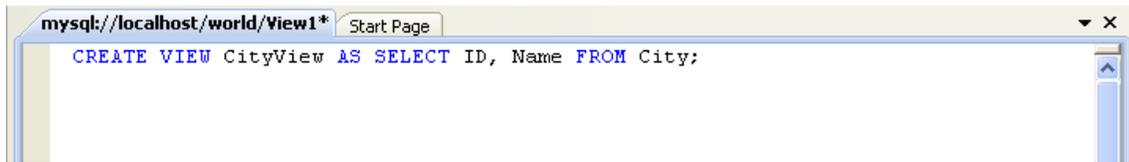
22.2.3.4. Editing Views

To create a new view, right-click the Views node under the connection node in Server Explorer. From the node's context menu, choose the [Create View](#) command. This command opens the SQL Editor.

Figure 22.13. Editing View SQL



You can then enter the SQL for your view.

Figure 22.14. View SQL Added

To modify an existing view, double-click a node of the view to modify, or right-click this node and choose the [Alter View](#) command from a context menu. Either of the commands opens the SQL Editor.

All other view properties can be set in the Properties window. These properties are:

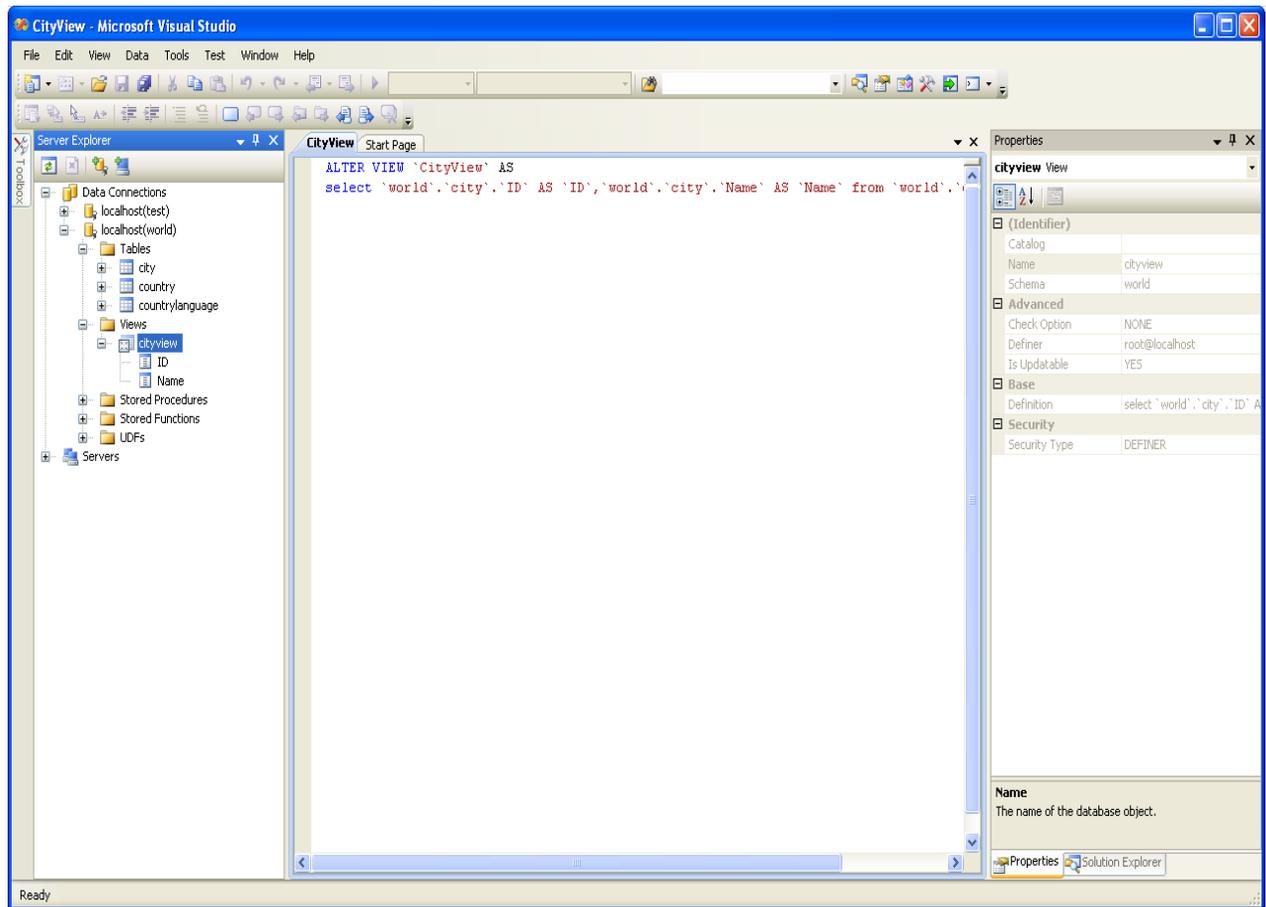
- Catalog
- Check Option
- Definer
- Definition
- Definer
- Is Updatable
- Name
- Schema
- Security Type

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case, set the desired value with an embedded combobox.

The properties [Is Updatable](#) and [Schema](#) are readonly.

To save changes you have made, use either [Save](#) or [Save All](#) buttons of the Visual Studio main toolbar, or press **Control+S**.

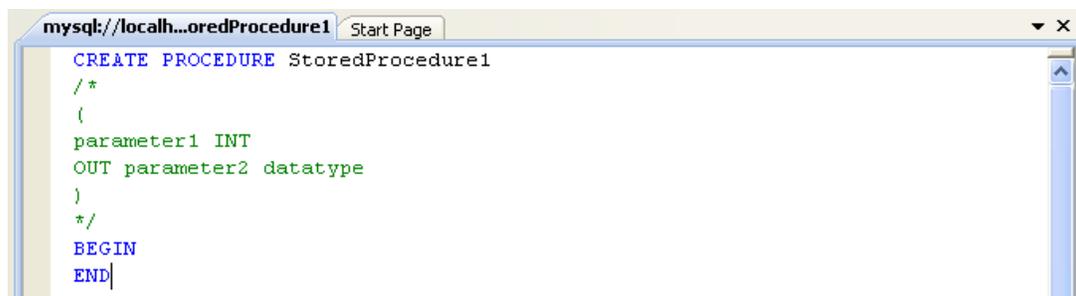
Figure 22.15. View SQL Saved



22.2.3.5. Editing Stored Procedures and Functions

To create a new stored procedure, right-click the **Stored Procedures** node under the connection node in Server Explorer. From the node's context menu, choose the **Create Routine** command. This command opens the SQL Editor.

Figure 22.16. Edit Stored Procedure SQL



To create a new stored function, right-click the **Functions** node under the connection node in Server Explorer. From the node's context menu, choose the **Create Routine** command.

To modify an existing stored routine (procedure or function), double-click the node of the routine to modify, or right-click this node and choose the **Alter Routine** command from the context menu. Either of the commands opens the SQL Editor.

To create or alter the routine definition using SQL Editor, type this definition in the SQL Editor using standard SQL. All other routine properties can be set in the Properties window. These properties are:

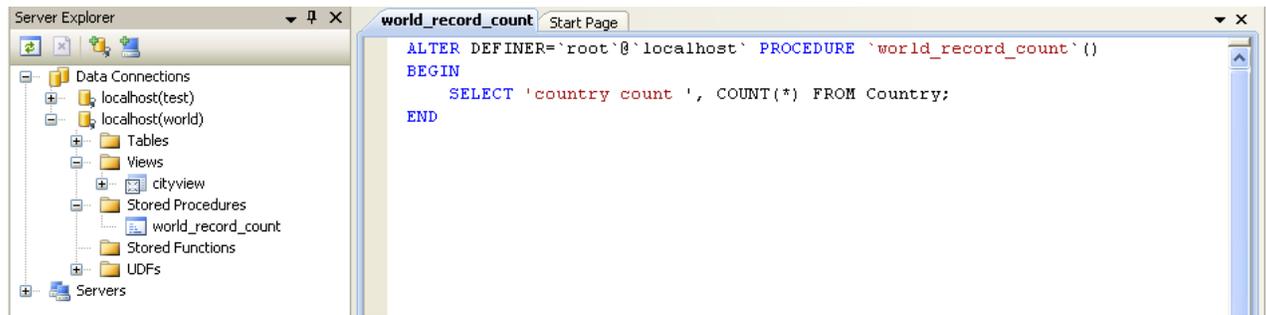
- Body
- Catalog
- Comment
- Creation Time
- Data Access
- Definer
- Definition
- External Name
- External Language
- Is Deterministic
- Last Modified
- Name
- Parameter Style
- Returns
- Schema
- Security Type
- Specific Name
- SQL Mode
- SQL Path
- Type

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case, set the desired value using the embedded combo box.

You can also set all the options directly in the SQL Editor, using the standard `CREATE PROCEDURE` or `CREATE FUNCTION` statement. However, it is recommended to use the Properties window instead.

To save changes you have made, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or press **Control+S**.

Figure 22.17. Stored Procedure SQL Saved



To observe the runtime behavior of a stored routine and debug any problems, use the Stored Procedure Debugger (available in Connector/Net 6.6 and higher). See [Section 22.2.3.8, “Debugging Stored Procedures and Functions”](#) for details.

22.2.3.6. Editing Triggers

To create a new trigger, right-click the node of the table in which to add the trigger. From the node's context menu, choose the **Create Trigger** command. This command opens the SQL Editor.

To modify an existing trigger, double-click the node of the trigger to modify, or right-click this node and choose the **Alter Trigger** command from the context menu. Either of the commands opens the SQL Editor.

To create or alter the trigger definition using SQL Editor, type the trigger statement in the SQL Editor using standard SQL.



Note

Enter only the trigger statement, that is, the part of the `CREATE TRIGGER` query that is placed after the `FOR EACH ROW` clause.

All other trigger properties are set in the Properties window. These properties are:

- Definer
- Event Manipulation
- Name
- Timing

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case, set the desired value using the embedded combo box.

The properties `Event Table`, `Schema`, and `Server` in the Properties window are read-only.

To save changes you have made, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or press **Control+S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

To observe the runtime behavior of a stored routine and debug any problems, use the Stored Procedure Debugger (available in Connector/Net 6.6 and higher). See [Section 22.2.3.8, “Debugging Stored Procedures and Functions”](#) for details.

22.2.3.7. Editing User Defined Functions (UDF)

To create a new User Defined Function (UDF), right-click the **UDFs** node under the connection node in Server Explorer. From the node's context menu, choose the **Create UDF** command. This command opens the UDF Editor.

To modify an existing UDF, double-click the node of the UDF to modify, or right-click this node and choose the **Alter UDF** command from the context menu. Either of these commands opens the UDF Editor.

The UDF editor enables you to set the following properties:

- Name
- So-name (DLL name)
- Return type
- Is Aggregate

There are text fields for both names, a combo box for the return type, and a check box to indicate if the UDF is aggregate. All these options are also accessible using the Properties window.

The property `Server` in the Properties window is read-only.

To save changes you have made, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or press **Control+S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

To observe the runtime behavior of a stored routine and debug any problems, use the Stored Procedure Debugger (available in Connector/Net 6.6 and higher). See [Section 22.2.3.8, "Debugging Stored Procedures and Functions"](#) for details.

22.2.3.8. Debugging Stored Procedures and Functions

The stored procedure debugger, new in Connector/Net 6.6, provides facilities for setting breakpoints, stepping into individual statements (Step Into, Step Out, Step Over), evaluating and changing local variable values, evaluating breakpoints, and other typical debugging tasks.

Installing the Debugger

To enable the stored procedure debugger, install Connector/Net 6.6 or higher and choose the **Complete** option.

Privileges

The debugger recreates at the start of each debug session a `serversidedebugger` database in your server. This database helps to track the instrumented code and implement observability logic in the debugged routine. Your current connection needs to have privileges to create that database, and its associated stored routines, functions, and tables.

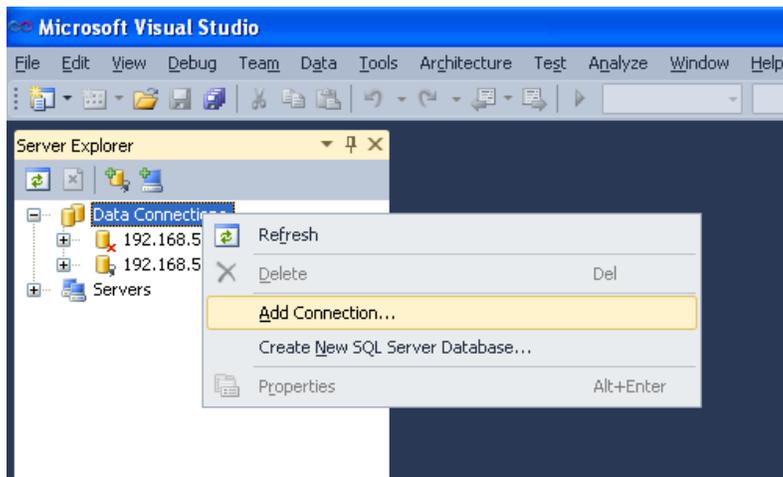
The debugger makes changes behind the scenes to temporarily add instrumentation code to the stored routines that you debug. You must have the `ALTER ROUTINE` privilege for each stored procedure, function, or trigger that you debug. (Including procedures and functions that are called, and triggers that are fired, by a procedure that you are debugging.)

Starting the Debugger

To start the debugger, follow these steps:

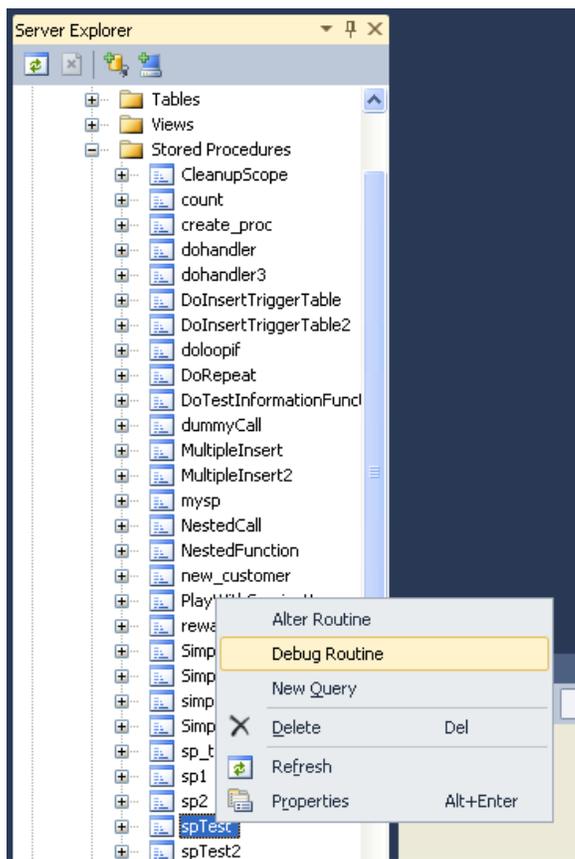
- Choose a connection in the Visual Studio Server Explorer.

Figure 22.18. Connection Dialog



- Expand the **Stored Procedures** folder. Only stored procedures can be debugged directly. To debug a user-defined function, create a stored procedure that calls the function.
- Click on a stored procedure node, then right-click and from the context menu choose **Debug Routine**.

Figure 22.19. Choose a Stored Routine to Debug



At this point, Visual Studio switches to debug mode, opening the source code of the routine being debugged in step mode, positioned on the first statement.

If the initial routine you debug has one or more arguments, a popup will show up with a grid (a row per each argument and three columns: one for the argument, one for the argument value (this is editable) and one for nullifying that argument value (a checkbox)). After setting up all the argument values, you can press **OK** to start the debug session, or **Cancel** to cancel the debug session.

Figure 22.20. Setting Arguments (1 of 2)

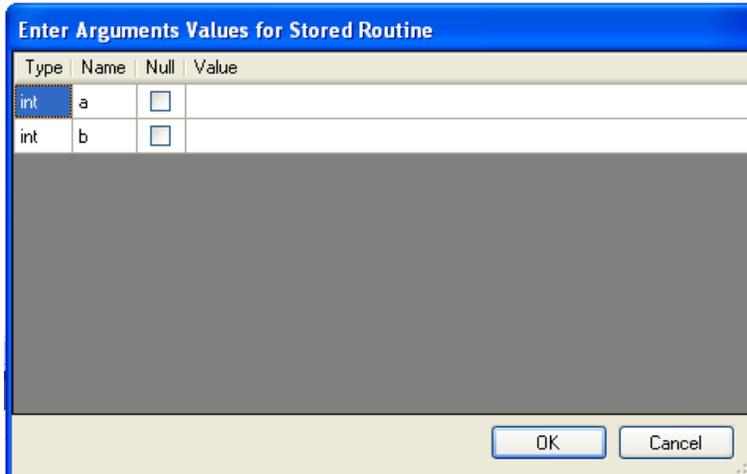
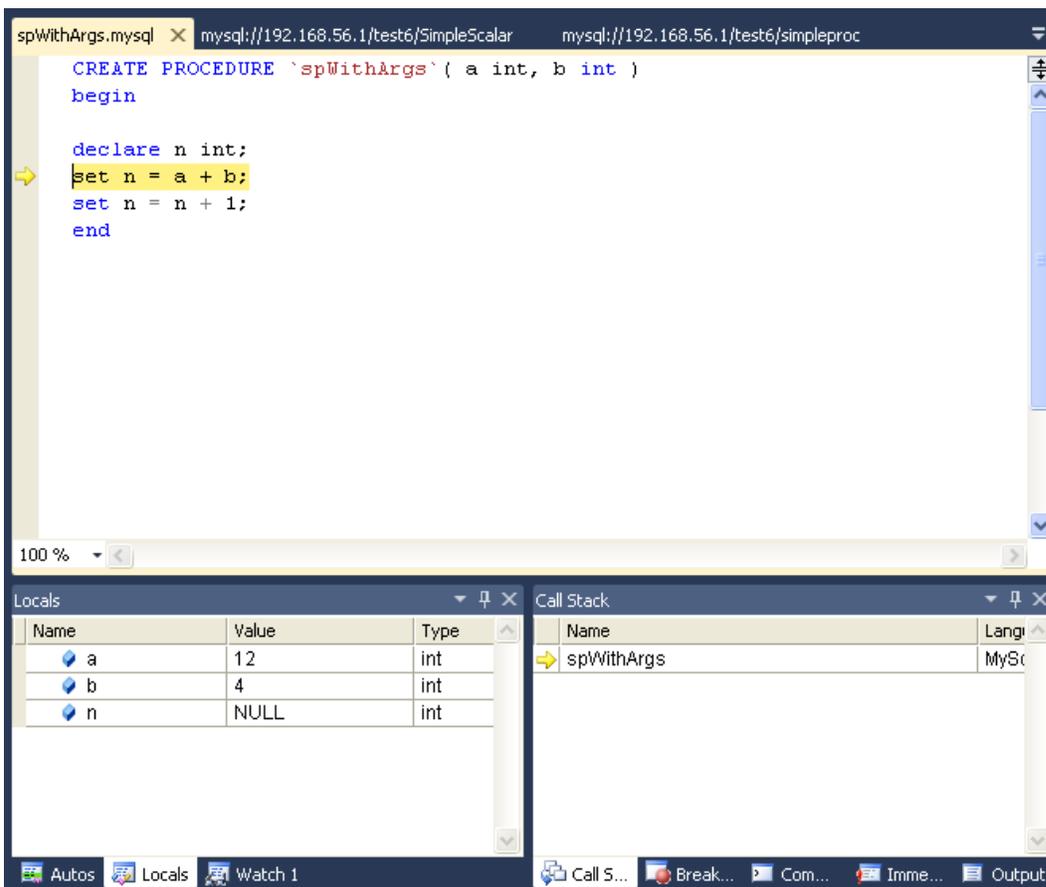


Figure 22.21. Setting Arguments (2 of 2)



How the Debugger Works

To have visibility into the internal workings of a stored routine, the debugger prepares a special version of the procedure, function, or trigger being debugged, instrumented with extra code to keep track of the current line being stepped into and the values of all the local variables. Any other stored procedures, functions, or triggers called from the routine being debugged are instrumented the same way. The debug versions of the routines are prepared for you automatically, and when the debug session ends (by either pressing **F5** or **Shift+F5**), the original versions of the routines are automatically restored.

A copy of the original version of each instrumented routine (the version without instrumentation) is stored in the `AppData\Roaming\MySQLDebuggerCache` folder for the current Windows user (the path returned by calling `System.Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)` in .NET, plus appending `MySQLDebuggerCache`. There is one file for each instrumented routine, named `routine_name.mysql`. For example, in Windows 7, for a user named `fergs`, the path is `C:\Users\fergs\AppData\Roaming\MySQLDebuggerCache`.

Two threads are used, one for the debugger and one for the routine being debugged. The threads run in strict alternation, switching between the debugger and the routine as each statement is executed in the stored routine.

Basic Debugging Operations

The debugger has the same look and feel as the standard Visual Studio debuggers for C#, VB.NET or C++. In particular, the following are true:

Locals and Watches

- To show the Locals tab, choose the menu item **Debug -> Windows -> Locals**.

The **Locals** tab lists all the variables available in the current scope: variables defined with `DECLARE` at any point in the routine, argument parameters, and session variables that are referenced.

- If the last step operation changes the value of a local, its value will be highlighted in red (until another statement is executed or stepped).
- You can change the value of any local.
- To show the **Watch** tab, choose the menu item **Debug -> Windows -> Watch**.

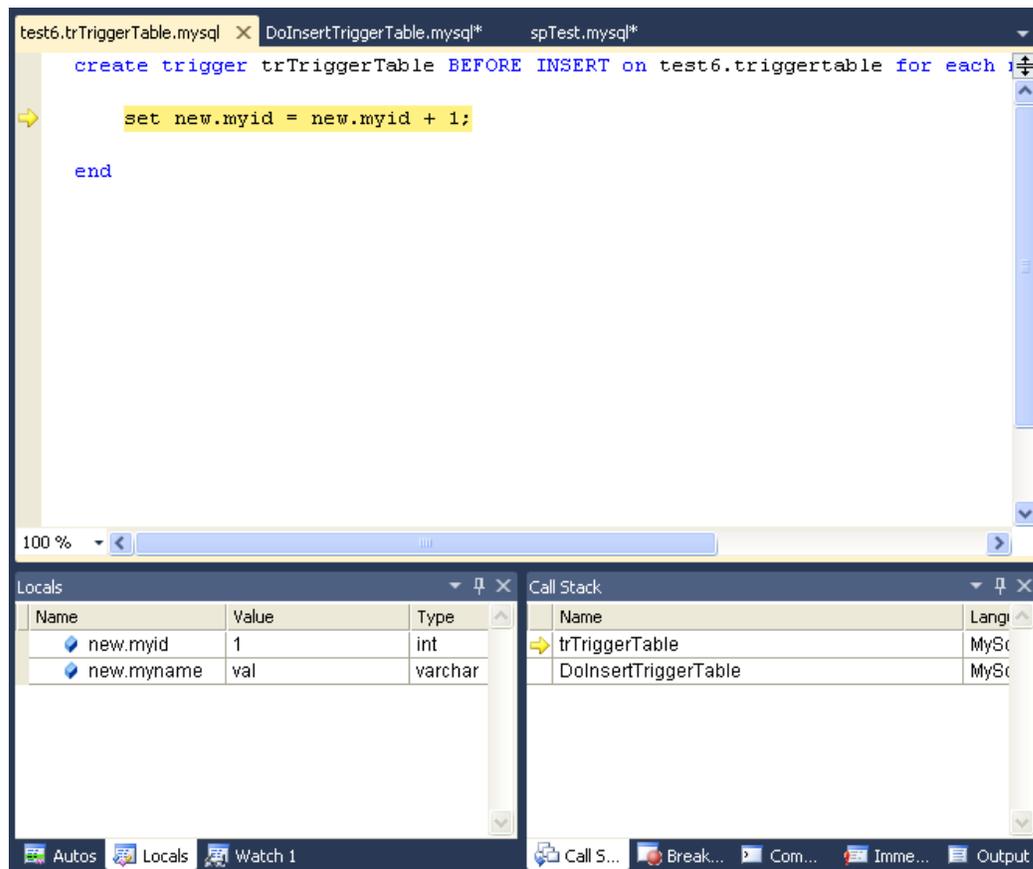
To define a watch, type any valid MySQL expression, optionally including function calls. If the watch evaluation makes sense in the current context (current stack frame), it will show its value, otherwise it will show an error message in the same row the watch was defined.

- When debugging a trigger, in addition to any locals declared or session variables referenced, the new and old object (when applicable) will be listed. For example in a trigger for `INSERT`, for a table defined like:

```
create table t1( id int, myname varchar( 50 ));
```

the locals will list the extra variables `new.id` and `new.myname`. For an `UPDATE` trigger, you will also get the extra variables `old.id` and `old.myname`. These variables from the new and old objects can be manipulated the same way as any ordinary local variable.

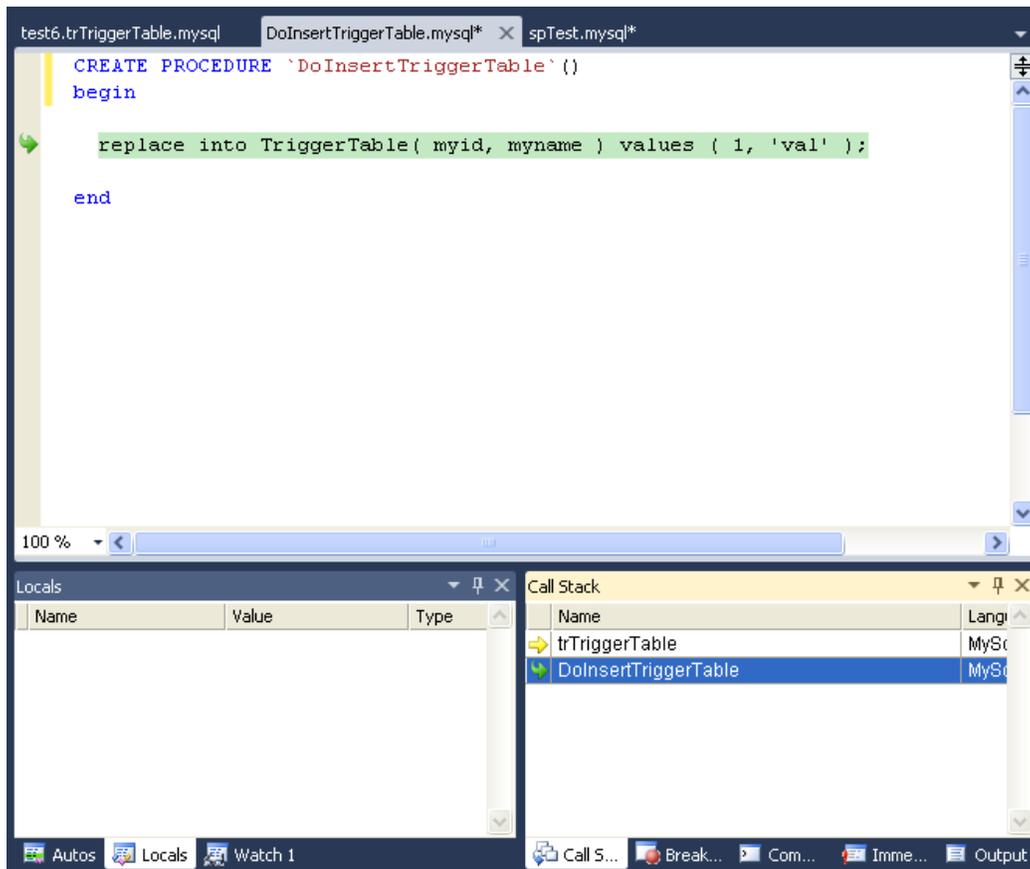
Figure 22.22. Debugging a Trigger



Call Stack

- To show the **Call Stack** tab, choose the menu item **Debug -> Windows -> Call Stack**.
- The stack trace (in the **Call Stack** tab) will list all the stack traces, one for each routine invocation. The one with a yellow mark is the current stepping point. Clicking in another will activate in the editor the tab for that routine source, highlighting in green the last statement stepped.

Figure 22.23. Call Stack



Stepping

- Stepping of a new routine starts in the first executable instruction (excluding declares, handlers, cursor declarations, and so on).

Figure 22.24. Debug Stepping

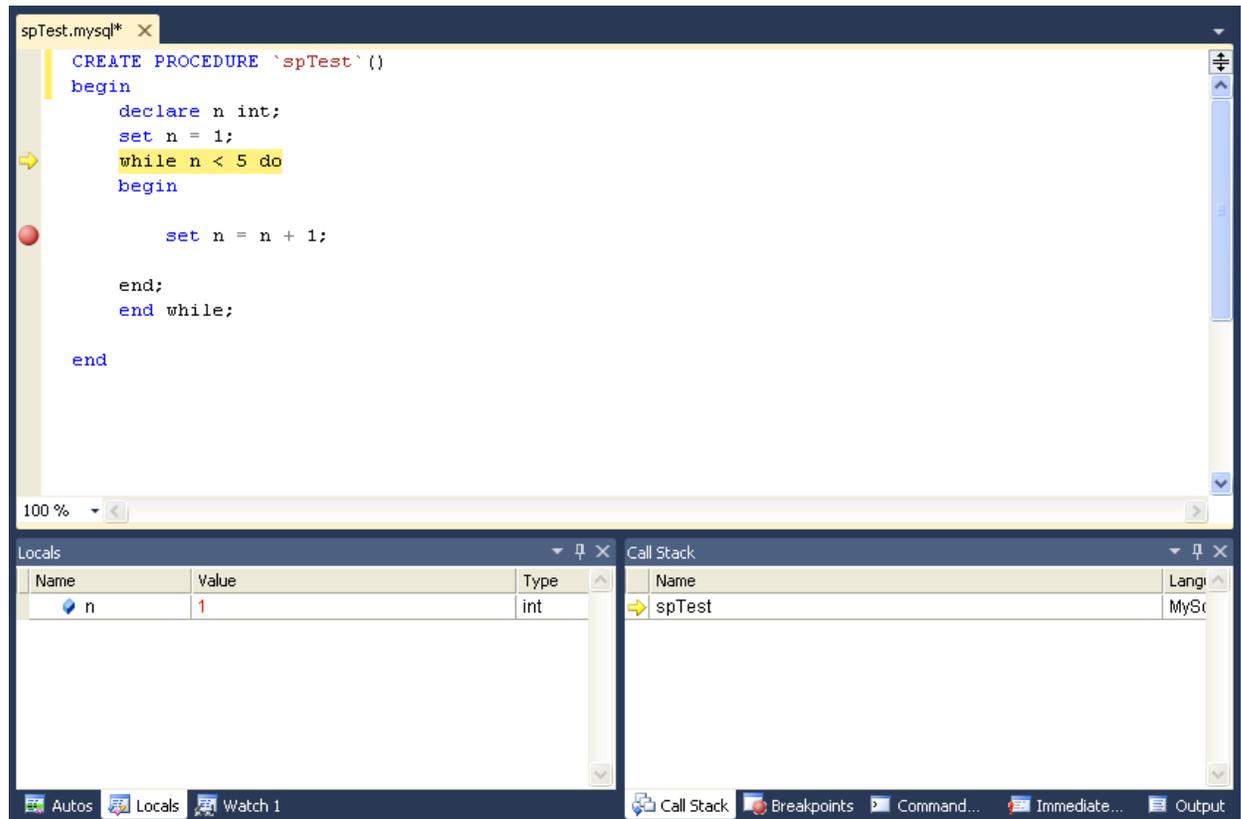


Figure 22.25. Function Stepping (1 of 2)

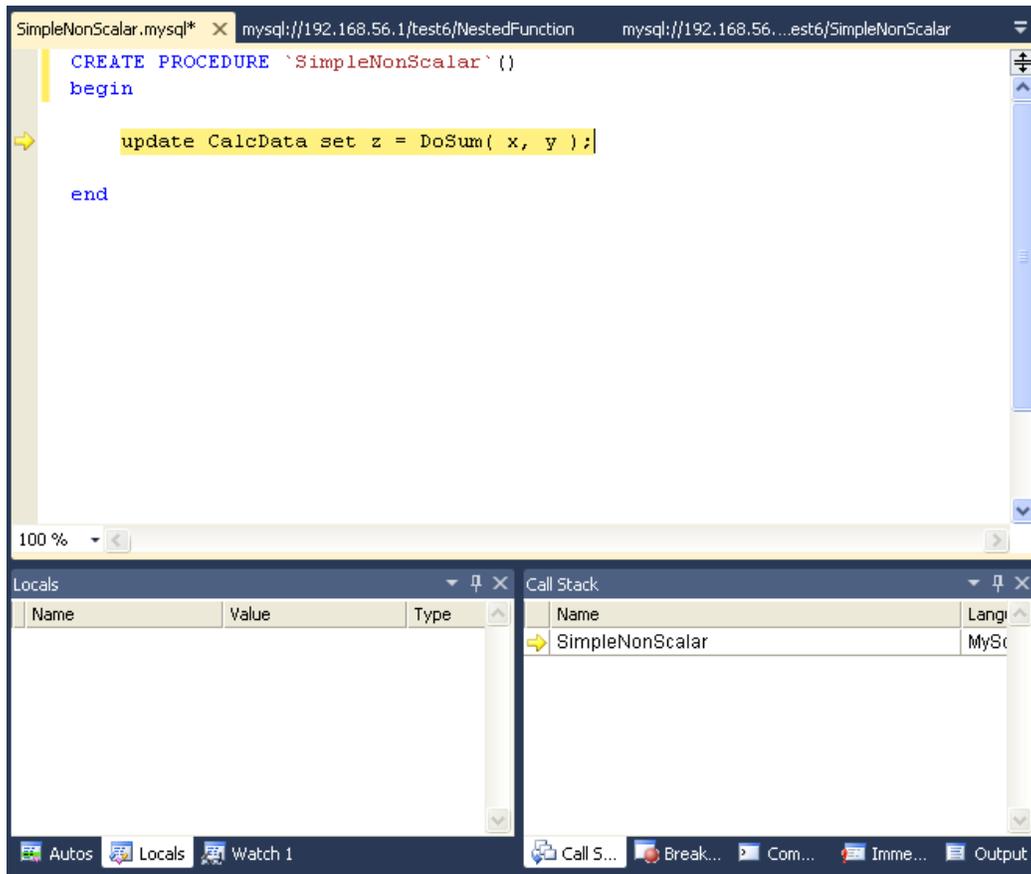
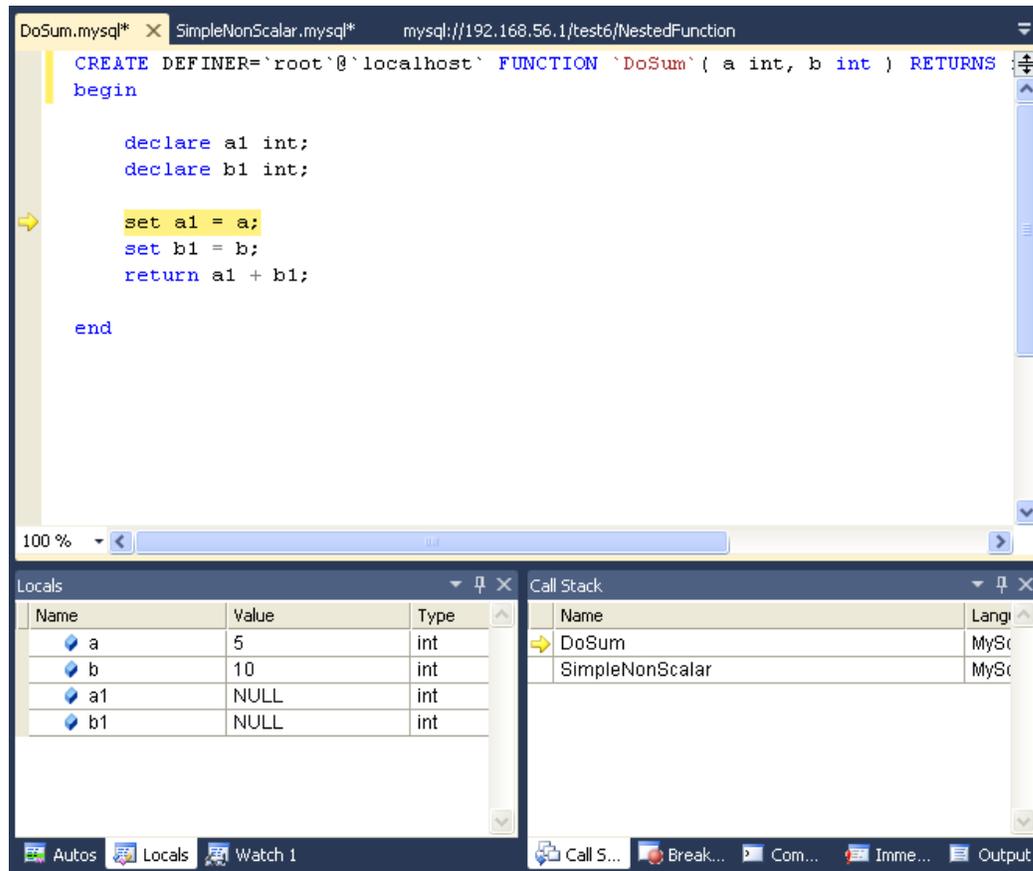


Figure 22.26. Function Stepping (2 of 2)



- To step into the code of a condition handler, the condition must be triggered in the rest of the MySQL routine.
- The next statement to be executed is highlighted in yellow.
- To continue stepping, you can choose between **Step Into** (by pressing **F11**), **Step Out** (by pressing **F10**) or **Step Over** (by pressing **Shift+F11**).
- You can step out of any of functions, triggers or stored procedures. If you step from the main routine, it will run that routine to completion and finish the debug session.
- You can step over stored procedure calls, stored functions, and triggers. (To step over a trigger, step over the statement that would cause the trigger to fire.)
- When stepping into a single statement, the debugger will step into each individual function invoked by that statement and each trigger fired by that statement. The order in which they are debugged is the same order in which the MySQL server executes them.
- You can step into triggers triggered from **INSERT**, **DELETE**, **UPDATE**, and **REPLACE** statements.
- Also, the number of times you enter into a stored function or trigger depends on how many rows are evaluated by the function or affected by the trigger. For example, if you press **F11 (Step Into)** into an **UPDATE** statement that modifies three rows (calling a function for a column in the **SET** clause, thus invoking the function for each of the three rows), you will step into into that function three times in succession, once for each of the rows. You can accelerate this debug session by disabling any

breakpoints defined in the given stored function and pressing **Shift+F11** to step out. In this example, the order in which the different instances of the stored function are debugged is server-specific: the same order used by the current MySQL server instance to evaluate the three function invocations.

Breakpoints

- To show the **Breakpoints** tab, choose the menu item **Debug -> Windows -> Breakpoints**.
- The **Breakpoints** tab will show all the breakpoints defined. From here, you can enable and disable breakpoints one by one or all at once (using the toolbar on top of the **Breakpoints** tab).
- You can define new breakpoints only in the middle of a debug session. Click in the left gray border of any MySQL editor, or click anywhere in a MySQL editor and press **F9**. In the familiar Visual Studio way, you press **F9** once to create a breakpoint in that line, and press it again to remove that breakpoint.
- Once a breakpoint is defined, it will appear enabled (as filled red circle left to the current row if that line is a valid statement to put a breakpoint) or disabled (as a non-filled red circle left to the current row if that row is not valid to put a breakpoint).
- To define conditional breakpoints, after creating the breakpoint, right click in the red dot and choose **Condition...**. There you can put any valid MySQL expression and state if the condition is **Is True** or **Has changed**. The former will trigger the breakpoint every time the condition is true, the latter every time the condition value has changed. (If you define a conditional breakpoint, it is not enough to step into the line with the breakpoint defined to trigger such a breakpoint.)

Figure 22.27. Conditional Breakpoints

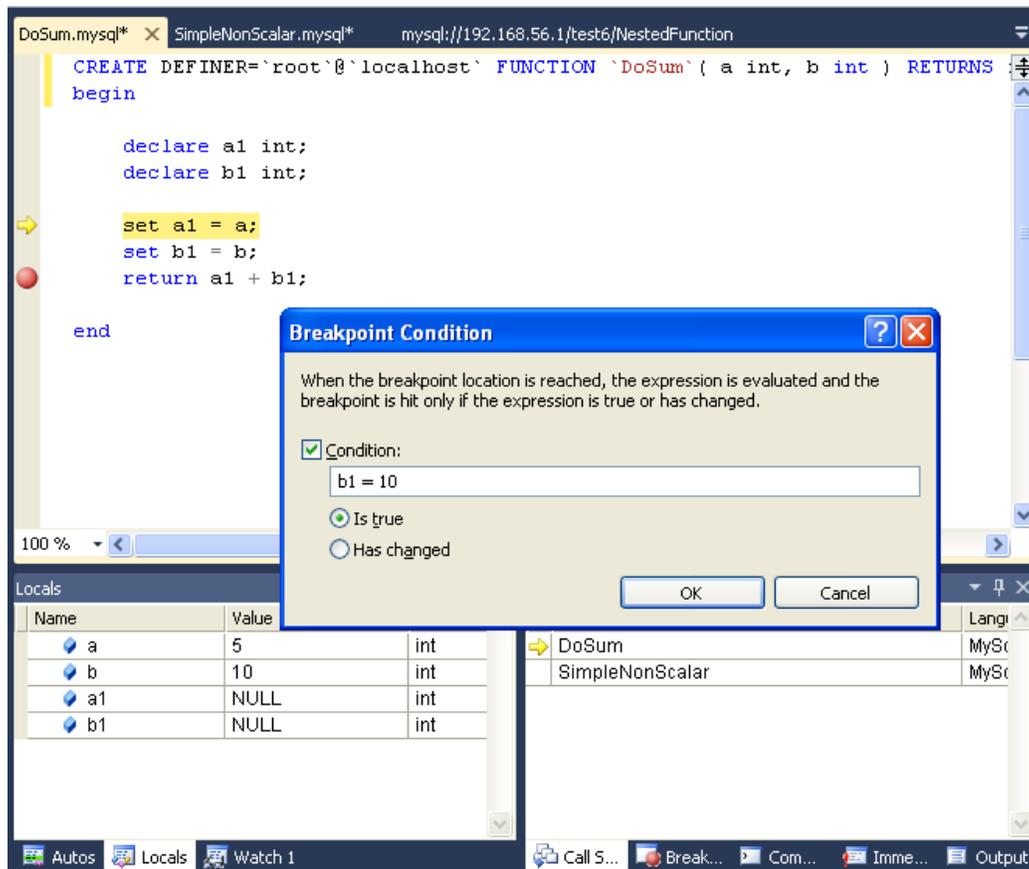
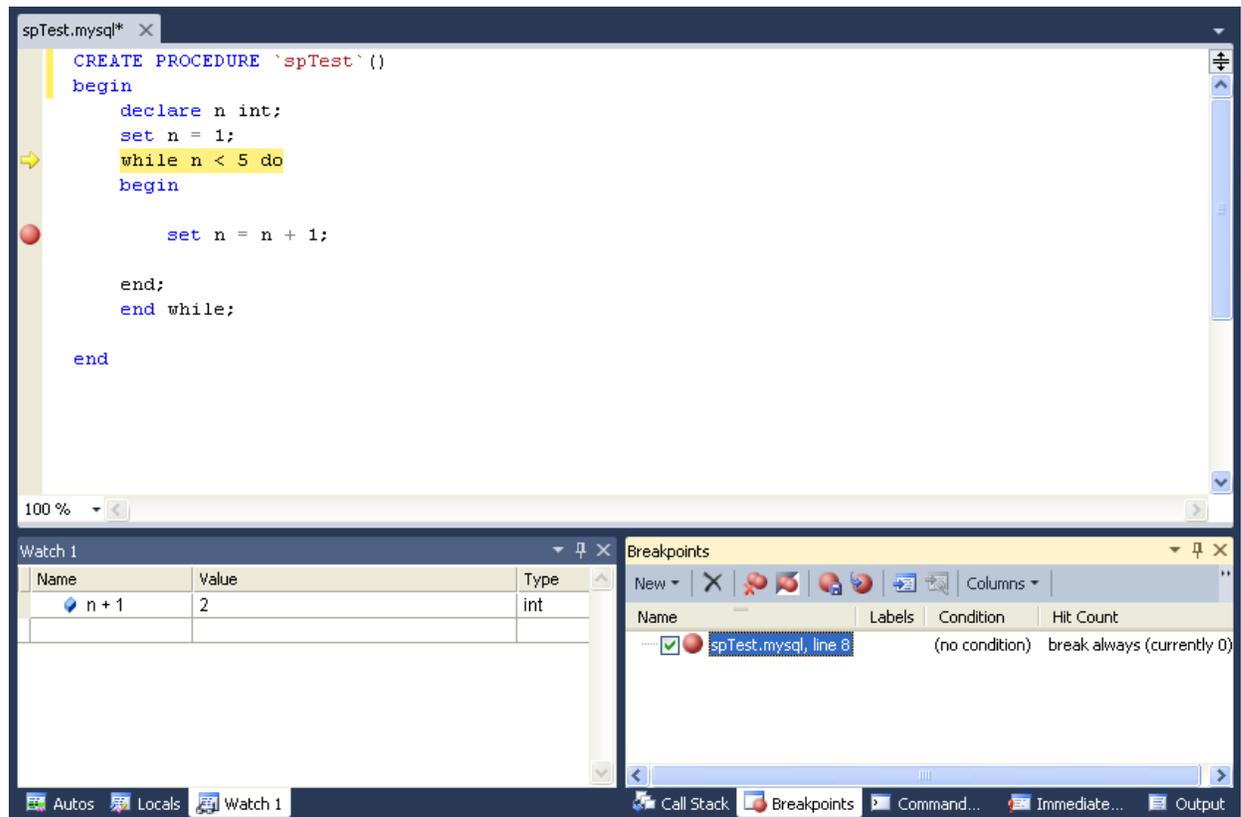


Figure 22.28. Expressions and Breakpoints



- To define pass count breakpoints, after creating the breakpoint, right click in the red dot and choose **Hit Count...**. In the popup dialog, define the specific condition to set. For example, **break when the hit count is equal to** and a value 3 will trigger the breakpoint the third time it is hit.

Other Features

- To abort the debug session (and the execution of the current call stack of routines), press Shift+F5.
- To run the routine to completion (or until next breakpoint hit), press F5.
- For all functionality you can use (in addition to the shortcuts documented), see the options in the **Debug** menu of Visual Studio.

Limitations

- Code being debugged must not use `get_lock` or `release_lock` MySQL functions, since they are used internally by the debugger infrastructure to synchronize the debugger and the debugged routine.
- Code being debugged must avoid using any transaction code (`START TRANSACTION`, `COMMIT`, `ROLLBACK`) due to the possibility of wiping out the contents of the debugger tables. (This limitation may be removed in the future).
- You cannot debug the routines in the `serversidedebugger` database.
- The MySQL server running the routine being debugged can be any version between 5.0 and 5.6, running on Windows, Linux, or any other supported platform.

- We recommend always running debug sessions on test and development servers, rather than against a MySQL production server, because debugging can cause temporary performance issues or even deadlocks. The instrumented versions of the routines being debugged use locks, that the rest of the production code may not be aware of.

Keyboard Shortcuts

The following list summarizes the keyboard shortcuts for debugging:

- **F9** Toggles breakpoints
- **F11**: Step into once
- **F10**: Step over once
- **Shift+F11**: Step out once
- **F5**: Run
- **Shift+F5**: Abort current debug session

22.2.3.9. Cloning Database Objects

Tables, views, stored procedures, and functions can be cloned using the appropriate Clone command from the context menu: [Clone Table](#), [Clone View](#), [Clone Routine](#). The clone commands open the corresponding editor for a new object: the **Table Editor** for cloning a table, and the **SQL Editor** for cloning a view or a routine.

The editor is filled with values of the original object. You can modify these values in the usual manner.

To save the cloned object, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or press **Control+S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

22.2.3.10. Dropping Database Objects

Tables, views, stored routines, triggers, and UDFs can be dropped with the appropriate Drop command selected from its context menu: [Drop Table](#), [Drop View](#), [Drop Routine](#), [Drop Trigger](#), [Drop UDF](#).

You will be asked to confirm the execution of the corresponding drop query in a confirmation dialog.

You can only drop a single object at a time.

22.2.3.11. Using the ADO.NET Entity Framework

Connector/Net 6.0 introduced support for the ADO.NET Entity Framework. ADO.NET Entity Framework was included with .NET Framework 3.5 Service Pack 1, and Visual Studio 2008 Service Pack 1. ADO.NET Entity Framework was released on 11th August 2008.

ADO.NET Entity Framework provides an Object Relational Mapping (ORM) service, mapping the relational database schema to objects. The ADO.NET Entity Framework defines several layers, these can be summarized as:

- **Logical** - this layer defines the relational data and is defined by the Store Schema Definition Language (SSDL).

- **Conceptual** - this layer defines the .NET classes and is defined by the Conceptual Schema Definition Language (CSDL)
- **Mapping** - this layer defines the mapping from .NET classes to relational tables and associations, and is defined by Mapping Specification Language (MSL).

Connector/Net integrates with Visual Studio 2008 to provide a range of helpful tools to assist the developer.

A full treatment of ADO.NET Entity Framework is beyond the scope of this manual. If you are unfamiliar with ADO.NET, review the [Microsoft ADO.NET Entity Framework documentation](#).

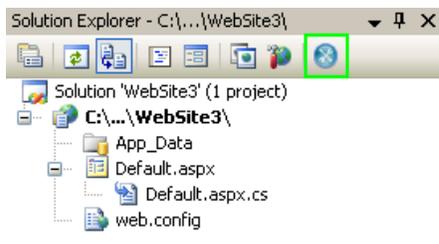
Tutorials on getting started with ADO.NET Entity Framework are available. See [Section 22.2.4.5, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source"](#) and [Section 22.2.4.6, "Tutorial: Databinding in ASP.NET using LINQ on Entities"](#).

22.2.3.12. MySQL Website Configuration Tool

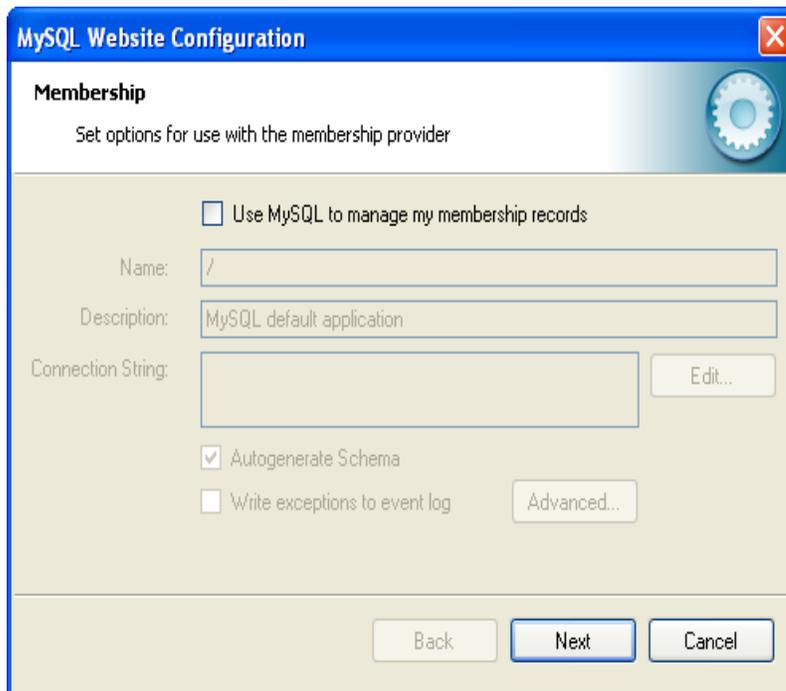
MySQL Connector/Net 6.1 introduced the MySQL Website Configuration Tool. This is a facility available in Visual Studio that enables you to configure the Membership, Role, Session State and Profile Provider, without editing configuration files. You set your configuration options within the tool, and the tool modifies your `web.config` file accordingly.

The MySQL Website Configuration Tool appears as a small icon on the Solution Explorer toolbar in Visual Studio, as show by the following screenshot:

Figure 22.29. MySQL Website Configuration Tool



Clicking the Website Configuration Tool icon launches the wizard and displays the first screen:

Figure 22.30. MySQL Website Configuration Tool - Membership

The screenshot shows a dialog box titled "MySQL Website Configuration" with a "Membership" tab. The dialog has a blue header bar with a close button (X) in the top right corner. Below the header, the text "Membership" is displayed in bold, followed by the instruction "Set options for use with the membership provider". A gear icon is located in the top right corner of the main content area. The main content area contains the following elements:

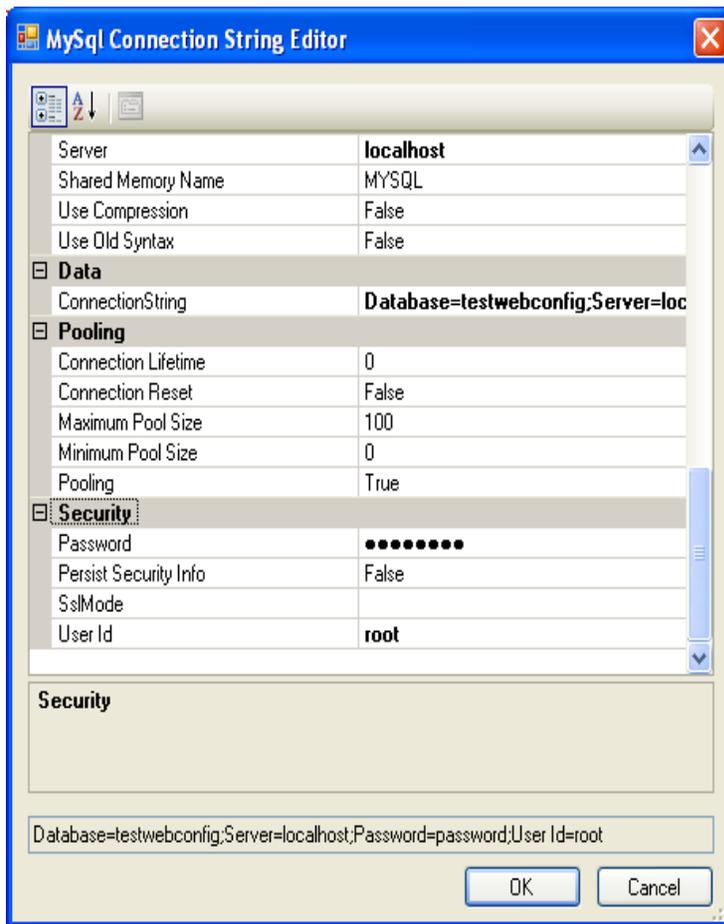
- A checkbox labeled "Use MySQL to manage my membership records" which is currently unchecked.
- A "Name:" label followed by a text box containing the character "/".
- A "Description:" label followed by a text box containing the text "MySQL default application".
- A "Connection String:" label followed by a large text box and an "Edit..." button to its right.
- A checked checkbox labeled "Autogenerate Schema".
- An unchecked checkbox labeled "Write exceptions to event log" followed by an "Advanced..." button.

At the bottom of the dialog, there are three buttons: "Back", "Next" (which is highlighted with a blue border), and "Cancel".

This allows you to enable use of the MySQL Membership Provider. Click the check box to enable this. You can now enter the name of the application that you are creating the configuration for. You can also enter a description for the application.

You can then click the **Edit...** button to launch the Connection String Editor:

Figure 22.31. MySQL Website Configuration Tool - Connection String Editor



Note that if you have already defined a connection string for the providers manually in `web.config`, or previously using the tool, this will be automatically loaded and displayed, and can then be modified in this dialog.

You can also ensure that the necessary schemas are created automatically for you by selecting the Autogenerate Schema check box. These schemas are used to store membership information. The database used to storage is the one specified in the connection string.

You can also ensure that exceptions generated by the application will be written to the Windows event log by selecting the **Write exceptions to event log** check box.

Clicking the Advanced... button launches a dialog that enables you to set Membership Options. These options dictate such variables as password length required when a user signs up, whether the password is encrypted and whether the user can reset their password or not.

Figure 22.32. MySQL Website Configuration Tool - Advanced Options

Membership Options

Password Options

Minimum Required Password Length: 7

Min Required Non-alphanumeric Characters: 1

Password strength regular expression:

Password Format: Clear

User Options

Require Question/Answer Enable Password Reset

Requires Unique Email Enable Password Retrieval

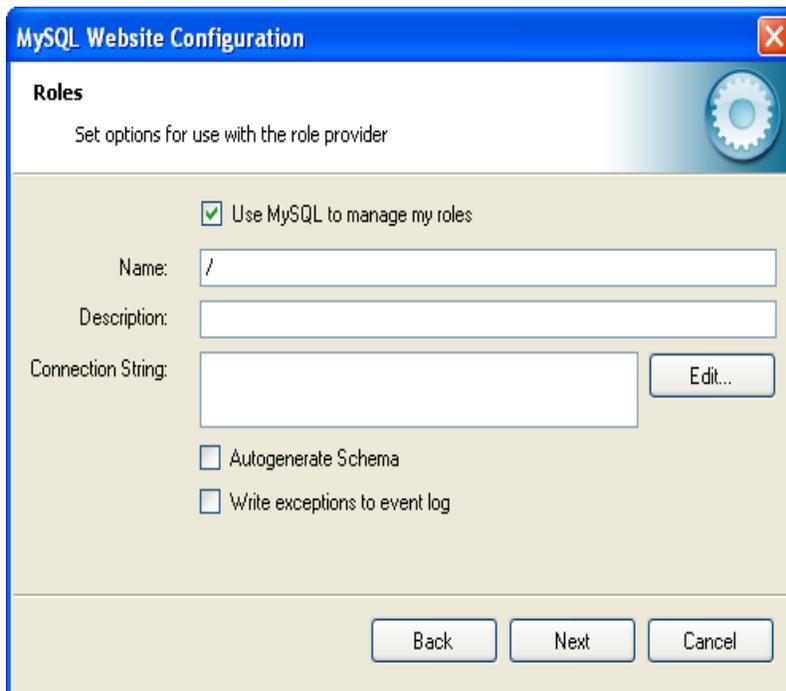
Access Options

Maximum Invalid Password Attempts: 5

Password Attempt Window: 10

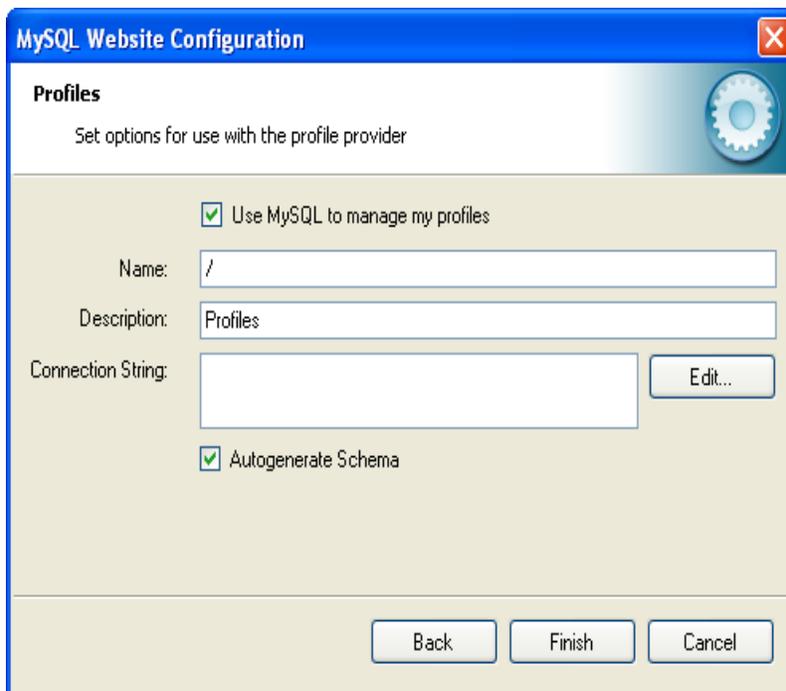
OK Cancel

Once information has been set up as required for configuration of the Membership Provider, the **Next** button can be clicked to display the Roles Provider screen:

Figure 22.33. MySQL Website Configuration Tool - Roles

The screenshot shows the 'Roles' configuration window. The title bar reads 'MySQL Website Configuration'. Below the title bar, the word 'Roles' is displayed in bold, followed by the instruction 'Set options for use with the role provider'. A gear icon is visible in the top right corner. The main area contains the following elements: a checked checkbox for 'Use MySQL to manage my roles'; a 'Name:' label with a text box containing '/'; a 'Description:' label with an empty text box; a 'Connection String:' label with a text box and an 'Edit...' button to its right; two unchecked checkboxes for 'Autogenerate Schema' and 'Write exceptions to event log'; and three buttons at the bottom: 'Back', 'Next', and 'Cancel'.

Again the connection string can be edited, a description added and Autogenerate Schema can be enabled before clicking **Next** to go to the Profiles Provider screen:

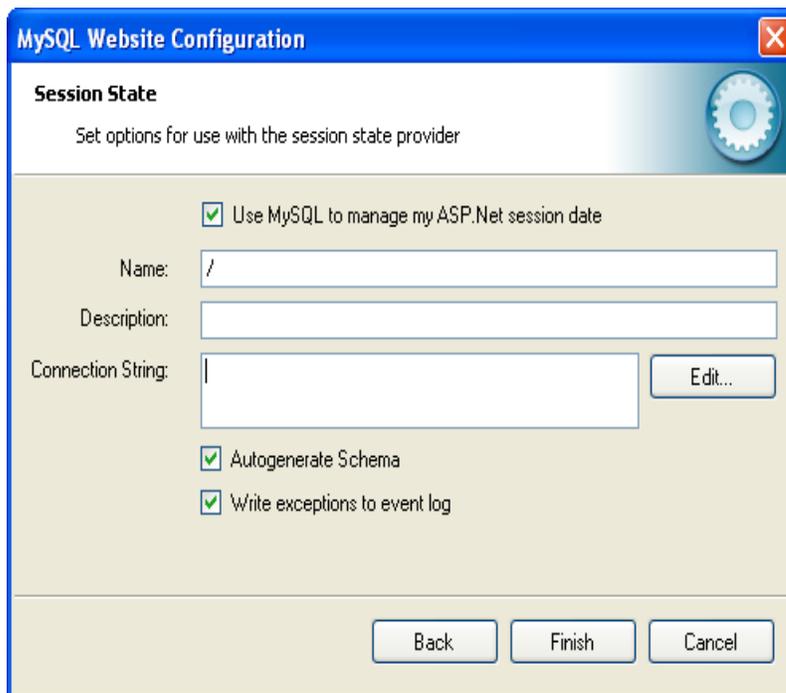
Figure 22.34. MySQL Website Configuration Tool - Profiles

The screenshot shows the 'Profiles' configuration window. The title bar reads 'MySQL Website Configuration'. Below the title bar, the word 'Profiles' is displayed in bold, followed by the instruction 'Set options for use with the profile provider'. A gear icon is visible in the top right corner. The main area contains the following elements: a checked checkbox for 'Use MySQL to manage my profiles'; a 'Name:' label with a text box containing '/'; a 'Description:' label with a text box containing 'Profiles'; a 'Connection String:' label with a text box and an 'Edit...' button to its right; a checked checkbox for 'Autogenerate Schema'; and three buttons at the bottom: 'Back', 'Finish', and 'Cancel'.

This screen display similar options to the previous screens.

Click **Next** to proceed to the Session State configuration page:

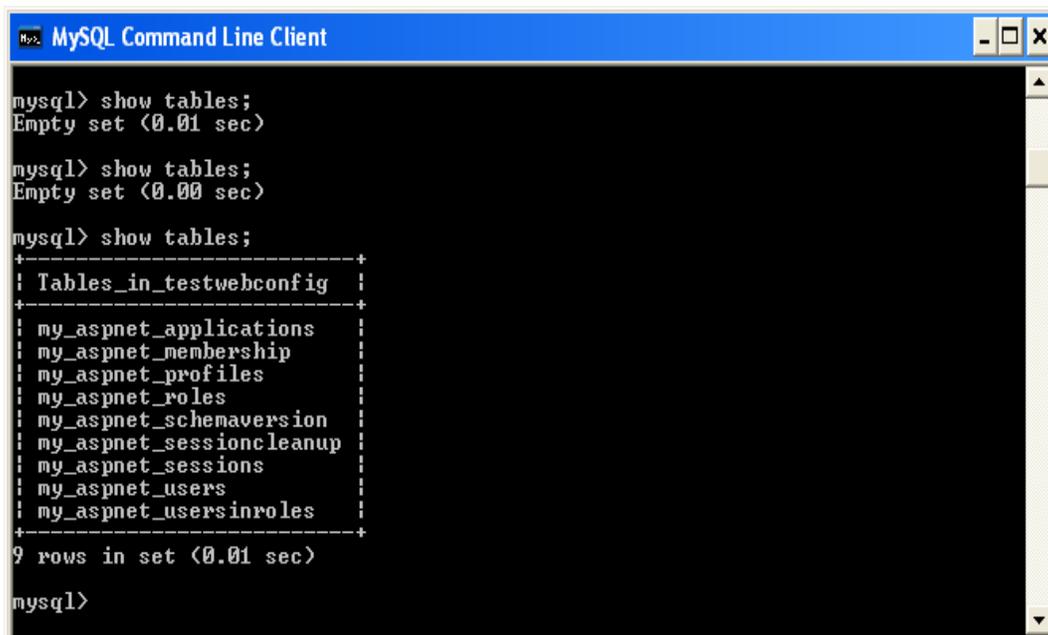
Figure 22.35. MySQL Website Configuration Tool - Session State



Once you have set up the Session State Provider as required, click **Finish** to exit the wizard.

At this point, select the **Authentication Type** to **From Internet**. Launch the ASP.NET Configuration Tool and select the Security tab. Click the **Select authentication type** link and ensure that the **From the internet** radio button is selected. You can now examine the database you created to store membership information. All the necessary tables will have been created for you:

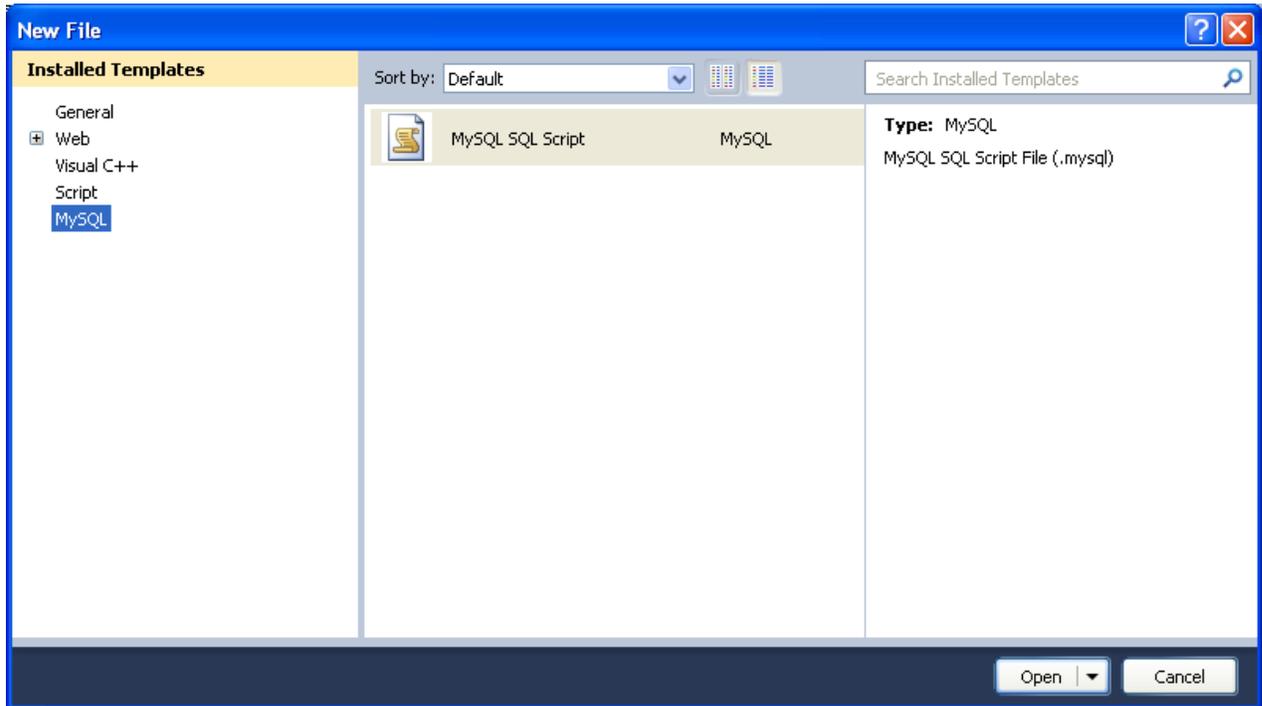
Figure 22.36. MySQL Website Configuration Tool - Tables



22.2.3.13. MySQL SQL Editor

MySQL Connector/Net 6.3.2 introduced a new MySQL SQL Editor. The easiest way to invoke the editor is by selecting the **New**, **File** menu item from the Visual Studio main menu. This displays the **New File** dialog:

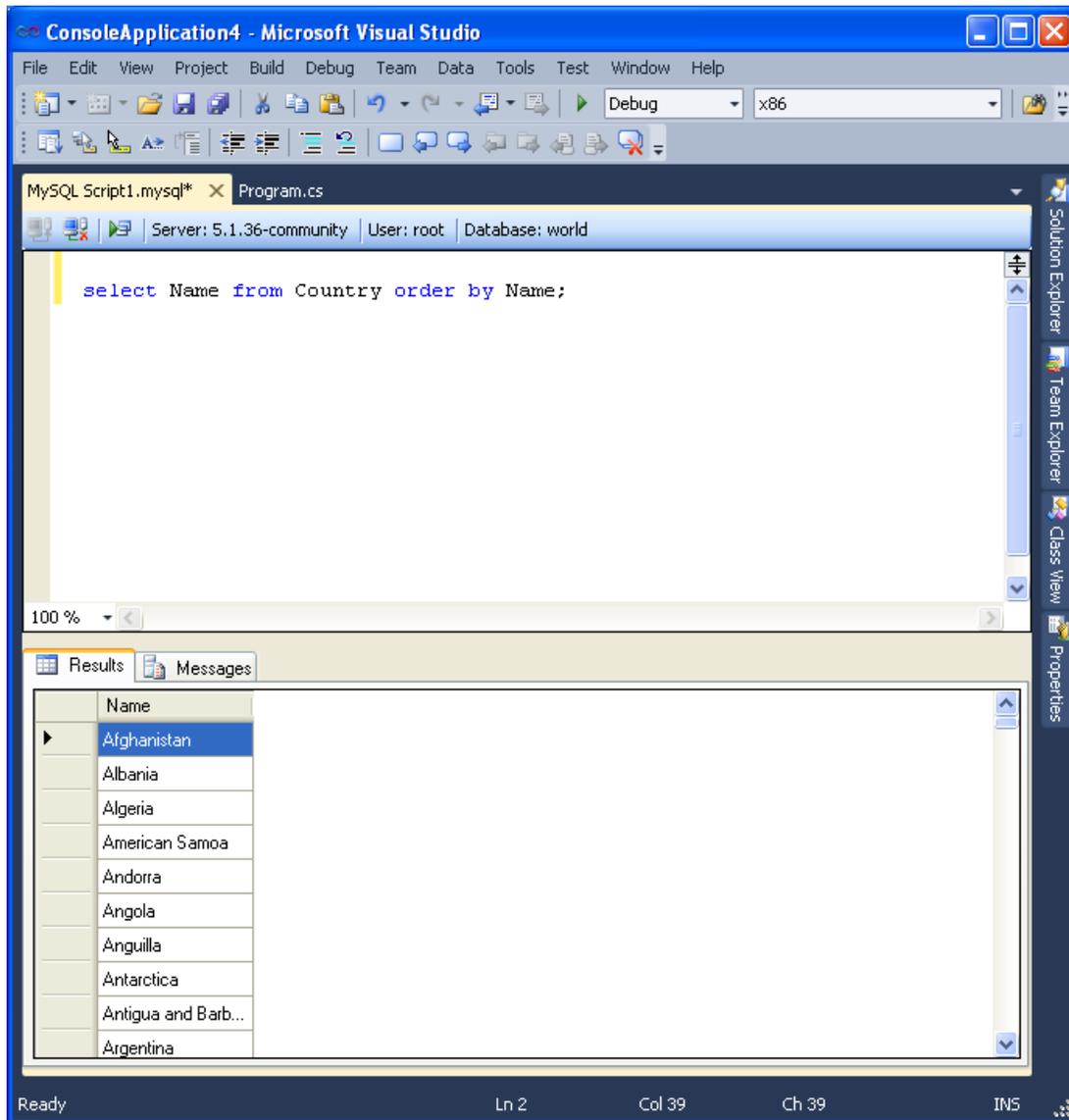
Figure 22.37. MySQL SQL Editor - New File



From the **New File** dialog, select the MySQL template, and then double-click the **MySQL SQL Script** document, or click the **Open** button.

The MySQL SQL Editor will be displayed. You can now enter SQL code as required, or connect to a MySQL server. Click the **Connect to MySQL** button in the MySQL SQL Editor toolbar. You can enter the connection details into the **Connect to MySQL** dialog that is displayed. You can enter the server name, user ID, password and database to connect to, or click the **Advanced** button to select other connection string options. Click the **Connect** button to connect to the MySQL server. To execute your SQL code against the server, click the **Run SQL** button on the toolbar.

Figure 22.38. MySQL SQL Editor - Query

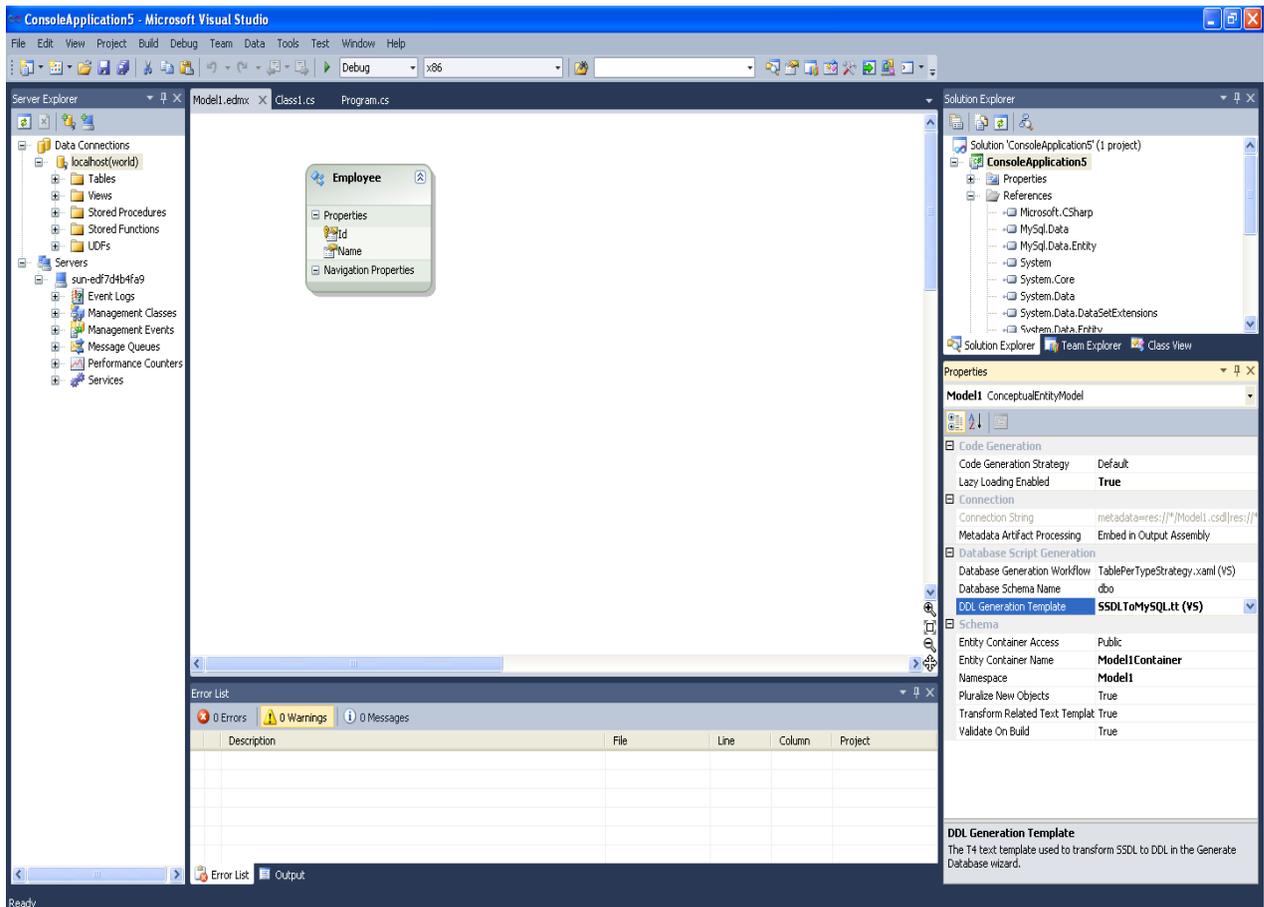


The results from any queries are displayed on the **Results** tab. Any errors are displayed on the **Messages** tab.

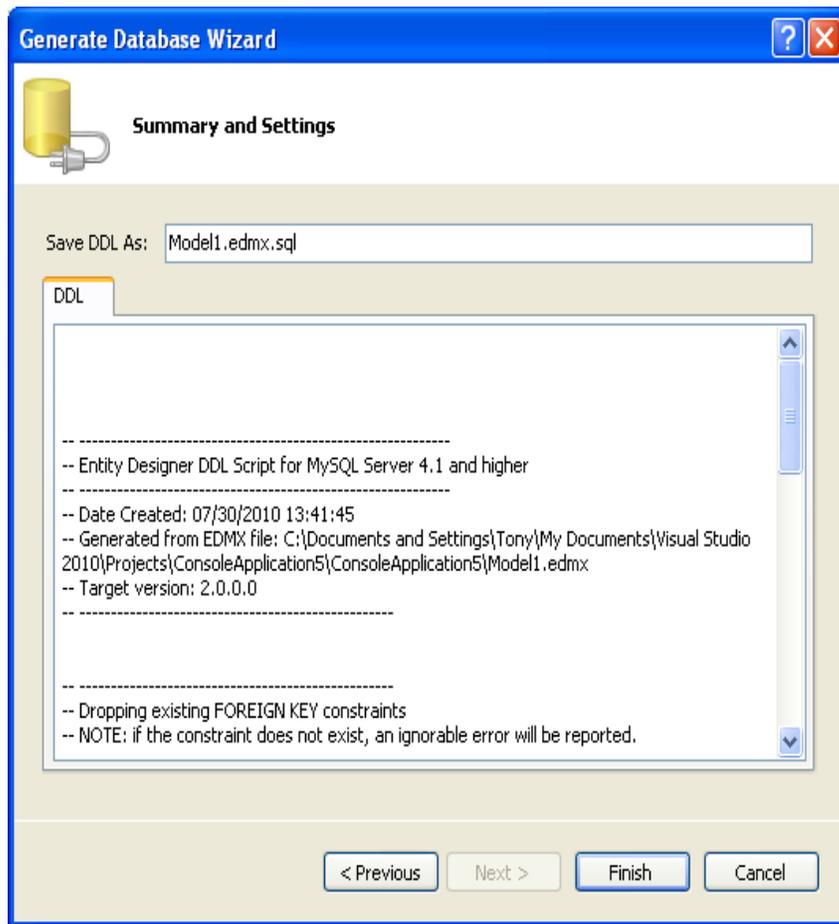
22.2.3.14. DDL T4 Template Macro

MySQL Connector/Net 6.3 introduced the ability to convert an Entity Framework model to MySQL **DDL** code. Starting with a blank model, you can develop an entity model in Visual Studio's designer. Once the model is created, you can select the model's properties, and in the Database Script Generation category of the model's properties, the property **DDL Generation** can be found. Select the value **SSDLToMySQL.tt(VS)** from the drop-down listbox.

Figure 22.39. DDL T4 Template Macro - Model Properties



Right-clicking the model design area displays a context-sensitive menu. Selecting **Generate Database from Model** from the menu displays the **Generate Database Wizard**. The wizard can then be used to generate MySQL DDL code.

Figure 22.40. DDL T4 Template Macro - Generate Database Wizard

22.2.4. Connector/Net Tutorials

The following tutorials illustrate how to develop MySQL programs using technologies such as Visual Studio, C#, ASP.NET, and the .NET and Mono frameworks. Work through the first tutorial to verify that you have the right software components installed and configured, then choose other tutorials to try depending on the features you intend to use in your applications.

22.2.4.1. Tutorial: An Introduction to Connector/Net Programming

This section provides a gentle introduction to programming with Connector/Net. The example code is written in C#, and is designed to work on both Microsoft .NET Framework and Mono.

This tutorial is designed to get you up and running with Connector/Net as quickly as possible, it does not go into detail on any particular topic. However, the following sections of this manual describe each of the topics introduced in this tutorial in more detail. In this tutorial you are encouraged to type in and run the code, modifying it as required for your setup.

This tutorial assumes you have MySQL and Connector/Net already installed. It also assumes that you have installed the World example database, which can be downloaded from the [MySQL Documentation page](#). You can also find details on how to install the database on the same page.

**Note**

Before compiling the example code, make sure that you have added References to your project as required. The References required are [System](#), [System.Data](#) and [MySql.Data](#).

22.2.4.1.1. The MySqlConnection Object

For your Connector/Net application to connect to a MySQL database, it must establish a connection by using a [MySqlConnection](#) object.

The [MySqlConnection](#) constructor takes a connection string as one of its parameters. The connection string provides necessary information to make the connection to the MySQL database. The connection string is discussed more fully in [Section 22.2.5.1, "Connecting to MySQL Using Connector/Net"](#). For a list of supported connection string options, see [Section 22.2.6, "Connector/Net Connection String Options Reference"](#).

The following code shows how to create a connection object:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial1
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();
            // Perform database operations
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

When the [MySqlConnection](#) constructor is invoked, it returns a connection object, which is used for subsequent database operations. Open the connection before any other operations take place. Before the application exits, close the connection to the database by calling [Close](#) on the connection object.

Sometimes an attempt to perform an [Open](#) on a connection object can fail, generating an exception that can be handled using standard exception handling code.

In this section you have learned how to create a connection to a MySQL database, and open and close the corresponding connection object.

22.2.4.1.2. The MySqlCommand Object

Once a connection has been established with the MySQL database, the next step is to carry out the desired database operations. This can be achieved through the use of the [MySqlCommand](#) object.

You will see how to create a [MySqlCommand](#) object. Once it has been created, there are three main methods of interest that you can call:

- **ExecuteReader** - used to query the database. Results are usually returned in a [MySqlDataReader](#) object, created by [ExecuteReader](#).
- **ExecuteNonQuery** - used to insert and delete data.
- **ExecuteScalar** - used to return a single value.

Once a [MySqlCommand](#) object has been created, you will call one of the above methods on it to carry out a database operation, such as perform a query. The results are usually returned into a [MySqlDataReader](#) object, and then processed, for example the results might be displayed. The following code demonstrates how this could be done.

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial2
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'";
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            MySqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine(rdr[0]+" -- "+rdr[1]);
            }
            rdr.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

When a connection has been created and opened, the code then creates a [MySqlCommand](#) object. Note that the SQL query to be executed is passed to the [MySqlCommand](#) constructor. The [ExecuteReader](#) method is then used to generate a [MySqlReader](#) object. The [MySqlReader](#) object contains the results generated by the SQL executed on the command object. Once the results have been obtained in a [MySqlReader](#) object, the results can be processed. In this case, the information is printed out by a [while](#) loop. Finally, the [MySqlReader](#) object is disposed of by running its [Close](#) method on it.

In the next example, you will see how to use the [ExecuteNonQuery](#) method.

The procedure for performing an `ExecuteNonQuery` method call is simpler, as there is no need to create an object to store results. This is because `ExecuteNonQuery` is only used for inserting, updating and deleting data. The following example illustrates a simple update to the `Country` table:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial3
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "INSERT INTO Country (Name, HeadOfState, Continent) VALUES ('Disneyland', 'Mickey Mouse', 'USA')";
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

The query is constructed, the command object created and the `ExecuteNonQuery` method called on the command object. You can access your MySQL database with the `mysql` command interpreter and verify that the update was carried out correctly.

Finally, you will see how the `ExecuteScalar` method can be used to return a single value. Again, this is straightforward, as a `MySqlDataReader` object is not required to store results, a simple variable will do. The following code illustrates how to use `ExecuteScalar`:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial4
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT COUNT(*) FROM Country";
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            object result = cmd.ExecuteScalar();
            if (result != null)
            {
                Console.WriteLine(result);
            }
        }
    }
}
```

```
        int r = Convert.ToInt32(result);
        Console.WriteLine("Number of countries in the World database is: " + r);
    }

}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}

conn.Close();
Console.WriteLine("Done.");
}
}
```

This example uses a simple query to count the rows in the [Country](#) table. The result is obtained by calling [ExecuteScalar](#) on the command object.

22.2.4.1.3. Working with Decoupled Data

Previously, when using [MySqlDataReader](#), the connection to the database was continually maintained, unless explicitly closed. It is also possible to work in a manner where a connection is only established when needed. For example, in this mode, a connection could be established to read a chunk of data, the data could then be modified by the application as required. A connection could then be reestablished only if and when the application writes data back to the database. This decouples the working data set from the database.

This decoupled mode of working with data is supported by Connector/Net. There are several parts involved in allowing this method to work:

- **Data Set** - The Data Set is the area in which data is loaded to read or modify it. A [DataSet](#) object is instantiated, which can store multiple tables of data.
- **Data Adapter** - The Data Adapter is the interface between the Data Set and the database itself. The Data Adapter is responsible for efficiently managing connections to the database, opening and closing them as required. The Data Adapter is created by instantiating an object of the [MySqlDataAdapter](#) class. The [MySqlDataAdapter](#) object has two main methods: [Fill](#) which reads data into the Data Set, and [Update](#), which writes data from the Data Set to the database.
- **Command Builder** - The Command Builder is a support object. The Command Builder works in conjunction with the Data Adapter. When a [MySqlDataAdapter](#) object is created, it is typically given an initial [SELECT](#) statement. From this [SELECT](#) statement the Command Builder can work out the corresponding [INSERT](#), [UPDATE](#) and [DELETE](#) statements that would be required to update the database. To create the Command Builder, an object of the class [MySqlCommandBuilder](#) is created.

Each of these classes will now be discussed in more detail.

Instantiating a DataSet object

A [DataSet](#) object can be created simply, as shown in the following example code snippet:

```
DataSet dsCountry;
...
dsCountry = new DataSet();
```

Although this creates the [DataSet](#) object, it has not yet filled it with data. For that, a Data Adapter is required.

Instantiating a MySqlDataAdapter object

The `MySqlDataAdapter` can be created as illustrated by the following example:

```
MySqlDataAdapter daCountry;  
...  
string sql = "SELECT Code, Name, HeadOfState FROM Country WHERE Continent='North America'";  
daCountry = new MySqlDataAdapter (sql, conn);
```

Note, the `MySqlDataAdapter` is given the SQL specifying the data to work with.

Instantiating a `MySqlCommandBuilder` object

Once the `MySqlDataAdapter` has been created, it is necessary to generate the additional statements required for inserting, updating and deleting data. There are several ways to do this, but in this tutorial you will see how this can most easily be done with `MySqlCommandBuilder`. The following code snippet illustrates how this is done:

```
MySqlCommandBuilder cb = new MySqlCommandBuilder(daCountry);
```

Note that the `MySqlDataAdapter` object is passed as a parameter to the command builder.

Filling the Data Set

To do anything useful with the data from your database, you need to load it into a Data Set. This is one of the jobs of the `MySqlDataAdapter` object, and is carried out with its `Fill` method. The following example code illustrates this:

```
DataSet dsCountry;  
...  
dsCountry = new DataSet();  
...  
daCountry.Fill(dsCountry, "Country");
```

Note the `Fill` method is a `MySqlDataAdapter` method, the Data Adapter knows how to establish a connection with the database and retrieve the required data, and then populates the Data Set when the `Fill` method is called. The second parameter "Country" is the table in the Data Set to update.

Updating the Data Set

The data in the Data Set can now be manipulated by the application as required. At some point, changes to data will need to be written back to the database. This is achieved through a `MySqlDataAdapter` method, the `Update` method.

```
daCountry.Update(dsCountry, "Country");
```

Again, the Data Set and the table within the Data Set to update are specified.

Working Example

The interactions between the `DataSet`, `MySqlDataAdapter` and `MySqlCommandBuilder` classes can be a little confusing, so their operation can perhaps be best illustrated by working code.

In this example, data from the World database is read into a Data Grid View control. Here, the data can be viewed and changed before clicking an update button. The update button then activates code to write changes back to the database. The code uses the principles explained above. The application was built using the Microsoft Visual Studio to place and create the user interface controls, but the main code that uses the key classes described above is shown below, and is portable.

```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using MySql.Data;
using MySql.Data.MySqlClient;

namespace WindowsFormsApplication5
{
    public partial class Form1 : Form
    {
        MySqlDataAdapter daCountry;
        DataSet dsCountry;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);
            try
            {
                label2.Text = "Connecting to MySQL...";

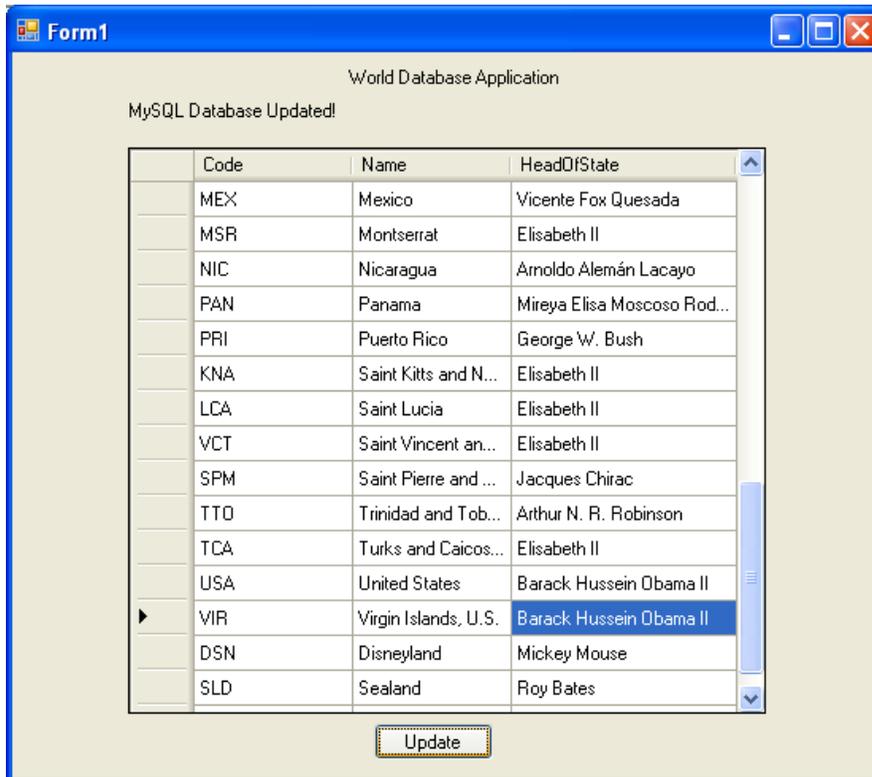
                string sql = "SELECT Code, Name, HeadOfState FROM Country WHERE Continent='North America'";
                daCountry = new MySqlDataAdapter (sql, conn);
                MySqlCommandBuilder cb = new MySqlCommandBuilder(daCountry);

                dsCountry = new DataSet();
                daCountry.Fill(dsCountry, "Country");
                dataGridView1.DataSource = dsCountry;
                dataGridView1.DataMember = "Country";
            }
            catch (Exception ex)
            {
                label2.Text = ex.ToString();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            daCountry.Update(dsCountry, "Country");
            label2.Text = "MySQL Database Updated!";
        }
    }
}
```

The application running is shown below:

Figure 22.41. World Database Application



22.2.4.1.4. Working with Parameters

This part of the tutorial shows you how to use parameters in your Connector/Net application.

Although it is possible to build SQL query strings directly from user input, this is not advisable as it does not prevent erroneous or malicious information being entered. It is safer to use parameters as they will be processed as field data only. For example, imagine the following query was constructed from user input:

```
string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent = "+user_continent;
```

If the string `user_continent` came from a Text Box control, there would potentially be no control over the string entered by the user. The user could enter a string that generates a run time error, or in the worst case actually harms the system. When using parameters it is not possible to do this because a parameter is only ever treated as a field parameter, rather than an arbitrary piece of SQL code.

The same query written user a parameter for user input would be:

```
string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent = @Continent";
```

Note that the parameter is preceded by an '@' symbol to indicate it is to be treated as a parameter.

As well as marking the position of the parameter in the query string, it is necessary to add a parameter to the Command object. This is illustrated by the following code snippet:

```
cmd.Parameters.AddWithValue("@Continent", "North America");
```

In this example the string "North America" is supplied as the parameter value statically, but in a more practical example it would come from a user input control.

A further example illustrates the complete process:

```

using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial5
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent=@Continent";
            MySqlCommand cmd = new MySqlCommand(sql, conn);

            Console.WriteLine("Enter a continent e.g. 'North America', 'Europe': ");
            string user_input = Console.ReadLine();

            cmd.Parameters.AddWithValue("@Continent", user_input);

            MySqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine(rdr["Name"]+" --- "+rdr["HeadOfState"]);
            }
            rdr.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}

```

In this part of the tutorial you have see how to use parameters to make your code more secure.

22.2.4.1.5. Working with Stored Procedures

This section illustrates how to work with stored procedures. Putting database-intensive operations into stored procedures lets you define an API for your database application. You can reuse this API across multiple applications and multiple programming languages. This technique avoids duplicating database code, saving time and effort when you make updates due to schema changes, tune the performance of queries, or add new database operations for logging, security, and so on. Before working through this tutorial, familiarize yourself with the [CREATE PROCEDURE](#) and [CREATE FUNCTION](#) statements that create different kinds of stored routines.

For the purposes of this tutorial, you will create a simple stored procedure to see how it can be called from Connector/Net. In the MySQL Client program, connect to the World database and enter the following stored procedure:

```

DELIMITER //
CREATE PROCEDURE country_hos
(IN con CHAR(20))
BEGIN
    SELECT Name, HeadOfState FROM Country
    WHERE Continent = con;

```

```
END //  
DELIMITER ;
```

Test that the stored procedure works as expected by typing the following into the `mysql` command interpreter:

```
CALL country_hos('Europe');
```

Note that The stored routine takes a single parameter, which is the continent to restrict your search to.

Having confirmed that the stored procedure is present and correct, you can see how to access it from Connector/Net.

Calling a stored procedure from your Connector/Net application is similar to techniques you have seen earlier in this tutorial. A `MySqlCommand` object is created, but rather than taking an SQL query as a parameter, it takes the name of the stored procedure to call. Set the `MySqlCommand` object to the type of stored procedure, as shown by the following code snippet:

```
string rtn = "country_hos";  
MySqlCommand cmd = new MySqlCommand(rtn, conn);  
cmd.CommandType = CommandType.StoredProcedure;
```

In this case, the stored procedure requires you to pass a parameter. This can be achieved using the techniques seen in the previous section on parameters, [Section 22.2.4.1.4, "Working with Parameters"](#), as shown in the following code snippet:

```
cmd.Parameters.AddWithValue("@con", "Europe");
```

The value of the parameter `@con` could more realistically have come from a user input control, but for simplicity it is set as a static string in this example.

At this point, everything is set up and you can call the routine using techniques also learned in earlier sections. In this case, the `ExecuteReader` method of the `MySqlCommand` object is used.

Complete working code for the stored procedure example is shown below:

```
using System;  
using System.Data;  
  
using MySql.Data;  
using MySql.Data.MySqlClient;  
  
public class Tutorial6  
{  
    public static void Main()  
    {  
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";  
        MySqlConnection conn = new MySqlConnection(connStr);  
        try  
        {  
            Console.WriteLine("Connecting to MySQL...");  
            conn.Open();  
  
            string rtn = "country_hos";  
            MySqlCommand cmd = new MySqlCommand(rtn, conn);  
            cmd.CommandType = CommandType.StoredProcedure;  
  
            cmd.Parameters.AddWithValue("@con", "Europe");  
  
            MySqlDataReader rdr = cmd.ExecuteReader();  
            while (rdr.Read())  
            {  
                Console.WriteLine(rdr[0] + " --- " + rdr[1]);  
            }  
            rdr.Close();  
        }  
    }  
}
```

```
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }

    conn.Close();
    Console.WriteLine("Done.");
}
}
```

In this section, you have seen how to call a stored procedure from Connector/Net. For the moment, this concludes our introductory tutorial on programming with Connector/Net.

22.2.4.2. Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider

Many web sites feature the facility for the user to create a user account. They can then log into the web site and enjoy a personalized experience. This requires that the developer creates database tables to store user information, along with code to gather and process this data. This represents a burden on the developer, and there is the possibility for security issues to creep into the developed code. However, ASP.NET 2.0 introduced the Membership system. This system is designed around the concept of Membership, Profile and Role Providers, which together provide all of the functionality to implement a user system, that previously would have to have been created by the developer from scratch.

Currently, MySQL Connector/Net provides Membership, Role, Profile and Session State Providers.

This tutorial shows you how to set up your ASP.NET web application to use the MySQL Connector/Net Membership and Role Providers. It assumes that you have MySQL Server installed, along with MySQL Connector/Net and Microsoft Visual Studio. This tutorial was tested with MySQL Connector/Net 6.0.4 and Microsoft Visual Studio 2008 Professional Edition. It is recommended you use 6.0.4 or above for this tutorial.

1. Create a new database in the MySQL Server using the MySQL Command Line Client program ([mysql](#)), or other suitable tool. It does not matter what name is used for the database, but record it. You specify it in the connection string constructed later in this tutorial. This database contains the tables, automatically created for you later, used to store data about users and roles.
2. Create a new ASP.NET Web Site in Visual Studio. If you are not sure how to do this, refer to [Section 22.2.4.6, "Tutorial: Databinding in ASP.NET using LINQ on Entities"](#), which demonstrates how to create a simple ASP.NET web site.
3. Add References to [MySQL.Data](#) and [MySQL.Web](#) to the web site project.
4. Locate the [machine.config](#) file on your system, which is the configuration file for the .NET Framework.
5. Search the [machine.config](#) file to find the membership provider [MySQLMembershipProvider](#).
6. Add the attribute `autogenerateschema="true"`. The appropriate section should now resemble the following (note: for the sake of brevity some information has been excluded):

```
<membership>
  <providers>
    <add name="AspNetSqlMembershipProvider"
        type="System.Web.Security.SqlMembershipProvider"
        ...
        connectionStringName="LocalSqlServer"
        ... />
    <add name="MySQLMembershipProvider"
        autogenerateschema="true"
        type="MySQL.Web.Security.MySQLMembershipProvider, MySQL.Web, Version=6.0.4.0, Culture=neutral, Pub
```

```
connectionStringName="LocalMySQLServer"
... />
</providers>
</membership>
```

Note that the name for the connection string to be used to connect to the server that contains the membership database is `LocalMySQLServer`.

The `autogenerateschema="true"` attribute will cause MySQL Connector/Net to silently create, or upgrade, the schema on the database server, to contain the required tables for storing membership information.

7. It is now necessary to create the connection string referenced in the previous step. Load the web site's `web.config` file into Visual Studio.
8. Locate the section marked `<connectionStrings>`. Add the following connection string information:

```
<connectionStrings>
  <remove name="LocalMySQLServer" />
  <add name="LocalMySQLServer"
    connectionString="Datasource=localhost;Database=users;uid=root;pwd=password;"
    providerName="MySql.Data.MySqlClient" />
</connectionStrings>
```

The database specified is the one created in the first step. You could alternatively have used an existing database.

9. At this point build the solution to ensure no errors are present. This can be done by selecting **Build**, **Build Solution** from the main menu, or pressing **F6**.
10. ASP.NET supports the concept of locally and remotely authenticated users. With local authentication the user is validated using their Windows credentials when they attempt to access the web site. This can be useful in an Intranet environment. With remote authentication, a user is prompted for their login details when accessing the web site, and these credentials are checked against the membership information stored in a database server such as MySQL Server. You will now see how to choose this form of authentication.

Start the ASP.NET Web Site Administration Tool. This can be done quickly by clicking the small hammer/Earth icon in the Solution Explorer. You can also launch this tool by selecting **Website**, **ASP.NET Configuration** from the main menu.

11. In the ASP.NET Web Site Administration Tool click the **Security** tab.
12. Now click the **User Authentication Type** link.
13. Select the **From the internet** radio button. The web site will now need to provide a form to allow the user to enter their login details. These will be checked against membership information stored in the MySQL database.

Figure 22.42. Authentication Type

ASP.NET Web Site Administration Tool

Home Security Application Provider

How will users access your site?

From the internet
 Select this option if users will access your web site from the public internet. Users will be required to log on using a web form. The site will use forms authentication to identify users according to user information that you store in a database.

From a local network
 Select this option if users will access your web site only from a private local network. The site will use built-in Microsoft Windows authentication to identify users. Users with a valid Windows user name and password will be able to access your site.

14. You now need to specify the Role and Membership Provider to be used. Click the **Provider** tab.
15. Click the **Select a different provider for each feature (advanced)** link.
16. Now select the **MySQLMembershipProvider** and the **MySQLRoleProvider** radio buttons.

Figure 22.43. Select Membership and Role Provider

ASP.NET Web Site Administration Tool

Home Security Application **Provider**

Use this page to select a provider for each feature.

Membership Provider

AspNetSqlMembershipProvider [Test](#)

MySQLMembershipProvider

Role Provider

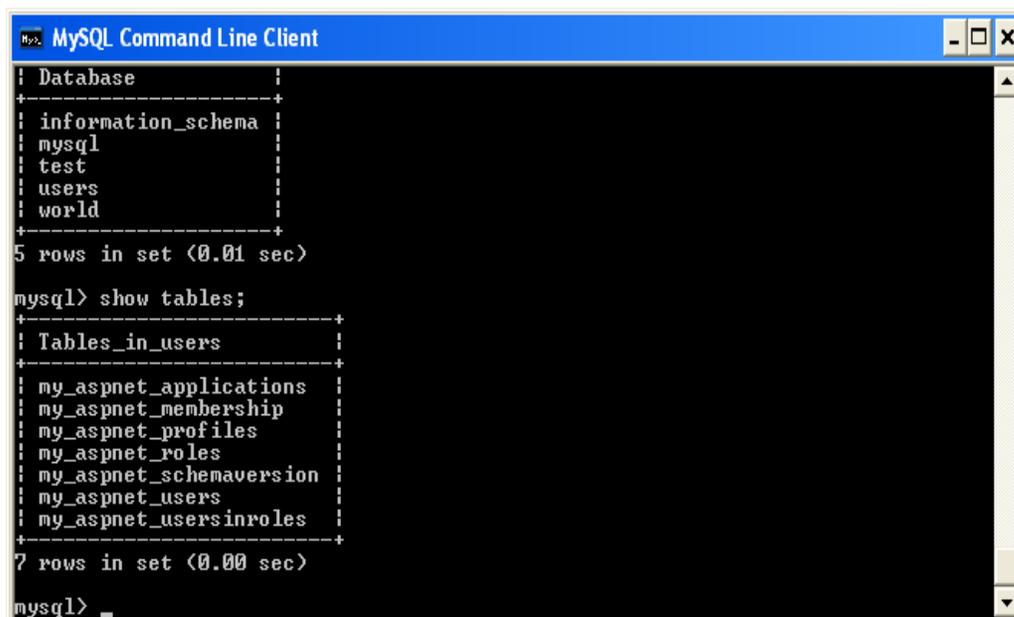
AspNetSqlRoleProvider [Test](#)

AspNetWindowsTokenRoleProvider

MySQLRoleProvider

17. In Visual Studio, rebuild the solution by selecting **Build**, **Rebuild Solution** from the main menu.
18. Check that the necessary schema has been created. This can be achieved using the `mysql` command interpreter.

Figure 22.44. Membership and Role Provider Tables

A screenshot of the MySQL Command Line Client window. The window title is "MySQL Command Line Client". The terminal shows the following output:

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| test |
| users |
| world |
+-----+
5 rows in set (0.01 sec)

mysql> show tables;
+-----+
| Tables_in_users |
+-----+
| my_aspnet_applications |
| my_aspnet_membership |
| my_aspnet_profiles |
| my_aspnet_roles |
| my_aspnet_schemaversion |
| my_aspnet_users |
| my_aspnet_usersinroles |
+-----+
7 rows in set (0.00 sec)

mysql>
```

19. Assuming all is present and correct, you can now create users and roles for your web application. The easiest way to do this is with the ASP.NET Web Site Administration Tool. However, many web applications contain their own modules for creating roles and users. For simplicity, the ASP.NET Web Site Administration Tool will be used in this tutorial.
20. In the ASP.NET Web Site Administration Tool, click the **Security** tab. Now that both the Membership and Role Provider are enabled, you will see links for creating roles and users. Click the **Create or Manage Roles** link.

Figure 22.45. Security Tab

ASP.NET Web Site Administration Tool

Home Security Application Provider

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

[Use the security Setup Wizard to configure security step by step.](#)

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: 1 Create user Manage users Select authentication type	Existing roles: 2 Disable Roles Create or Manage roles	Create access rules Manage access rules

21. You can now enter the name of a new Role and click **Add Role** to create the new Role. Create new Roles as required.
22. Click the **Back** button.
23. Click the **Create User** link. You can now fill in information about the user to be created, and also allocate that user to one or more Roles.

Figure 22.46. Create User

ASP.NET Web Site Administration Tool

Home Security Application Provider

Add a user by entering the user's ID, password, and e-mail address on this page.

Create User

Sign Up for Your New Account

User Name: johndoe

Password: ●●●●●●●●

Confirm Password: ●●●●●●●●

E-mail: johndoe@example.com

Security Question: Statler and ...

Security Answer: Waldorf

Roles

Select roles for this user:

Visitors

Admins

Active User

24. Using the `mysql` command interpreter, you can check that your database has been correctly populated with the Membership and Role data.

Figure 22.47. Membership and Roles Table Contents

```

mysql> select * from my_aspnet_users;
+----+-----+-----+-----+-----+
| id | applicationId | name | isAnonymous | lastActivityDate |
+----+-----+-----+-----+-----+
| 1 | 1 | johndoe | 0 | 2009-07-14 17:17:25 |
+----+-----+-----+-----+-----+
1 row in set (0.01 sec)

mysql> select * from my_aspnet_roles;
+----+-----+-----+
| id | applicationId | name |
+----+-----+-----+
| 1 | 1 | Visitors |
| 2 | 1 | Admins |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

In this tutorial, you have seen how to set up the MySQL Connector/Net Membership and Role Providers for use in your ASP.NET web application.

22.2.4.3. Tutorial: MySQL Connector/Net ASP.NET Session State Provider

MySQL Connector/Net from version 6.1 has included a MySQL Session State Provider. This provider enables you to store session state in a MySQL database. The following tutorial shows you how to prepare to use the MySQL Session State Provider, and then store session data into the MySQL database. This tutorial uses Microsoft Visual Studio 2008 Professional Edition, MySQL Connector/Net 6.1.1 and MySQL Server 5.1. This tutorial also assumes you have created an empty database, for example `test`, where you will store session data. You could do this using the `mysql` command interpreter.

1. In Visual Studio, create a new ASP.NET web site. If you are not sure how to do this, refer to the tutorial [Section 22.2.4.6, "Tutorial: Databinding in ASP.NET using LINQ on Entities"](#), which demonstrates how to do this.
2. Launch the MySQL Website Configuration tool. Due to a bug in 6.1.1, this may not appear unless you are connected to a server in the Server Explorer. If you are unfamiliar with the MySQL Website Configuration tool, consider first working through the tutorial in [Section 22.2.3.12, "MySQL Website Configuration Tool"](#).
3. Navigate through the wizard to the Session State page. Make sure the check box **Use MySQL to manage my ASP.NET session data** is selected.
4. On the same page, configure the connection string to the database that will contain your session data. If this database is empty, MySQL Connector/Net will create the schema required to store session data.
5. Ensure that the check box **Autogenerate Schema** is selected so that MySQL Connector/Net will create the schema in your database to store the session data correctly.
6. Enter the name of your application.
7. Click **Finish**. The MySQL Website Configuration tool will now update your application's `web.config` file with information about the connection string and default providers to be used. In this case, we have selected the MySQL Session State Provider.

At this point, you are ready to use the MySQL database to store session data. To test that the set up has worked, you can write a simple program that uses session variables.

1. Open `Default.aspx.cs`. In the `Page_Load` method, add the following code:

```
Session["SessionVariable1"] = "Test string";
```

2. Build your solution.
3. Run the solution (without debugging). When the application runs, the provider will autogenerate tables required in the database you chose when setting up the application.
4. Check that the schema was in fact created. Using the MySQL Command Line Client use the target database and then type `SHOW TABLES;`. You will see that MySQL Connector/Net has created the required schema automatically, as we selected this to happen in the MySQL Website Configuration tool.
5. Now view the contents of these tables by typing `SELECT * FROM my_aspnet_sessions;` in the `mysql` command interpreter. This will display the session data our application used. Note that this is stored in binary format so some data may not display as expected.

At this point, you have installed the Session State Provider and carried out a preliminary test of the installation. You will now work a bit more with the Session State Provider.

In this part of the tutorial, you will set and retrieve a session variable. You can work with your existing project.

1. Select the `Default.aspx` and switch to Design View. Add a text box and three buttons. Change the text property for the buttons to “Store Session Variable”, “Clear Textbox”, and “Show Session Variable”. These will be `Button1`, `Button2` and `Button3` respectively. Build your solution to ensure that no errors have been introduced.
2. Still in the Design View, double-click `Button1`. Now to the `Button1_Click` event handler add code some the handler resembles the following:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Session["SessionString"] = TextBox1.Text;
}
```

You have created a new Session variable accessed using the key “SessionString”. This will be set to the text that was entered into the text box when `Button1` is clicked.

3. In Design View, double-click `Button2` to add its click event handler. This button needs to clear text from the text box. The code to do this is as follows:

```
protected void Button2_Click(object sender, EventArgs e)
{
    TextBox1.Text = "";
}
```

The code simply assigns an empty string to the `Text` property of the text box.

4. In the Design View double-click `Button3` and modify the click handler as follows:

```
protected void Button3_Click(object sender, EventArgs e)
{
    TextBox1.Text = (String)Session["SessionString"];
}
```

This will retrieve the session string and display it in the text box.

5. Now modify the `Page_Load` method as follows:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        TextBox1.Text = "Enter some text";
    }
}
```

This ensures that when the page loads the text box `Text` property is reset.

6. Ensure that the solution is saved and then rebuild the solution.
7. Run the solution without debugging.
8. The form will be displayed. Enter some text into the text box. Now click `Store Session Variable`. At this point you have stored the string in a session variable.
9. Now click `Clear Text` to clear the text box.
10. Now click `Show Session Variable` to retrieve and display the session variable.
11. Refresh the page to destroy the form and display a new form.
12. Click `Show Session Variable` the text box will display the stored session variable, demonstrating that the refreshing the page does not destroy the session variable.

This illustrates that the session state data is not destroyed when a page is reloaded.

22.2.4.4. Tutorial: MySQL Connector/Net ASP.NET Profile Provider

This tutorial shows you how to use the MySQL Profile Provider to store user profile information in a MySQL database. The tutorial uses MySQL Connector/Net 6.1.1, MySQL Server 5.1 and Microsoft Visual Studio 2008 Professional Edition.

Many modern web sites allow the user to create a personal profile. This requires a significant amount of code, but ASP.NET reduces this considerable by including the functionality in its Profile classes. The Profile Provider provides an abstraction between these classes and a data source. The MySQL Profile Provider enables profile data to be stored in a MySQL database. This enables the profile properties to be written to a persistent store, and be retrieved when required. The Profile Provider also enables profile data to be managed effectively, for example it enables profiles that have not been accessed since a specific date to be deleted.

The following steps show you how you can select the MySQL Profile Provider.

1. Create a new ASP.NET web project.
2. Select the MySQL Website Configuration tool. Due to a bug in 6.1.1 you may have to first connect to a server in Server Explorer before the tool's icon will display in the toolbar of the Solution Explorer.
3. In the MySQL Website Configuration tool navigate through the tool to the Profiles page.
4. Select the **Use MySQL to manage my profiles** check box.
5. Select the **Autogenerate Schema** check box.
6. Click the **Edit...** button and configure a connection string for the database that will be used to store user profile information.
7. Navigate to the last page of the tool and click **Finish** to save your changes and exit the tool.

At this point you are now ready to start using the MySQL Profile Provider. With the following steps you can carry out a preliminary test of your installation.

1. Open your `web.config` file.
2. Add a simple profile such as the following:

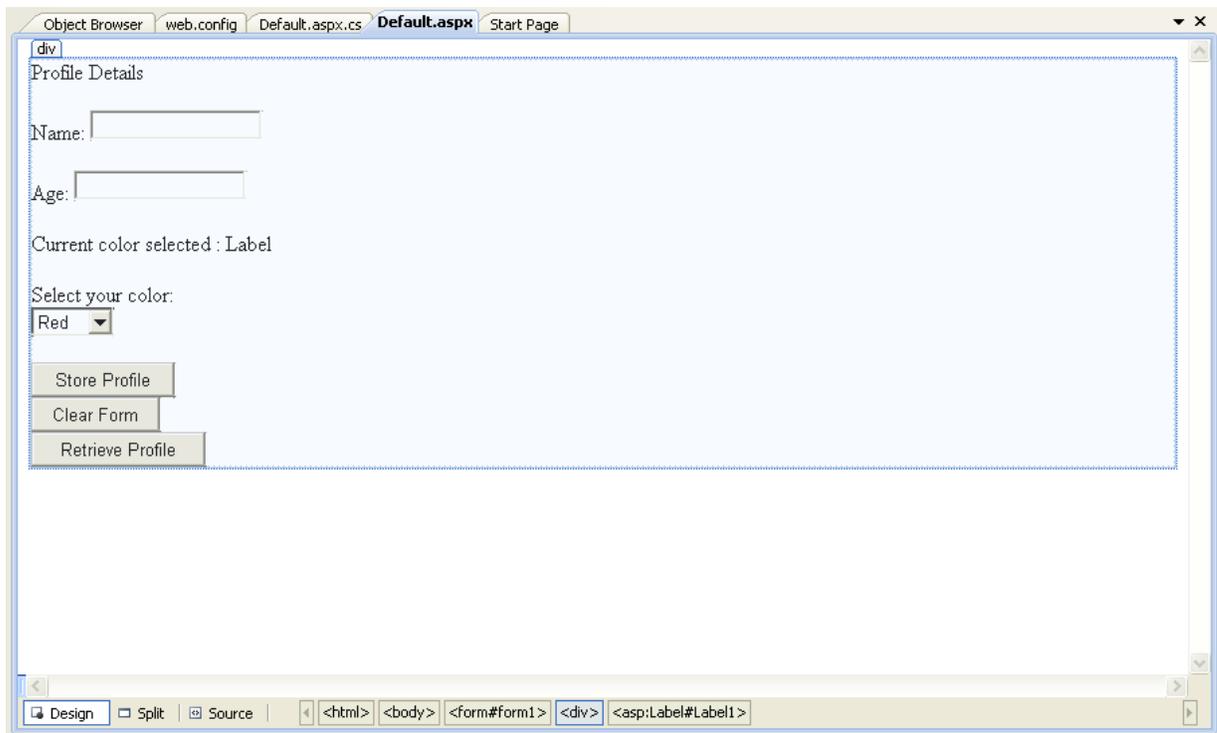
```
<system.web>
  <anonymousIdentification enabled="true"/>
  <profile defaultProvider="MySQLProfileProvider">
    ...
    <properties>
      <add name="Name" allowAnonymous="true"/>
      <add name="Age" allowAnonymous="true" type="System.UInt16"/>
      <group name="UI">
        <add name="Color" allowAnonymous="true" defaultValue="Blue"/>
        <add name="Style" allowAnonymous="true" defaultValue="Plain"/>
      </group>
    </properties>
  </profile>
  ...
```

Note that `anonymousIdentification` has been set to true. This enables users who have not been authenticated to use profiles. They are identified by a GUID in a cookie rather than by user name.

Now that the simple profile has been defined in `web.config`, the next step is to write some code to test the profile.

1. In Design View design a simple page with the following controls:

Figure 22.48. Simple Profile Application



These will allow the user to enter some profile information. The user can also use the buttons to save their profile, clear the page, and restore their profile data.

2. In the Code View add code as follows:

```
...
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        TextBox1.Text = Profile.Name;
        TextBox2.Text = Profile.Age.ToString();
        Label1.Text = Profile.UI.Color;
    }
}

// Store Profile
protected void Button1_Click(object sender, EventArgs e)
{
    Profile.Name = TextBox1.Text;
    Profile.Age = UInt16.Parse(TextBox2.Text);
}

// Clear Form
protected void Button2_Click(object sender, EventArgs e)
{
    TextBox1.Text = "";
    TextBox2.Text = "";
    Label1.Text = "";
}

// Retrieve Profile
```

```
protected void Button3_Click(object sender, EventArgs e)
{
    TextBox1.Text = Profile.Name;
    TextBox2.Text = Profile.Age.ToString();
    Label1.Text = Profile.UI.Color;
}

protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    Profile.UI.Color = DropDownList1.SelectedValue;
}
...
```

3. Save all files and build the solution to check that no errors have been introduced.
4. Run the application.
5. Enter your name, age and select a color from the listbox. Now store this information in your profile by clicking **Store Profile**. Note that if you do not select a color from the listbox your profile will use the default color **Blue** that was specified in the `web.config` file.
6. Click **Clear Form** to clear text from the textboxes and the label that displays your chosen color.
7. Now click **Retrieve Profile** to restore your profile data from the MySQL database.
8. Now exit the browser to terminate the application.
9. Run the application again. Note that when the page loads your profile information is restored from the MySQL database.

In this tutorial you have seen how to using the MySQL Profile Provider with MySQL Connector/Net.

22.2.4.5. Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source

In this tutorial you will learn how to create a Windows Forms Data Source from an Entity in an Entity Data Model. This tutorial assumes that you have installed the World example database, which can be downloaded from the [MySQL Documentation page](#). You can also find details on how to install the database on the same page. It will also be convenient for you to create a connection to the World database after it is installed. For instructions on how to do this see [Section 22.2.3.1, "Making a Connection"](#).

Creating a new Windows Forms application

The first step is to create a new Windows Forms application.

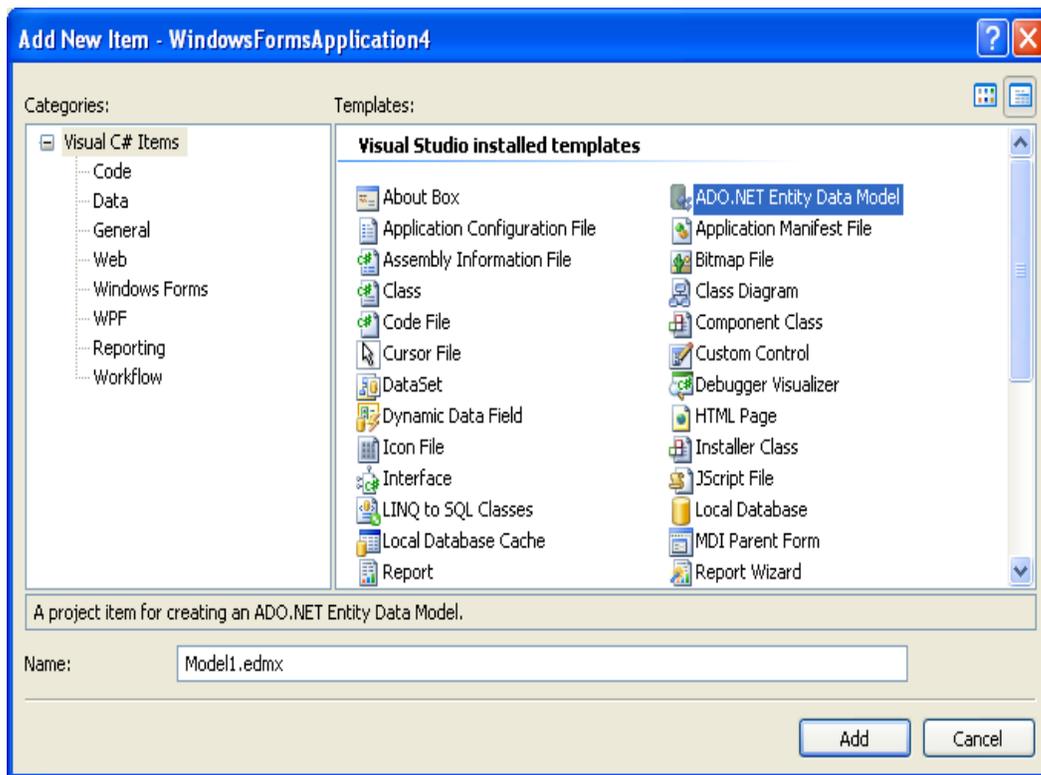
1. In Visual Studio, select **File, New, Project** from the main menu.
2. Choose the **Windows Forms Application** installed template. Click **OK**. The solution is created.

Adding an Entity Data Model

You will now add an Entity Data Model to your solution.

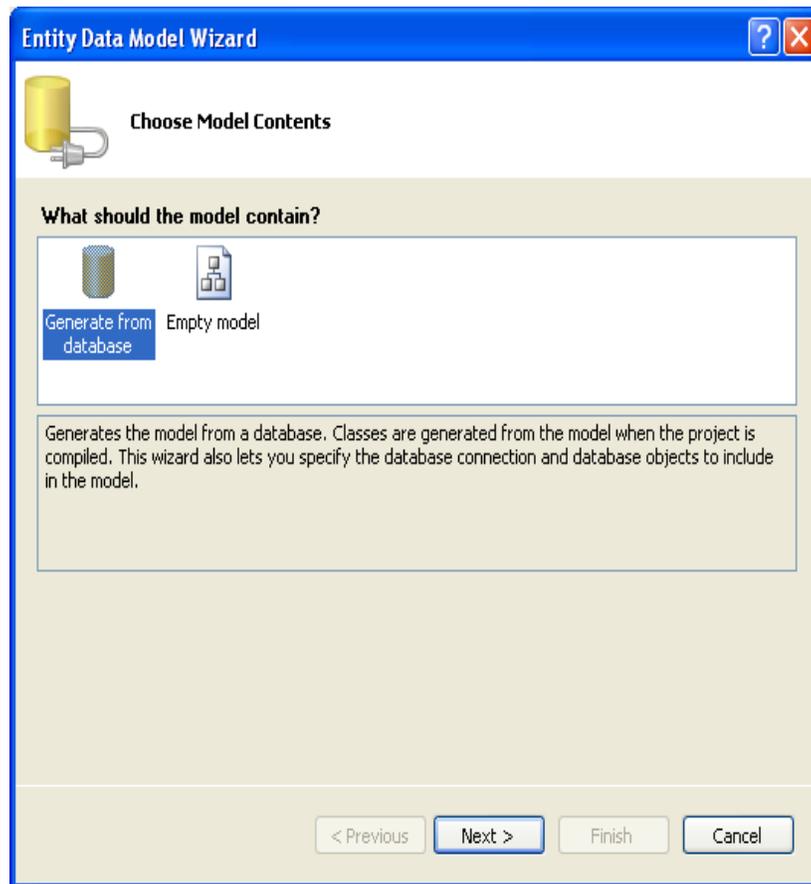
1. In the Solution Explorer, right-click your application and select **Add, New Item...**. From **Visual Studio installed templates** select **ADO.NET Entity Data Model**. Click **Add**.

Figure 22.49. Add Entity Data Model



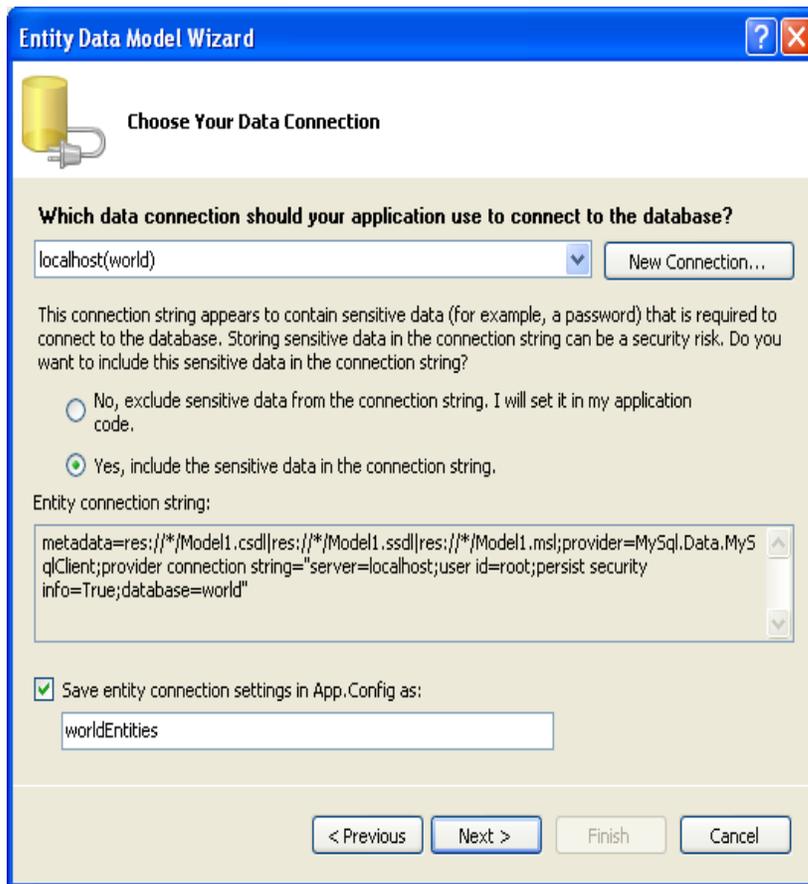
2. You will now see the Entity Data Model Wizard. You will use the wizard to generate the Entity Data Model from the world example database. Select the icon **Generate from database**. Click **Next**.

Figure 22.50. Entity Data Model Wizard Screen 1

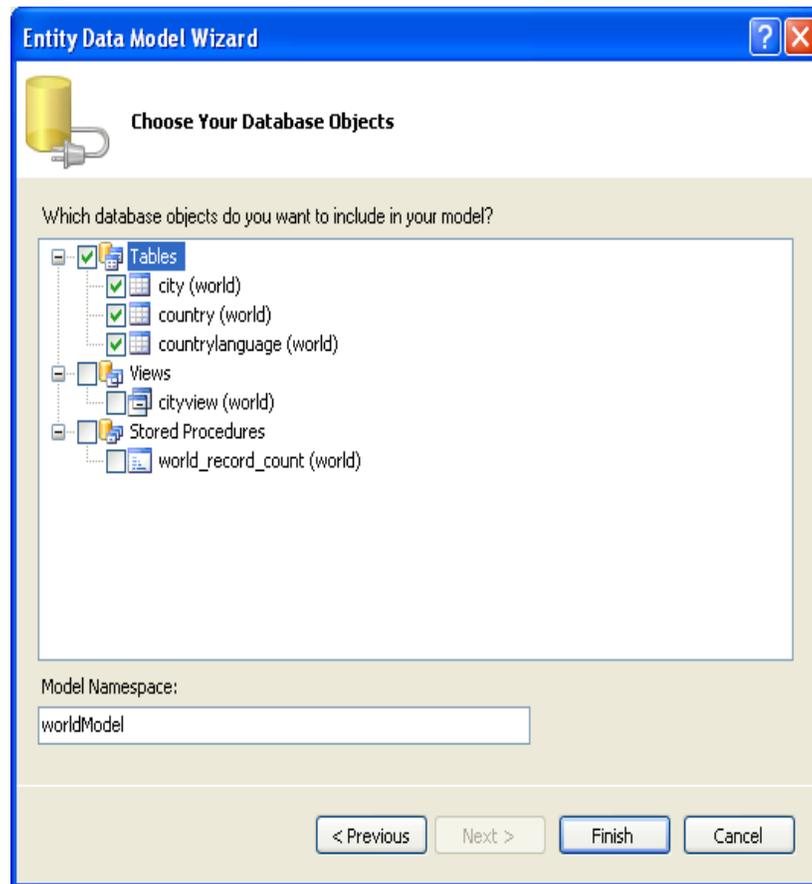


3. You can now select the connection you made earlier to the World database. If you have not already done so, you can create the new connection at this time by clicking **New Connection...**. For further instructions on creating a connection to a database see [Section 22.2.3.1, "Making a Connection"](#).

Figure 22.51. Entity Data Model Wizard Screen 2

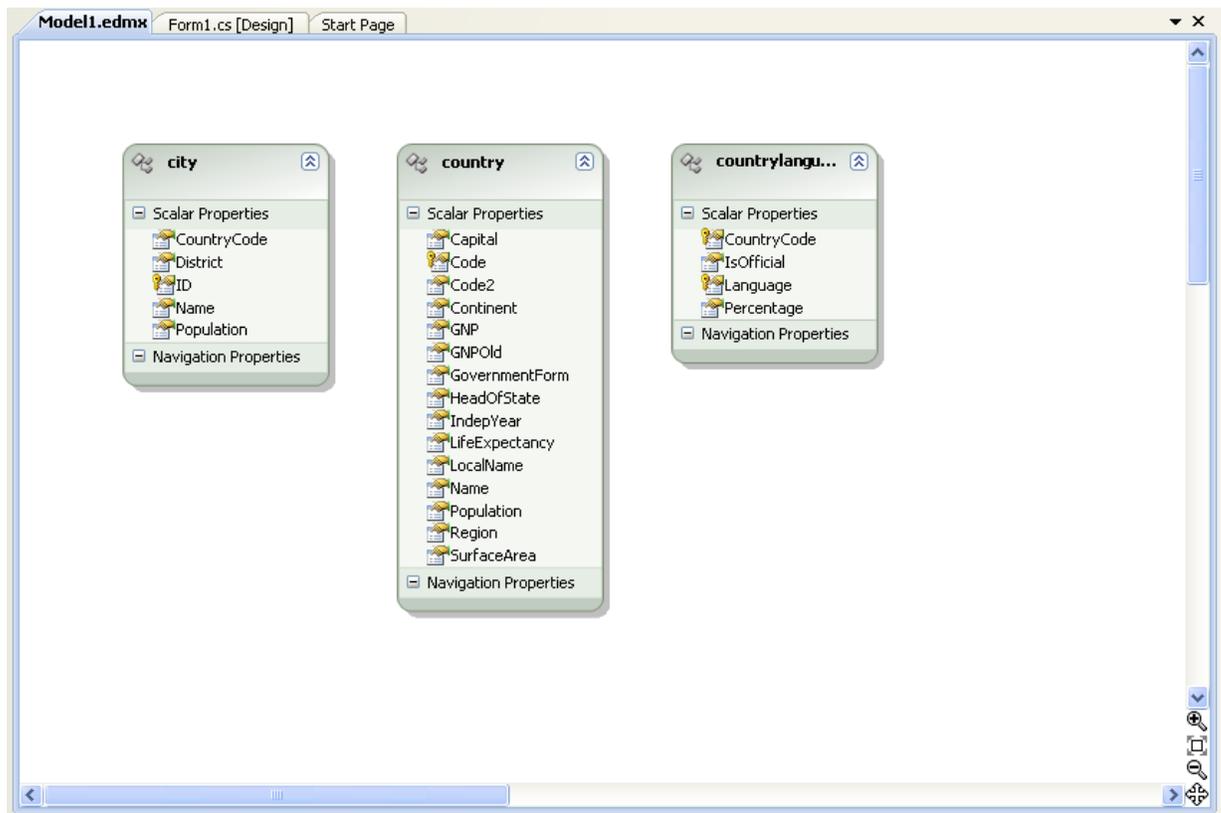


4. Make a note of the entity connection settings to be used in App.Config, as these will be used later to write the necessary control code.
5. Click **Next**.
6. The Entity Data Model Wizard connects to the database. You are then presented with a tree structure of the database. From this you can select the object you would like to include in your model. If you had created Views and Stored Routines these will be displayed along with any tables. In this example you just need to select the tables. Click **Finish** to create the model and exit the wizard.

Figure 22.52. Entity Data Model Wizard Screen 3

7. Visual Studio will generate the model and then display it.

Figure 22.53. Entity Data Model Diagram

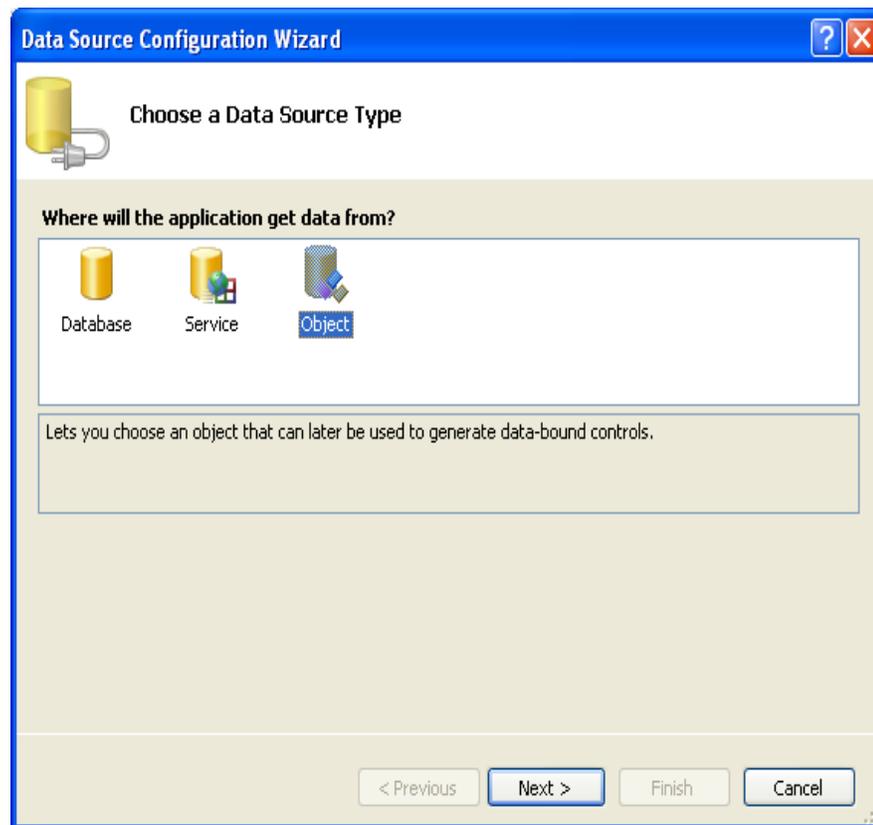


8. From the Visual Studio main menu select **Build, Build Solution**, to ensure that everything compiles correctly so far.

Adding a new Data Source

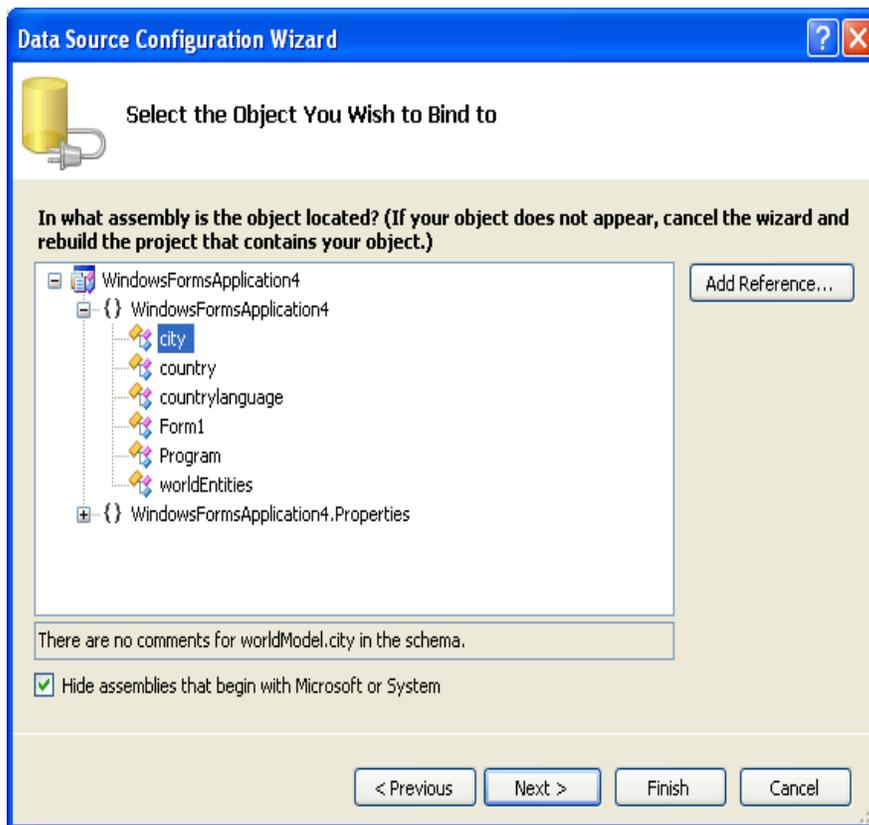
You will now add a new Data Source to your project and see how it can be used to read and write to the database.

1. From the Visual Studio main menu select **Data, Add New Data Source...**. You will be presented with the Data Source Configuration Wizard.

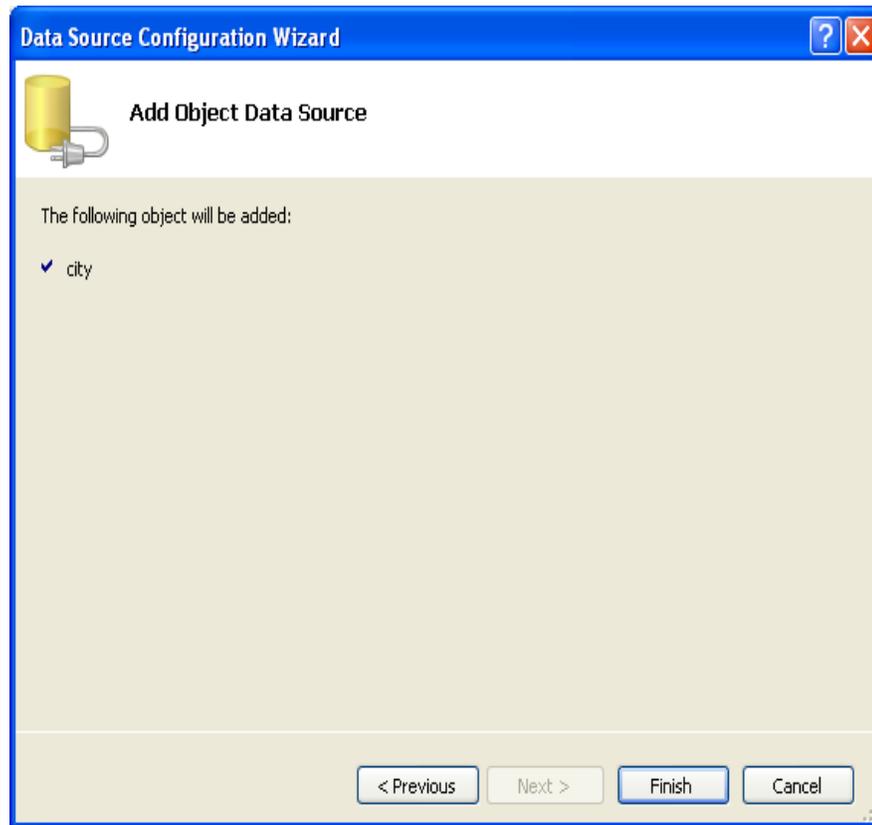
Figure 22.54. Entity Data Source Configuration Wizard Screen 1

2. Select the **Object** icon. Click **Next**.
3. You will now select the Object to bind to. Expand the tree. In this tutorial, you will select the city table. Once the city table has been selected click **Next**.

Figure 22.55. Entity Data Source Configuration Wizard Screen 2



4. The wizard will confirm that the city object is to be added. Click **Finish**.

Figure 22.56. Entity Data Source Configuration Wizard Screen 3

5. The city object will be display in the Data Sources panel. If the Data Sources panel is not displayed, select **Data, Show Data Sources** from the Visual Studio main menu. The docked panel will then be displayed.

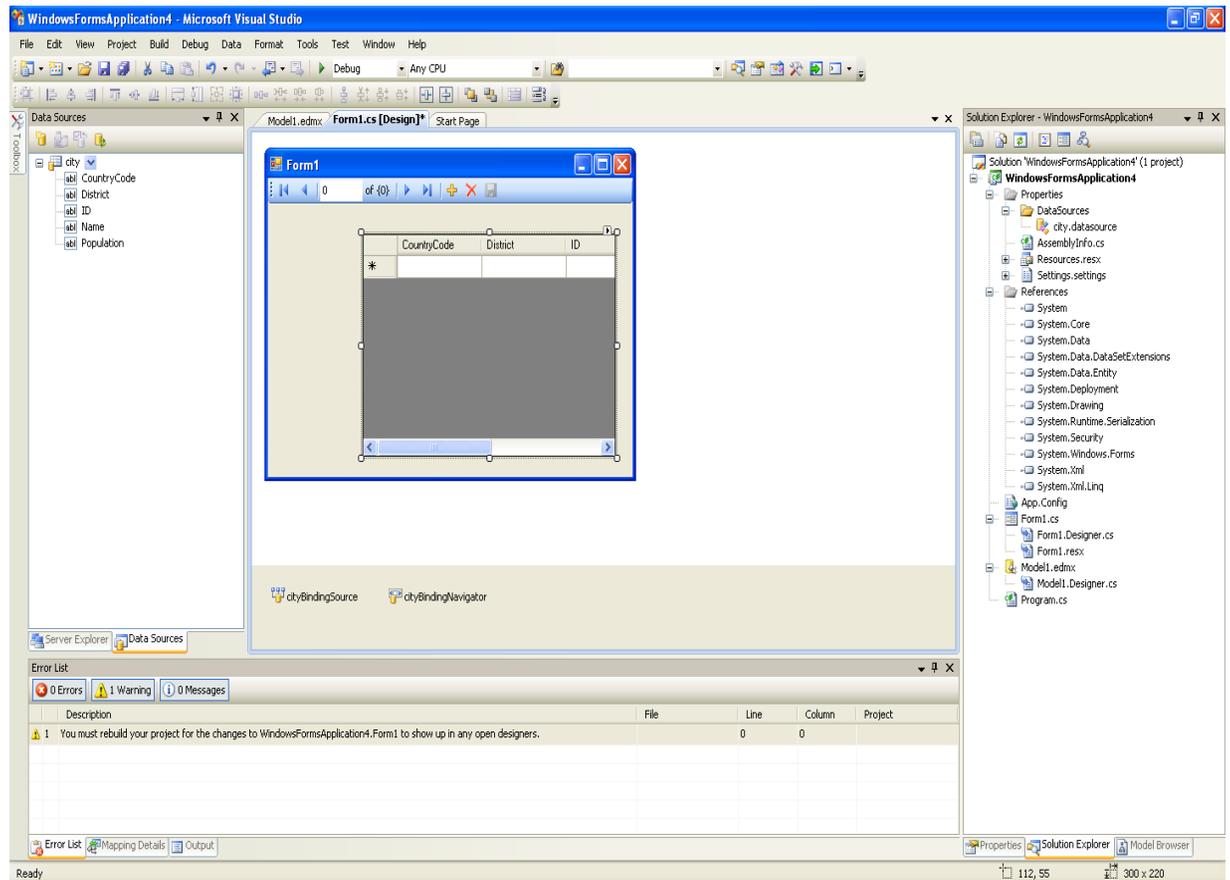
Figure 22.57. Data Sources

Using the Data Source in a Windows Form

You will now learn how to use the Data Source in a Windows Form.

1. In the Data Sources panel select the Data Source you just created and drag and drop it onto the Form Designer. By default the Data Source object will be added as a Data Grid View control. Note that the Data Grid View control is bound to the `cityBindingSource` and the Navigator control is bound to `cityBindingNavigator`.

Figure 22.58. Data Form Designer



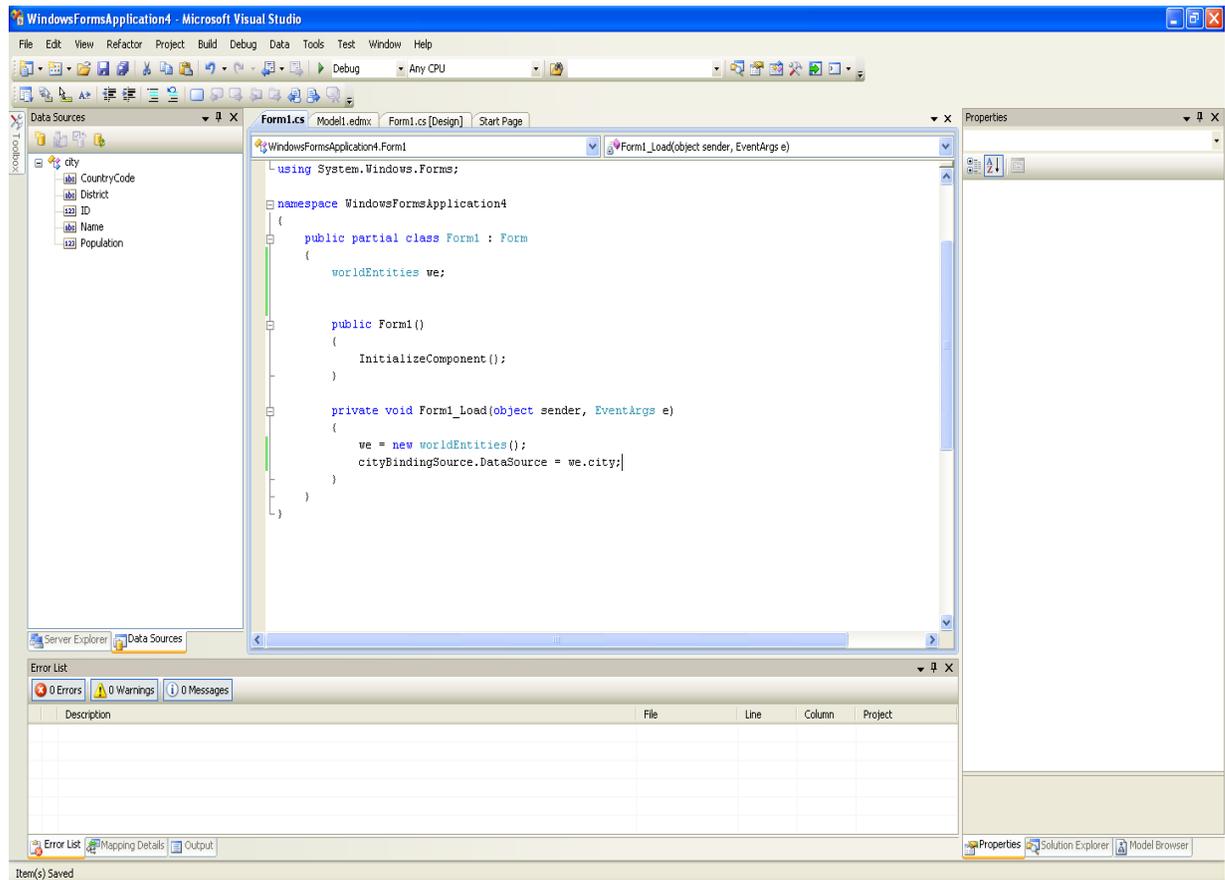
2. Save and rebuild the solution before continuing.

Adding Code to Populate the Data Grid View

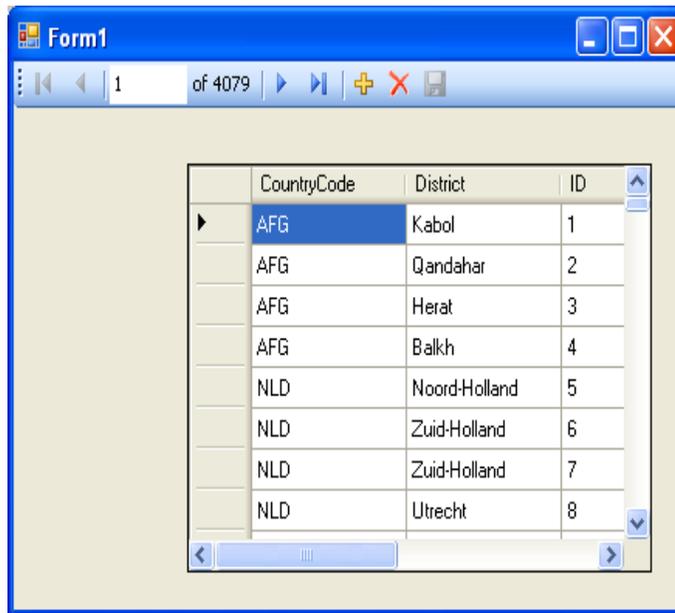
You are now ready to add code to ensure that the Data Grid View control will be populated with data from the City database table.

1. Double-click the form to access its code.
2. Add code to instantiate the Entity Data Model's EntityContainer object and retrieve data from the database to populate the control.

Figure 22.59. Adding Code to the Form



3. Save and rebuild the solution.
4. Run the solution. Ensure the grid is populated and you can navigate the database.

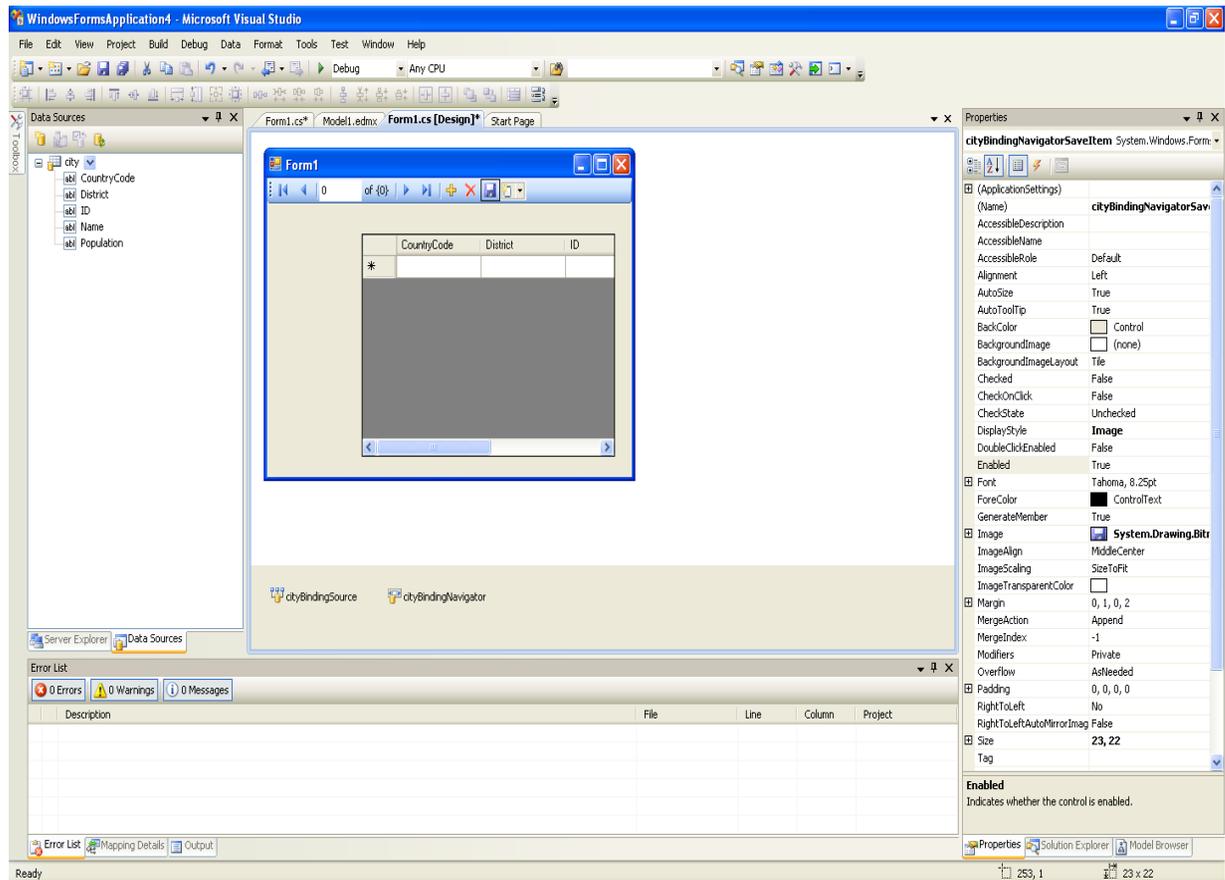
Figure 22.60. The Populated Grid Control**Adding Code to Save Changes to the Database**

You will now add code to enable you to save changes to the database.

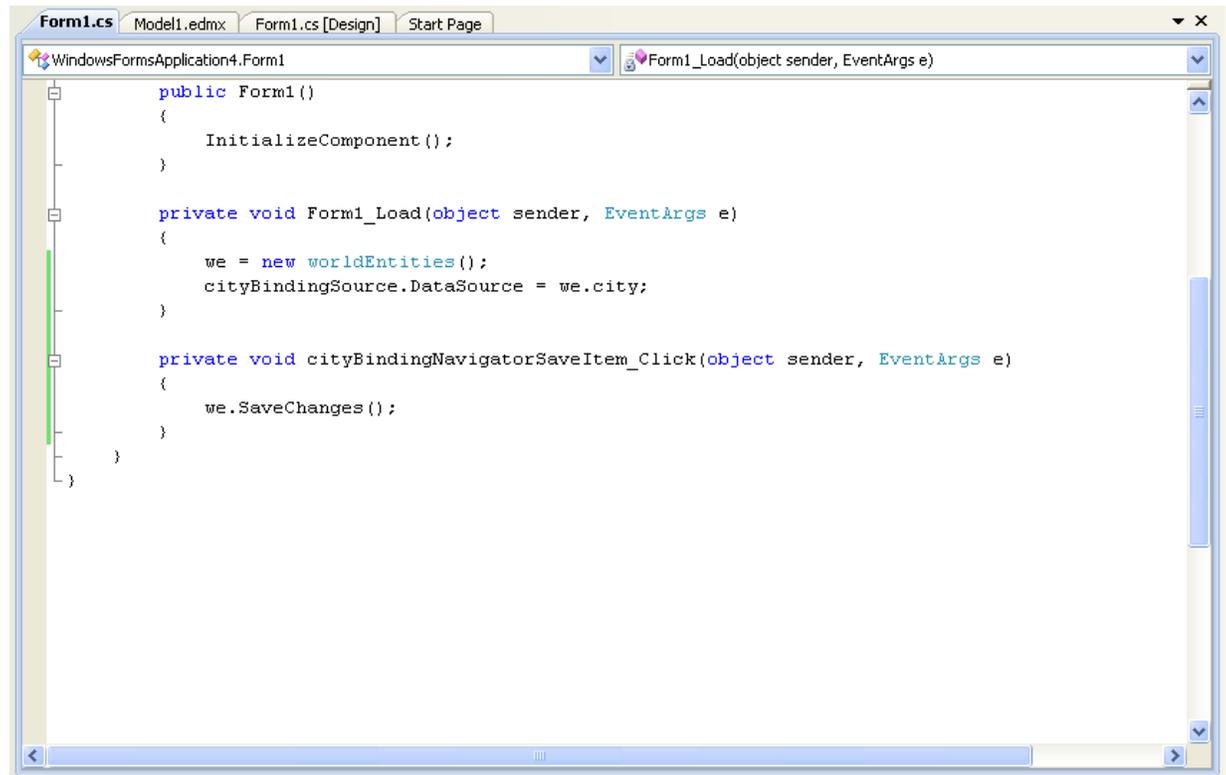
The Binding source component ensures that changes made in the Data Grid View control are also made to the Entity classes bound to it. However, that data needs to be saved back from the entities to the database itself. This can be achieved by the enabling of the Save button in the Navigator control, and the addition of some code.

1. In the Form Designer, click the Save icon in the Form toolbar and ensure that its Enabled property is set to True.

Figure 22.61. Save Button Enabled



2. Double-click the Save icon in the Form toolbar to display its code.
3. You now need to add code to ensure that data is saved to the database when the save button is clicked in the application.

Figure 22.62. Adding Save Code to the Form

4. Once the code has been added, save the solution and rebuild it. Run the application and verify that changes made in the grid are saved.

22.2.4.6. Tutorial: Databinding in ASP.NET using LINQ on Entities

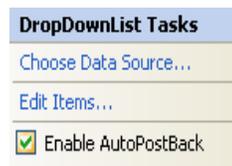
In this tutorial you create an ASP.NET web page that binds LINQ queries to entities using the Entity Framework mapping.

If you have not already done so, install the World example database prior to attempting this tutorial. See the tutorial [Section 22.2.4.5, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source"](#) for instructions on downloading and installing this database.

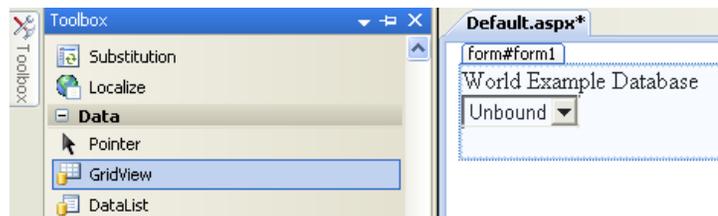
Creating an ASP.NET web site

In this part of the tutorial, you create an ASP.NET web site. The web site uses the World database. The main web page features a drop down list from which you can select a country. Data about that country's cities is then displayed in a grid view control.

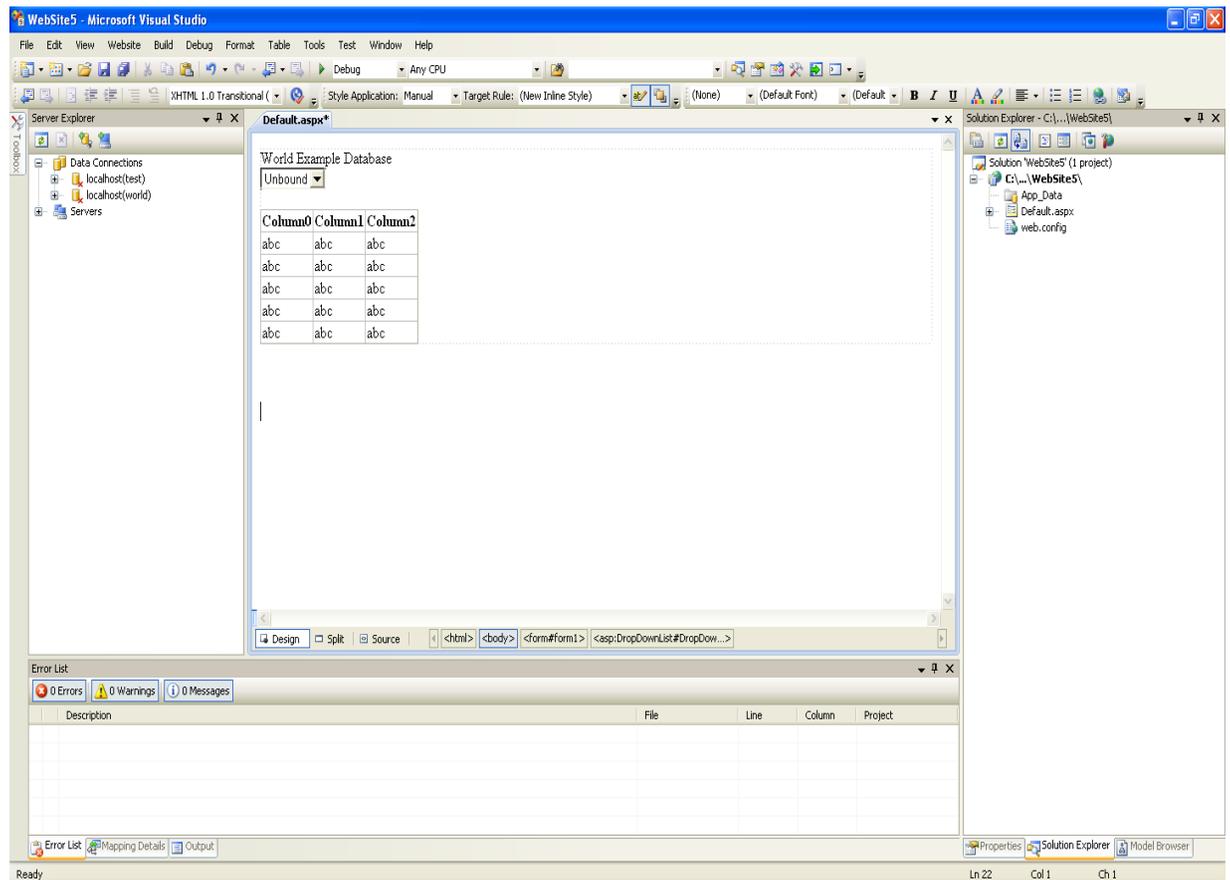
1. From the Visual Studio main menu select **File, New, Web Site...**
2. From the Visual Studio installed templates select **ASP.NET Web Site**. Click OK. You will be presented with the Source view of your web page by default.
3. Click the Design view tab situated underneath the Source view panel.

Figure 22.65. Enable AutoPostBack

- From the Toolbox select the **GridView** control.

Figure 22.66. Grid View Control

Drag and drop the Grid View control to a location just below the Drop Down List you already placed.

Figure 22.67. Placed Grid View Control

- At this point it is recommended that you save your solution, and build the solution to ensure that there are no errors.

9. If you run the solution you will see that the text and drop down list are displayed, but the list is empty. Also, the grid view does not appear at all. Adding this functionality is described in the following sections.

At this stage you have a web site that will build, but further functionality is required. The next step will be to use the Entity Framework to create a mapping from the World database into entities that you can control programmatically.

Creating an ADO.NET Entity Data Model

In this stage of the tutorial you will add an ADO.NET Entity Data Model to your project, using the World database at the storage level. The procedure for doing this is described in the tutorial [Section 22.2.4.5, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source"](#), and so will not be repeated here.

Populating a Drop Data List Box with using the results of a entity LINQ query

In this part of the tutorial you will write code to populate the DropDownList control. When the web page loads the data to populate the list will be achieved by using the results of a LINQ query on the model created previously.

1. In the Design view panel, double-click any blank area. This brings up the `Page_Load` method.
2. Modify the relevant section of code according to the following listing:

```
...
public partial class _Default : System.Web.UI.Page
{
    worldModel.worldEntities we;

    protected void Page_Load(object sender, EventArgs e)
    {
        we = new worldModel.worldEntities();

        if (!IsPostBack)
        {
            var countryQuery = from c in we.country
                               orderby c.Name
                               select new { c.Code, c.Name };
            DropDownList1.DataValueField = "Code";
            DropDownList1.DataTextField = "Name";
            DropDownList1.DataSource = countryQuery;
            DataBind();
        }
    }
}
...
```

Note that the list control only needs to be populated when the page first loads. The conditional code ensures that if the page is subsequently reloaded, the list control is not repopulated, which would cause the user selection to be lost.

3. Save the solution, build it and run it. You should see the list control has been populated. You can select an item, but as yet the grid view control does not appear.

At this point you have a working Drop Down List control, populated by a LINQ query on your entity data model.

Populating a Grid View control using an entity LINQ query

In the last part of this tutorial you will populate the Grid View Control using a LINQ query on your entity data model.

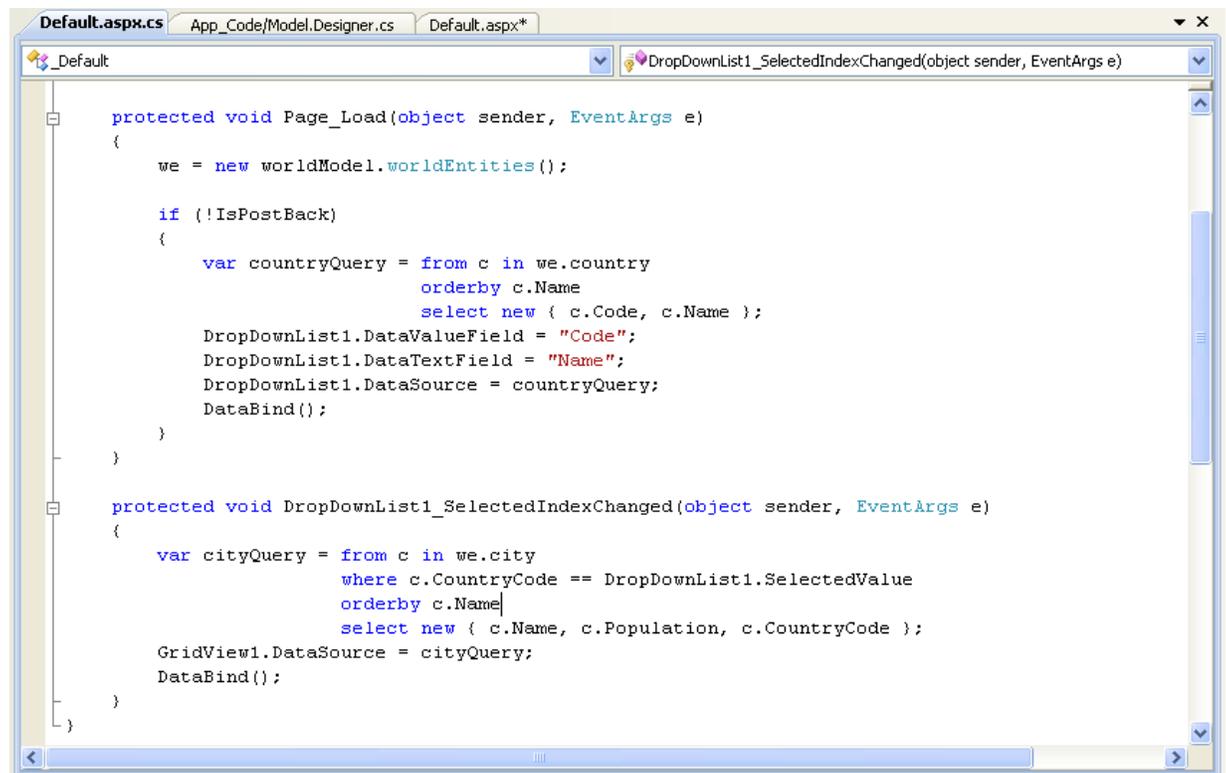
1. In the Design view, double-click the **DropDownList** control. This causes its `SelectedIndexChanged` code to be displayed. This method is called when a user selects an item in the list control and thus fires an `AutoPostBack` event.
2. Modify the relevant section of code accordingly to the following listing:

```
...
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    var cityQuery = from c in we.city
                    where c.CountryCode == DropDownList1.SelectedValue
                    orderby c.Name
                    select new { c.Name, c.Population, c.CountryCode };
    GridView1.DataSource = cityQuery;
    DataBind();
}
...
```

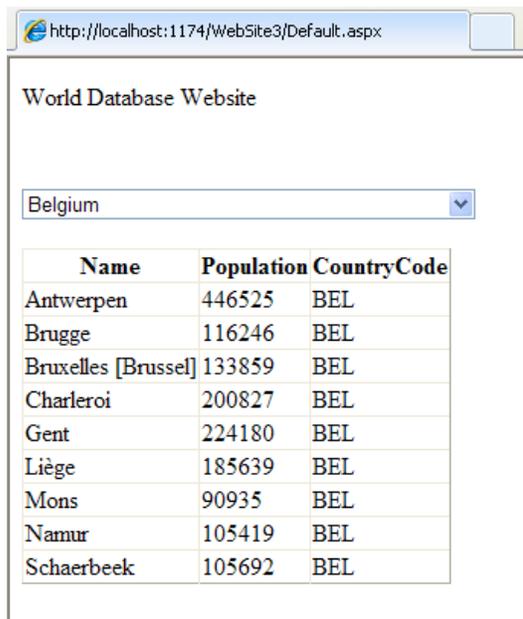
The grid view control is populated from the result of the LINQ query on the entity data model.

3. As a check compare your code to that shown in the following screenshot:

Figure 22.68. Source Code



4. Save, build and run the solution. As you select a country you will see its cities are displayed in the grid view control.

Figure 22.69. The Working Web Site

In this tutorial you have seen how to create an ASP.NET web site, you have also seen how you can access a MySQL database using LINQ queries on an entity data model.

22.2.4.7. Tutorial: Using SSL with MySQL Connector/Net

In this tutorial you will learn how you can use MySQL Connector/Net to connect to a MySQL server configured to use SSL. Support for SSL client certificates was added with MySQL Connector/Net 6.2.

MySQL Server uses the PEM format for certificates and private keys. This tutorial will use the test certificates from the server test suite by way of example. You can obtain the MySQL Server source code from [MySQL Downloads](#). The certificates can be found in the directory `./mysql-test/std_data`.

To carry out the steps in this tutorial, you must have Open SSL installed. This can be downloaded for Microsoft Windows at no charge from [Shining Light Productions](#).

Further details on the connection string options used in this tutorial can be found at [Section 22.2.6, "Connector/Net Connection String Options Reference"](#).

Configuring the MySQL Server to use SSL

1. In the MySQL Server configuration file, set the SSL parameters as follows:

```
ssl-ca=path/to/repo/mysql-test/std_data/cacert.pem
ssl-cert=path/to/repo/mysql-test/std_data/server-cert.pem
ssl-key=path/to/repo/mysql-test/std_data/server-key.pem
```

Adjust the directories according to the location in which you installed the MySQL source code.

2. In this step you create a test user and set the user to require SSL.

Using the MySQL Command Line Client, connect as root and create the user `sslclient`.

3. To set privileges and requirements, issue the following command:

```
GRANT ALL PRIVILEGES ON *.* TO sslclient@%' REQUIRE SSL;
```

Creating a certificate file to use with the .NET client

1. The .NET client does not use the PEM file format, as .NET does not support this format natively. You will be using test client certificates from the same server repository, for the purposes of this example. Convert these to PFX format first. This format is also known as PKCS#12. An article describing this procedure can be found at the [Citrix website](#). From the directory `server-repository-root/mysql-test/std_data`, issue the following command:

```
openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem -certfile cacert.pem -out client.pfx
```

2. When asked for an export password, enter the password "pass". The file `client.pfx` will be generated. This file is used in the remainder of the tutorial.

Connecting to the server using a file-based certificate

1. You will use PFX file, `client.pfx` you created in the previous step to authenticate the client. The following example demonstrates how to connect using the `SSL Mode`, `CertificateFile` and `CertificatePassword` connection string options:

```
using (MySQLConnection connection = new MySQLConnection(
    "database=test;user=sslclient;" +
    "CertificateFile=H:\bzs\mysql-trunk\mysql-test\std_data\client.pfx" +
    "CertificatePassword=pass;" +
    "SSL Mode=Required "))
{
    connection.Open();
}
```

The path to the certificate file will need to be changed to reflect your individual installation.

Connecting to the server using a store-based certificate

1. The first step is to import the PFX file, `client.pfx`, into the Personal Store. Double-click the file in Windows explorer. This launches the Certificate Import Wizard.
2. Follow the steps dictated by the wizard, and when prompted for the password for the PFX file, enter "pass".
3. Click **Finish** to close the wizard and import the certificate into the personal store.

Examine certificates in the Personal Store

1. Start the Microsoft Management Console by entering `mmc.exe` at a command prompt.
2. Select **File**, **Add/Remove snap-in**. Click **Add**. Select **Certificates** from the list of available snap-ins in the dialog.
3. Click **Add** button in the dialog, and select the **My user account** radio button. This is used for personal certificates.
4. Click the **Finish** button.
5. Click **OK** to close the Add/Remove Snap-in dialog.
6. You will now have **Certificates – Current User** displayed in the left panel of the Microsoft Management Console. Expand the Certificates - Current User tree item and select **Personal, Certificates**. The right-hand panel will display a certificate issued to MySQL. This is the certificate that was previously imported. Double-click the certificate to display its details.

7. After you have imported the certificate to the Personal Store, you can use a more succinct connection string to connect to the database, as illustrated by the following code:

```
using (MySQLConnection connection = new MySQLConnection(
    "database=test;user=sslclient;" +
    "Certificate Store Location=CurrentUser;" +
    "SSL Mode=Required"))
{
    connection.Open();
}
```

Certificate Thumbprint Parameter

If you have a large number of certificates in your store, and many have the same Issuer, this can be a source of confusion and result in the wrong certificate being used. To alleviate this situation, there is an optional Certificate Thumbprint parameter that can additionally be specified as part of the connection string. As mentioned before, you can double-click a certificate in the Microsoft Management Console to display the certificate's details. When the Certificate dialog is displayed click the **Details** tab and scroll down to see the thumbprint. The thumbprint will typically be a number such as `#47 94 36 00 9a 40 f3 01 7a 14 5c f8 47 9e 76 94 d7 aa de f0`. This thumbprint can be used in the connection string, as the following code illustrates:

```
using (MySQLConnection connection = new MySQLConnection(
    "database=test;user=sslclient;" +
    "Certificate Store Location=CurrentUser;" +
    "Certificate Thumbprint=479436009a40f3017a145cf8479e7694d7aadef0;" +
    "SSL Mode=Required"))
{
    connection.Open();
}
```

Spaces in the thumbprint parameter are optional and the value is case-insensitive.

22.2.4.8. Tutorial: Using MySqlCommand

This tutorial teaches you how to use the `MySqlCommand` class. This class enables you to execute a series of statements. Depending on the circumstances, this can be more convenient than using the `MySQLCommand` approach.

Further details of the `MySqlCommand` class can be found in the reference documentation supplied with MySQL Connector/Net.

To run the example programs in this tutorial, set up a simple test database and table using the `mysql` Command Line Client or MySQL Workbench. Commands for the `mysql` Command Line Client are given here:

```
CREATE DATABASE TestDB;
USE TestDB;
CREATE TABLE TestTable (id INT NOT NULL PRIMARY KEY
    AUTO_INCREMENT, name VARCHAR(100));
```

The main method of the `MySqlCommand` class is the `Execute` method. This method causes the script (sequence of statements) assigned to the Query property of the `MySqlCommand` object to be executed. Note the Query property can be set through the `MySqlCommand` constructor or using the Query property. `Execute` returns the number of statements executed.

The `MySqlCommand` object will execute the specified script on the connection set using the Connection property. Again, this property can be set directly or through the `MySqlCommand` constructor. The following code snippets illustrate this:

```

string sql = "SELECT * FROM TestTable";
...
MySQLScript script = new MySQLScript(conn, sql);
...
MySQLScript script = new MySQLScript();
script.Query = sql;
script.Connection = conn;
...
script.Execute();

```

The MySQLScript class has several events associated with it. There are:

1. Error - generated if an error occurs.
2. ScriptCompleted - generated when the script successfully completes execution.
3. StatementExecuted - generated after each statement is executed.

It is possible to assign event handlers to each of these events. These user-provided routines are called back when the connected event occurs. The following code shows how the event handlers are set up.

```

script.Error += new MySQLScriptErrorHandler(script_Error);
script.ScriptCompleted += new EventHandler(script_ScriptCompleted);
script.StatementExecuted += new MySQLStatementExecutedEventHandler(script_StatementExecuted);

```

In VisualStudio, you can save typing by using tab completion to fill out stub routines. Start by typing, for example, "script.Error +=". Then press **TAB**, and then press **TAB** again. The assignment is completed, and a stub event handler created. A complete working example is shown below:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace MySQLScriptTest
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=TestDB;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                string sql = "INSERT INTO TestTable(name) VALUES ('Superman');" +
                    "INSERT INTO TestTable(name) VALUES ('Batman');" +
                    "INSERT INTO TestTable(name) VALUES ('Wolverine');" +
                    "INSERT INTO TestTable(name) VALUES ('Storm');"

                MySQLScript script = new MySQLScript(conn, sql);

                script.Error += new MySQLScriptErrorHandler(script_Error);
                script.ScriptCompleted += new EventHandler(script_ScriptCompleted);
                script.StatementExecuted += new MySQLStatementExecutedEventHandler(script_StatementExecuted);

                int count = script.Execute();
            }
            catch { }
        }
    }
}

```

```

        Console.WriteLine("Executed " + count + " statement(s).");
        Console.WriteLine("Delimiter: " + script.Delimiter);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }

    conn.Close();
    Console.WriteLine("Done.");
}

static void script_StatementExecuted(object sender, MySqlScriptEventArgs args)
{
    Console.WriteLine("script_StatementExecuted");
}

static void script_ScriptCompleted(object sender, EventArgs e)
{
    /// EventArgs e will be EventArgs.Empty for this method
    Console.WriteLine("script_ScriptCompleted!");
}

static void script_Error(Object sender, MySqlScriptErrorEventArgs args)
{
    Console.WriteLine("script_Error: " + args.Exception.ToString());
}
}
}

```

Note that in the `script_ScriptCompleted` event handler, the `EventArgs` parameter `e` will be `EventArgs.Empty`. In the case of the `ScriptCompleted` event there is no additional data to be obtained, which is why the event object is `EventArgs.Empty`.

22.2.4.8.1. Using Delimiters with MySqlScript

Depending on the nature of the script, you may need control of the delimiter used to separate the statements that will make up a script. The most common example of this is where you have a multi-statement stored routine as part of your script. In this case if the default delimiter of “;” is used you will get an error when you attempt to execute the script. For example, consider the following stored routine:

```

CREATE PROCEDURE test_routine()
BEGIN
    SELECT name FROM TestTable ORDER BY name;
    SELECT COUNT(name) FROM TestTable;
END

```

This routine actually needs to be executed on the MySQL Server as a single statement. However, with the default delimiter of “;”, the `MySqlScript` class would interpret the above as two statements, the first being:

```

CREATE PROCEDURE test_routine()
BEGIN
    SELECT name FROM TestTable ORDER BY name;

```

Executing this as a statement would generate an error. To solve this problem `MySqlScript` supports the ability to set a different delimiter. This is achieved through the `Delimiter` property. For example, you could set the delimiter to “??”, in which case the above stored routine would no longer generate an error when executed. Multiple statements can be delimited in the script, so for example, you could have a three statement script such as:

```

string sql = "DROP PROCEDURE IF EXISTS test_routine?? " +
            "CREATE PROCEDURE test_routine() " +

```

```
"BEGIN " +
"SELECT name FROM TestTable ORDER BY name;" +
"SELECT COUNT(name) FROM TestTable;" +
"END??" +
"CALL test_routine()";
```

You can change the delimiter back at any point by setting the Delimiter property. The following code shows a complete working example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MySql.Data;
using MySql.Data.MySqlClient;

namespace ConsoleApplication8
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=TestDB;port=3306;password=*****;";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                string sql = "DROP PROCEDURE IF EXISTS test_routine??" +
                    "CREATE PROCEDURE test_routine() " +
                    "BEGIN " +
                    "SELECT name FROM TestTable ORDER BY name;" +
                    "SELECT COUNT(name) FROM TestTable;" +
                    "END??" +
                    "CALL test_routine()";

                MySqlScript script = new MySqlScript(conn);

                script.Query = sql;
                script.Delimiter = "??";
                int count = script.Execute();
                Console.WriteLine("Executed " + count + " statement(s)");
                script.Delimiter = ";";
                Console.WriteLine("Delimiter: " + script.Delimiter);
                Console.WriteLine("Query: " + script.Query);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }

            conn.Close();
            Console.WriteLine("Done.");
        }
    }
}
```

22.2.4.9. Tutorial: Generating MySQL DDL from an Entity Framework Model

In this tutorial, you will learn how to create MySQL [DDL](#) from an Entity Framework model. Use Visual Studio 2010 and MySQL Connector/Net 6.3 to carry out this tutorial.

1. Create a new console application in Visual Studio 2010.

2. Using the **Solution Explorer**, add a reference to `MySQL.Data.Entity`.
3. From the **Solution Explorer** select **Add, New Item**. In the **Add New Item** dialog select **Online Templates**. Select **ADO.NET Entity Data Model** and click **Add**. The **Entity Data Model** dialog will be displayed.
4. In the **Entity Data Model** dialog select **Empty Model**. Click **Finish**. A blank model will be created.
5. Create a simple model. A single Entity will do for the purposes of this tutorial.
6. In the **Properties** panel select `ConceptualEntityModel` from the drop-down listbox.
7. In the **Properties** panel, locate the **DDL Generation Template** in the category **Database Script Generation**.
8. For the **DDL Generation** property select `SSDLToMySQL.tt(VS)` from the drop-down listbox.
9. Save the solution.
10. Right-click an empty space in the model design area. The context-sensitive menu will be displayed.
11. From the context-sensitive menu select **Generate Database from Model**. The **Generate Database Wizard** dialog will be displayed.
12. In the **Generate Database Wizard** dialog select an existing connection, or create a new connection to a server. Select an appropriate radio button to show or hide sensitive data. For the purposes of this tutorial you can select **Yes** (although you might skip this for commercial applications).
13. Click **Next**. MySQL compatible DDL code will be generated. Click **Finish** to exit the wizard.

You have seen how to create MySQL DDL code from an Entity Framework model.

22.2.5. Connector/Net Programming

Connector/Net comprises several classes that are used to connect to the database, execute queries and statements, and manage query results.

The following are the major classes of Connector/Net:

- `MySQLCommand`: Represents an SQL statement to execute against a MySQL database.
- `MySQLCommandBuilder`: Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated MySQL database.
- `MySQLConnection`: Represents an open connection to a MySQL Server database.
- `MySQLDataAdapter`: Represents a set of data commands and a database connection that are used to fill a data set and update a MySQL database.
- `MySQLDataReader`: Provides a means of reading a forward-only stream of rows from a MySQL database.
- `MySQLException`: The exception that is thrown when MySQL returns an error.
- `MySQLHelper`: Helper class that makes it easier to work with the provider.
- `MySQLTransaction`: Represents an SQL **transaction** to be made in a MySQL database.

In the following sections, you will learn about some common use cases for Connector/Net, including BLOB handling, date handling, and using Connector/Net with common tools such as Crystal Reports.

22.2.5.1. Connecting to MySQL Using Connector/Net

All interaction between a .NET application and the MySQL server is routed through a `MySqlConnection` object. Before your application can interact with the server, it must instantiate, configure, and open a `MySqlConnection` object.

Even when using the `MySqlHelper` class, a `MySqlConnection` object is created by the helper class.

This section describes how to connect to MySQL using the `MySqlConnection` object.

22.2.5.2. Creating a Connector/Net Connection String

The `MySqlConnection` object is configured using a connection string. A connection string contains several key/value pairs, separated by semicolons. In each key/value pair, the option name and its corresponding value are joined by an equal sign. For the list of option names to use in the connection string, see [Section 22.2.6, “Connector/Net Connection String Options Reference”](#).

The following is a sample connection string:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

In this example, the `MySqlConnection` object is configured to connect to a MySQL server at `127.0.0.1`, with a user name of `root` and a password of `12345`. The default database for all statements will be the `test` database.



Note

Using the '@' symbol for parameters is now the preferred approach, although the old pattern of using '?' is still supported. To avoid conflicts when using the '@' symbol in combination with user variables, see the [Allow User Variables](#) connection string option in [Section 22.2.6, “Connector/Net Connection String Options Reference”](#). The [Old Syntax](#) connection string option has now been deprecated.

22.2.5.2.1. Opening a Connection

Once you have created a connection string it can be used to open a connection to the MySQL server.

The following code is used to create a `MySqlConnection` object, assign the connection string, and open the connection.

Connector/Net can also connect using the native Windows authentication plugin. See [Section 22.2.5.5, “Using the Windows Native Authentication Plugin”](#) for details.

You can further extend the authentication mechanism by writing your own authentication plugin. See [Section 22.2.5.6, “Writing a Custom Authentication Plugin”](#) for details.

Visual Basic Example

```
Dim conn As New MySql.Data.MySqlClient.MySqlConnection
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
```

```
        & "uid=root;" _  
        & "pwd=12345;" _  
        & "database=test;"  
  
Try  
    conn.ConnectionString = myConnectionString  
    conn.Open()  
  
Catch ex As MySql.Data.MySqlClient.MySqlException  
    MessageBox.Show(ex.Message)  
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;  
string myConnectionString;  
  
myConnectionString = "server=127.0.0.1;uid=root;" +  
    "pwd=12345;database=test;"  
  
try  
{  
    conn = new MySql.Data.MySqlClient.MySqlConnection();  
    conn.ConnectionString = myConnectionString;  
    conn.Open();  
}  
catch (MySql.Data.MySqlClient.MySqlException ex)  
{  
    MessageBox.Show(ex.Message);  
}
```

You can also pass the connection string to the constructor of the [MySqlConnection](#) class:

Visual Basic Example

```
Dim myConnectionString as String  
  
myConnectionString = "server=127.0.0.1;" _  
    & "uid=root;" _  
    & "pwd=12345;" _  
    & "database=test;"  
  
Try  
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)  
    conn.Open()  
Catch ex As MySql.Data.MySqlClient.MySqlException  
    MessageBox.Show(ex.Message)  
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;  
string myConnectionString;  
  
myConnectionString = "server=127.0.0.1;uid=root;" +  
    "pwd=12345;database=test;"  
  
try  
{  
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);  
    conn.Open();  
}  
catch (MySql.Data.MySqlClient.MySqlException ex)  
{  
    MessageBox.Show(ex.Message);  
}
```

Once the connection is open it can be used by the other Connector/Net classes to communicate with the MySQL server.

22.2.5.2.2. Handling Connection Errors

Because connecting to an external server is unpredictable, it is important to add error handling to your .NET application. When there is an error connecting, the `MySqlConnection` class will return a `MySqlException` object. This object has two properties that are of interest when handling errors:

- `Message`: A message that describes the current exception.
- `Number`: The MySQL error number.

When handling errors, you can your application's response based on the error number. The two most common error numbers when connecting are as follows:

- `0`: Cannot connect to server.
- `1045`: Invalid user name and/or password.

The following code shows how to adapt the application's response based on the actual error:

Visual Basic Example

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    Select Case ex.Number
        Case 0
            MessageBox.Show("Cannot connect to server. Contact administrator")
        Case 1045
            MessageBox.Show("Invalid username/password, please try again")
    End Select
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    switch (ex.Number)
    {
        case 0:
            MessageBox.Show("Cannot connect to server. Contact administrator");
        case 1045:
```

```
        MessageBox.Show("Invalid username/password, please try again");  
    }  
}
```



Important

Note that if you are using multilanguage databases you must specify the character set in the connection string. If you do not specify the character set, the connection defaults to the `latin1` charset. You can specify the character set as part of the connection string, for example:

```
MySQLConnection myConnection = new MySQLConnection("server=127.0.0.1;uid=root;" +  
    "pwd=12345;database=test;Charset=latin1;");
```

22.2.5.2.3. Using GetSchema on a Connection

The `GetSchema()` method of the connection object can be used to retrieve schema information about the database currently connected to. The schema information is returned in the form of a `DataTable`. The schema information is organized into a number of collections. Different forms of the `GetSchema()` method can be used depending on the information required. There are three forms of the `GetSchema()` method:

- `GetSchema()` - This call will return a list of available collections.
- `GetSchema(String)` - This call returns information about the collection named in the string parameter. If the string "MetaDataCollections" is used then a list of all available collections is returned. This is the same as calling `GetSchema()` without any parameters.
- `GetSchema(String, String[])` - In this call the first string parameter represents the collection name, and the second parameter represents a string array of restriction values. Restriction values limit the amount of data that will be returned. Restriction values are explained in more detail in the [Microsoft .NET documentation](#).

22.2.5.2.3.1. Collections

The collections can be broadly grouped into two types: collections that are common to all data providers, and collections specific to a particular provider.

Common

The following collections are common to all data providers:

- MetaDataCollections
- DataSourceInformation
- DataTypes
- Restrictions
- ReservedWords

Provider-specific

The following are the collections currently provided by MySQL Connector/Net, in addition to the common collections above:

- Databases

- Tables
- Columns
- Users
- Foreign Keys
- IndexColumns
- Indexes
- Foreign Key Columns
- UDF
- Views
- ViewColumns
- Procedure Parameters
- Procedures
- Triggers

Example Code

A list of available collections can be obtained using the following code:

```
using System;
using System.Data;
using System.Text;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace ConsoleApplication2
{
    class Program
    {
        private static void DisplayData(System.Data.DataTable table)
        {
            foreach (System.Data.DataRow row in table.Rows)
            {
                foreach (System.Data.DataColumn col in table.Columns)
                {
                    Console.WriteLine("{0} = {1}", col.ColumnName, row[col]);
                }
                Console.WriteLine("=====");
            }
        }

        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
            }
        }
    }
}
```

```

        DataTable table = conn.GetSchema("MetaDataCollections");
        //DataTable table = conn.GetSchema("UDF");
        DisplayData(table);

        conn.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
    Console.WriteLine("Done.");
}
}
}

```

Further information on the `GetSchema()` method and schema collections can be found in the [Microsoft .NET documentation](#).

22.2.5.3. Using MySqlCommand

A `MySqlCommand` has the `CommandText` and `CommandType` properties associated with it. The `CommandText` will be handled differently depending on the setting of `CommandType`. `CommandType` can be one of:

1. Text - A SQL text command (default)
2. StoredProcedure - The name of a Stored Procedure
3. TableDirect - The name of a table (new in Connector/Net 6.2)

The default `CommandType`, `Text`, is used for executing queries and other SQL commands. Some example of this can be found in the following section [Section 22.2.4.1.2, "The MySqlCommand Object"](#).

If `CommandType` is set to `StoredProcedure`, set `CommandText` to the name of the Stored Procedure to access.

If `CommandType` is set to `TableDirect`, all rows and columns of the named table will be returned when you call one of the Execute methods. In effect, this command performs a `SELECT *` on the table specified. The `CommandText` property is set to the name of the table to query. This is illustrated by the following code snippet:

```

...
MySqlCommand cmd = new MySqlCommand();
cmd.CommandText = "mytable";
cmd.Connection = someConnection;
cmd.CommandType = CommandType.TableDirect;
MySqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    Console.WriteLine(reader[0], reader[1]...);
}
...

```

Examples of using the `CommandType` of `StoredProcedure` can be found in the section [Section 22.2.5.9, "Accessing Stored Procedures with Connector/Net"](#).

Commands can have a timeout associated with them. This is useful as you may not want a situation where a command takes up an excessive amount of time. A timeout can be set using the `CommandTimeout` property. The following code snippet sets a timeout of one minute:

```

MySqlCommand cmd = new MySqlCommand();

```

```
cmd.CommandTimeout = 60;
```

The default value is 30 seconds. Avoid a value of 0, which indicates an indefinite wait. To change the default command timeout, use the connection string option [Default Command Timeout](#).

Prior to MySQL Connector/Net 6.2, `MySqlCommand.CommandTimeout` included user processing time, that is processing time not related to direct use of the connector. Timeout was implemented through a .NET Timer, that triggered after `CommandTimeout` seconds. This timer consumed a thread.

MySQL Connector/Net 6.2 introduced timeouts that are aligned with how Microsoft handles `SqlCommand.CommandTimeout`. This property is the cumulative timeout for all network reads and writes during command execution or processing of the results. A timeout can still occur in the `MySqlReader.Read` method after the first row is returned, and does not include user processing time, only IO operations. The 6.2 implementation uses the underlying stream timeout facility, so is more efficient in that it does not require the additional timer thread as was the case with the previous implementation.

Further details on this can be found in the relevant [Microsoft documentation](#).

22.2.5.4. Using Connector/Net with Connection Pooling

The Connector/Net supports connection pooling for better performance and scalability with database-intensive applications. This is enabled by default. You can turn it off or adjust its performance characteristics using the connection string options [Pooling](#), [Connection Reset](#), [Connection Lifetime](#), [Cache Server Properties](#), [Max Pool Size](#) and [Min Pool Size](#). See [Section 22.2.5.2, "Creating a Connector/Net Connection String"](#) for further information.

Connection pooling works by keeping the native connection to the server live when the client disposes of a `MySqlConnection`. Subsequently, if a new `MySqlConnection` object is opened, it will be created from the connection pool, rather than creating a new native connection. This improves performance.

Guidelines

To work as designed, it is best to let the connection pooling system manage all connections. Do not create a globally accessible instance of `MySqlConnection` and then manually open and close it. This interferes with the way the pooling works and can lead to unpredictable results or even exceptions.

One approach that simplifies things is to avoid manually creating a `MySqlConnection` object. Instead use the overloaded methods that take a connection string as an argument. Using this approach, Connector/Net will automatically create, open, close and destroy connections, using the connection pooling system for best performance.

Typed Datasets and the `MembershipProvider` and `RoleProvider` classes use this approach. Most classes that have methods that take a `MySqlConnection` as an argument, also have methods that take a connection string as an argument. This includes `MySqlDataAdapter`.

Instead of manually creating `MySqlCommand` objects, you can use the static methods of the `MySqlHelper` class. These take a connection string as an argument, and they fully support connection pooling.

Resource Usage

Starting with MySQL Connector/Net 6.2, there is a background job that runs every three minutes and removes connections from pool that have been idle (unused) for more than three minutes. The pool cleanup frees resources on both client and server side. This is because on the client side every connection uses a socket, and on the server side every connection uses a socket and a thread.

Prior to this change, connections were never removed from the pool, and the pool always contained the peak number of open connections. For example, a web application that peaked at 1000 concurrent database connections would consume 1000 threads and 1000 open sockets at the server, without ever freeing up those resources from the connection pool. Note, connections, no matter how old, will not be closed if the number of connections in the pool is less than or equal to the value set by the `Min Pool Size` connection string parameter.

22.2.5.5. Using the Windows Native Authentication Plugin

Connector/Net applications can authenticate to a MySQL server using the Windows Native Authentication Plugin as of Connector/Net 6.4.4 and MySQL 5.5.16. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. For background and usage information about the authentication plugin, see, [Section 6.3.6.3, “The Windows Native Authentication Plugin”](#).

The interface matches the `MySql.Data.MySqlClient` object. To enable, pass in `Integrated Security` to the connection string with a value of `yes` or `sspi`.

Passing in a user ID is optional. When Windows authentication is set up, a MySQL user is created and configured to be used by Windows authentication. By default, this user ID is named `auth_windows`, but can be defined using a different name. If the default name is used, then passing the user ID to the connection string from Connector/Net is optional, because it will use the `auth_windows` user. Otherwise, the name must be passed to the `connection string` using the standard user ID element.

22.2.5.6. Writing a Custom Authentication Plugin

Advanced users with special security requirements can create their own authentication plugins for Connector/Net applications. You can extend the handshake protocol, adding custom logic. This capability requires Connector/Net 6.6.3 or higher, and MySQL 5.5.16 or higher. For background and usage information about MySQL authentication plugins, see, [Section 23.2.3.7, “Authentication Plugins”](#) and [Section 23.2.4.9, “Writing Authentication Plugins”](#).

To write a custom authentication plugin, you will need a reference to the assembly `MySql.Data.dll`. The classes relevant for writing authentication plugins are available at the namespace `MySql.Data.MySqlClient.Authentication`.

How the Custom Authentication Plugin Works

At some point during handshake, the internal method

```
void Authenticate(bool reset)
```

of `MySqlAuthenticationPlugin` is called. This method in turns calls several overridable methods of the current plugin.

Creating the Authentication Plugin Class

You put the authentication plugin logic inside a new class derived from `MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin`. The following methods are available to be overridden:

```
protected virtual void CheckConstraints()  
protected virtual void AuthenticationFailed(Exception ex)  
protected virtual void AuthenticationSuccessful()  
protected virtual byte[] MoreData(byte[] data)  
protected virtual void AuthenticationChange()
```

```
public abstract string PluginName { get; }
public virtual string GetUsername()
public virtual object GetPassword()
protected byte[] AuthData;
```

The following is a brief explanation of each one:

```
/// <summary>
/// This method must check authentication method specific constraints in the
/// environment and throw an Exception
/// if the conditions are not met. The default implementation does nothing.
/// </summary>
protected virtual void CheckConstraints()

/// <summary>
/// This method, called when the authentication failed, provides a chance to
/// plugins to manage the error
/// the way they consider decide (either showing a message, logging it, etc.).
/// The default implementation wraps the original exception in a MySqlConnection
/// with an standard message and rethrows it.
/// </summary>
/// <param name="ex">The exception with extra information on the error.</param>
protected virtual void AuthenticationFailed(Exception ex)

/// <summary>
/// This method is invoked when the authentication phase was successful accepted
/// by the server.
/// Derived classes must override this if they want to be notified of such
/// condition.
/// </summary>
/// <remarks>The default implementation does nothing.</remarks>
protected virtual void AuthenticationSuccessful()

/// <summary>
/// This method provides a chance for the plugin to send more data when the
/// server requests so during the
/// authentication phase. This method will be called at least once, and more
/// than one depending upon whether the
/// server response packets have the 0x01 prefix.
/// </summary>
/// <param name="data">The response data from the server, during the
/// authentication phase the first time is called is null, in
/// subsequent calls contains the server response.</param>
/// <returns>The data generated by the plugin for server consumption.</returns>
/// <remarks>The default implementation always returns null.</remarks>
protected virtual byte[] MoreData(byte[] data)

/// <summary>
/// The plugin name.
/// </summary>
public abstract string PluginName { get; }

/// <summary>
/// Gets the user name to send to the server in the authentication phase.
/// </summary>
/// <returns>An string with the user name</returns>
/// <remarks>Default implementation returns the UserId passed from the
/// connection string.</remarks>
public virtual string GetUsername()

/// <summary>
/// Gets the password to send to the server in the authentication phase. This
/// can can be an string or a
/// </summary>
/// <returns>An object, can be byte[], string or null, with the password.
</returns>
```

```
/// <remarks>Default implementation returns null.</remarks>
public virtual object GetPassword()

/// <summary>
/// The authentication data passed when creating the plugin.
/// For example in mysql_native_password this is the seed to encrypt the
password.
/// </summary>
protected byte[] AuthData;
```

Sample Authentication Plugin

Here is an example showing how to create the authentication plugin, then enable it by means of a configuration file. Follow these steps:

1. Create a console app, adding a reference to `MySQL.Data.dll`.
2. Design the main program as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MySql.Data.MySqlClient;

namespace AuthPluginTest
{
    class Program
    {
        static void Main(string[] args)
        {
            // Customize the connection string as necessary.
            MySqlConnection con = new MySqlConnection("server=localhost;
database=test; user id=myuser; password=mypass");
            con.Open();
            con.Close();
        }
    }
}
```

3. Create your plugin class. In this example, we add an “alternative” implementation of the Native password plugin by just using the same code from the original plugin. We name our class `MySQLNativePasswordPlugin2`:

```
using System.IO;
using System;
using System.Text;
using System.Security.Cryptography;
using MySql.Data.MySqlClient.Authentication;
using System.Diagnostics;

namespace AuthPluginTest
{
    public class MySQLNativePasswordPlugin2 : MySQLAuthenticationPlugin
    {
        public override string PluginName
        {
            get { return "mysql_native_password"; }
        }

        public override object GetPassword()
        {
            Debug.WriteLine("Calling MySQLNativePasswordPlugin2.GetPassword");
        }
    }
}
```

```

    return Get411Password(Settings.Password, AuthData);
}

/// <summary>
/// Returns a byte array containing the proper encryption of the
/// given password/seed according to the new 4.1.1 authentication scheme.
/// </summary>
/// <param name="password"></param>
/// <param name="seed"></param>
/// <returns></returns>
private byte[] Get411Password(string password, byte[] seedBytes)
{
    // if we have no password, then we just return 1 zero byte
    if (password.Length == 0) return new byte[1];

    SHA1 sha = new SHA1CryptoServiceProvider();

    byte[] firstHash = sha.ComputeHash(Encoding.Default.GetBytes(password));
    byte[] secondHash = sha.ComputeHash(firstHash);

    byte[] input = new byte[seedBytes.Length + secondHash.Length];
    Array.Copy(seedBytes, 0, input, 0, seedBytes.Length);
    Array.Copy(secondHash, 0, input, seedBytes.Length, secondHash.Length);
    byte[] thirdHash = sha.ComputeHash(input);

    byte[] finalHash = new byte[thirdHash.Length + 1];
    finalHash[0] = 0x14;
    Array.Copy(thirdHash, 0, finalHash, 1, thirdHash.Length);

    for (int i = 1; i < finalHash.Length; i++)
        finalHash[i] = (byte)(finalHash[i] ^ firstHash[i - 1]);
    return finalHash;
}
}
}

```

4. Notice that the plugin implementation just overrides `GetPassword`, and provides an implementation to encrypt the password using the 4.1 protocol. We also put the following line in the `GetPassword` body:

```
Debug.WriteLine("Calling MySqlNativePasswordPlugin2.GetPassword");
```

to provide confirmation that the plugin was effectively used. (You could also put a breakpoint on that method.)

5. Enable the new plugin in the configuration file:

```

<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration,
MySql.Data"/>
  </configSections>
  <MySQL>
    <AuthenticationPlugins>
      <add name="mysql_native_password"
type="AuthPluginTest.MySqlNativePasswordPlugin2, AuthPluginTest"></add>
    </AuthenticationPlugins>
  </MySQL>
</startup><supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
</configuration>

```

6. Run the application. In Visual Studio, you will see the message `Calling MySqlNativePasswordPlugin2.GetPassword` in the debug window.
7. Continue enhancing the authentication logic, overriding more methods if you required.

22.2.5.7. Using Connector/Net with Table Caching

This feature exists with Connector/Net versions 6.4 and above.

Table caching is a feature that can be used to cache slow-changing datasets on the client side. This is useful for applications that are designed to use readers, but still want to minimize trips to the server for slow-changing tables.

This feature is transparent to the application, and is disabled by default.

Configuration

- To enable table caching, add `'table cache = true'` to the connection string.
- Optionally, specify the `'Default Table Cache Age'` connection string option, which represents the number of seconds a table is cached before the cached data is discarded. The default value is `60`.
- You can turn caching on and off and set caching options at runtime, on a per-command basis.

22.2.5.8. Using the Connector/Net with Prepared Statements

As of MySQL 4.1, it is possible to use prepared statements with Connector/Net. Use of prepared statements can provide significant performance improvements on queries that are executed more than once.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

22.2.5.8.1. Preparing Statements in Connector/Net

To prepare a statement, create a command object and set the `.CommandText` property to your query.

After entering your statement, call the `.Prepare` method of the `MySQLCommand` object. After the statement is prepared, add parameters for each of the dynamic elements in the query.

After you enter your query and enter parameters, execute the statement using the `.ExecuteNonQuery()`, `.ExecuteScalar()`, or `.ExecuteReader` methods.

For subsequent executions, you need only modify the values of the parameters and call the execute method again, there is no need to set the `.CommandText` property or redefine the parameters.

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = strConnection

Try
    conn.Open()
    cmd.Connection = conn
```

```

cmd.CommandText = "INSERT INTO myTable VALUES(NULL, @number, @text)"
cmd.Prepare()

cmd.Parameters.AddWithValue("@number", 1)
cmd.Parameters.AddWithValue("@text", "One")

For i = 1 To 1000
    cmd.Parameters("@number").Value = i
    cmd.Parameters("@text").Value = "A string value"

    cmd.ExecuteNonQuery()
Next
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = strConnection;

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, @number, @text)";
    cmd.Prepare();

    cmd.Parameters.AddWithValue("@number", 1);
    cmd.Parameters.AddWithValue("@text", "One");

    for (int i=1; i <= 1000; i++)
    {
        cmd.Parameters["@number"].Value = i;
        cmd.Parameters["@text"].Value = "A string value";

        cmd.ExecuteNonQuery();
    }
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

22.2.5.9. Accessing Stored Procedures with Connector/Net

MySQL server version 5 and up supports stored procedures with the SQL 2003 stored procedure syntax.

A stored procedure is a set of SQL statements that is stored in the server. Clients make a single call to the stored procedure, passing parameters that can influence the procedure logic and query conditions, rather than issuing individual hardcoded SQL statements.

Stored procedures can be particularly useful in situations such as the following:

- Stored procedures can act as an API or abstraction layer, allowing multiple client applications to perform the same database operations. The applications can be written in different languages and run on different platforms. The applications do not need to hardcode table and column names, complicated

queries, and so on. When you extend and optimize the queries in a stored procedure, all the applications that call the procedure automatically receive the benefits.

- When security is paramount, stored procedures keep applications from directly manipulating tables, or even knowing details such as table and column names. Banks, for example, use stored procedures for all common operations. This provides a consistent and secure environment, and procedures can ensure that each operation is properly logged. In such a setup, applications and users would not get any access to the database tables directly, but can only execute specific stored procedures.

Connector/Net supports the calling of stored procedures through the `MySqlCommand` object. Data can be passed in and out of a MySQL stored procedure through use of the `MySqlCommand.Parameters` collection.



Note

When you call a stored procedure, the command object makes an additional `SELECT` call to determine the parameters of the stored procedure. You must ensure that the user calling the procedure has the `SELECT [697]` privilege on the `mysql.proc` table to enable them to verify the parameters. Failure to do this will result in an error when calling the procedure.

This section will not provide in-depth information on creating Stored Procedures. For such information, please refer to <http://dev.mysql.com/doc/mysql/en/stored-routines.html>.

A sample application demonstrating how to use stored procedures with Connector/Net can be found in the `Samples` directory of your Connector/Net installation.

22.2.5.9.1. Using Stored Routines from Connector/Net

Stored procedures in MySQL can be created using a variety of tools. First, stored procedures can be created using the `mysql` command-line client. Second, stored procedures can be created using MySQL Workbench. Finally, stored procedures can be created using the `.ExecuteNonQuery` method of the `MySqlCommand` object.

Unlike the command-line and GUI clients, you are not required to specify a special delimiter when creating stored procedures in Connector/Net.

To call a stored procedure using Connector/Net, you create a `MySqlCommand` object and pass the stored procedure name as the `.CommandText` property. You then set the `.CommandType` property to `CommandType.StoredProcedure`.

After the stored procedure is named, you create one `MySqlCommand` parameter for every parameter in the stored procedure. `IN` parameters are defined with the parameter name and the object containing the value, `OUT` parameters are defined with the parameter name and the data type that is expected to be returned. All parameters need the parameter direction defined.

After defining the parameters, you call the stored procedure by using the `MySqlCommand.ExecuteNonQuery()` method.

Once the stored procedure is called, the values of the output parameters can be retrieved by using the `.Value` property of the `MySqlConnection.Parameters` collection.



Note

When a stored procedure is called using `MySqlCommand.ExecuteReader`, and the stored procedure has output parameters, the output parameters are only set after the `MySqlDataReader` returned by `ExecuteReader` is closed.

The following C# example code demonstrates the use of stored procedures. It assumes the database 'employees' has already been created:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace UsingStoredRoutines
{
    class Program
    {
        static void Main(string[] args)
        {
            MySqlConnection conn = new MySqlConnection();
            conn.ConnectionString = "server=localhost;user=root;database=employees;port=3306;password=****";
            MySqlCommand cmd = new MySqlCommand();

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
                cmd.Connection = conn;
                cmd.CommandText = "DROP PROCEDURE IF EXISTS add_emp";
                cmd.ExecuteNonQuery();
                cmd.CommandText = "DROP TABLE IF EXISTS emp";
                cmd.ExecuteNonQuery();
                cmd.CommandText = "CREATE TABLE emp (empno INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
                cmd.ExecuteNonQuery();

                cmd.CommandText = "CREATE PROCEDURE add_emp(" +
                    "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno
                    "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " +
                    "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END";

                cmd.ExecuteNonQuery();
            }
            catch (MySqlException ex)
            {
                Console.WriteLine("Error " + ex.Number + " has occurred: " + ex.Message);
            }
            conn.Close();
            Console.WriteLine("Connection closed.");
            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
                cmd.Connection = conn;

                cmd.CommandText = "add_emp";
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@lname", "Jones");
                cmd.Parameters["@lname"].Direction = ParameterDirection.Input;

                cmd.Parameters.AddWithValue("@fname", "Tom");
                cmd.Parameters["@fname"].Direction = ParameterDirection.Input;

                cmd.Parameters.AddWithValue("@bday", "1940-06-07");
                cmd.Parameters["@bday"].Direction = ParameterDirection.Input;

                cmd.Parameters.AddWithValue("@empno", MySqlDbType.Int32);
                cmd.Parameters["@empno"].Direction = ParameterDirection.Output;
```

```

        cmd.ExecuteNonQuery();

        Console.WriteLine("Employee number: "+cmd.Parameters["@empno"].Value);
        Console.WriteLine("Birthday: " + cmd.Parameters["@bday"].Value);
    }
    catch (MySql.Data.MySqlClient.MySqlException ex)
    {
        Console.WriteLine("Error " + ex.Number + " has occurred: " + ex.Message);
    }
    conn.Close();
    Console.WriteLine("Done.");
}
}
}

```

The following code shows the same application in Visual Basic:

```

Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text

Imports System.Data
Imports MySql.Data
Imports MySql.Data.MySqlClient

Module Module1

    Sub Main()
        Dim conn As New MySqlConnection()
        conn.ConnectionString = "server=localhost;user=root;database=world;port=3306;password=*****;"
        Dim cmd As New MySqlCommand()

        Try
            Console.WriteLine("Connecting to MySQL...")
            conn.Open()
            cmd.Connection = conn
            cmd.CommandText = "DROP PROCEDURE IF EXISTS add_emp"
            cmd.ExecuteNonQuery()
            cmd.CommandText = "DROP TABLE IF EXISTS emp"
            cmd.ExecuteNonQuery()
            cmd.CommandText = "CREATE TABLE emp (empno INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, first"
            cmd.ExecuteNonQuery()

            cmd.CommandText = "CREATE PROCEDURE add_emp(" & "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bd"

            cmd.ExecuteNonQuery()
        Catch ex As MySqlException
            Console.WriteLine("Error " & ex.Number & " has occurred: ") + ex.Message)
        End Try
        conn.Close()
        Console.WriteLine("Connection closed.")
        Try
            Console.WriteLine("Connecting to MySQL...")
            conn.Open()
            cmd.Connection = conn

            cmd.CommandText = "add_emp"
            cmd.CommandType = CommandType.StoredProcedure

            cmd.Parameters.AddWithValue("@lname", "Jones")
            cmd.Parameters("@lname").Direction = ParameterDirection.Input

            cmd.Parameters.AddWithValue("@fname", "Tom")
            cmd.Parameters("@fname").Direction = ParameterDirection.Input

```

```

cmd.Parameters.AddWithValue("@bday", "1940-06-07")
cmd.Parameters("@bday").Direction = ParameterDirection.Input

cmd.Parameters.AddWithValue("@empno", MySqlDbType.Int32)
cmd.Parameters("@empno").Direction = ParameterDirection.Output

cmd.ExecuteNonQuery()

Console.WriteLine("Employee number: " & cmd.Parameters("@empno").Value)
Console.WriteLine("Birthday: " & cmd.Parameters("@bday").Value)
Catch ex As MySql.Data.MySqlClient.MySqlException
    Console.WriteLine(("Error " & ex.Number & " has occurred: ") + ex.Message)
End Try
conn.Close()
Console.WriteLine("Done.")

End Sub

End Module

```

22.2.5.10. Handling BLOB Data With Connector/Net

One common use for MySQL is the storage of binary data in **BLOB** columns. MySQL supports four different BLOB data types: **TINYBLOB**, **BLOB**, **MEDIUMBLOB**, and **LONGBLOB**, all described in [Section 11.4.3, “The BLOB and TEXT Types”](#) and [Section 11.6, “Data Type Storage Requirements”](#).

Data stored in a **BLOB** column can be accessed using Connector/Net and manipulated using client-side code. There are no special requirements for using Connector/Net with **BLOB** data.

Simple code examples will be presented within this section, and a full sample application can be found in the [Samples](#) directory of the Connector/Net installation.

22.2.5.10.1. Preparing the MySQL Server

The first step in using MySQL with **BLOB** data is to configure the server. Let's start by creating a table to be accessed. In my file tables, I usually have four columns: an **AUTO_INCREMENT** column of appropriate size (**UNSIGNED SMALLINT**) to serve as a primary key to identify the file, a **VARCHAR** column that stores the file name, an **UNSIGNED MEDIUMINT** column that stores the size of the file, and a **MEDIUMBLOB** column that stores the file itself. For this example, I will use the following table definition:

```

CREATE TABLE file(
file_id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
file_name VARCHAR(64) NOT NULL,
file_size MEDIUMINT UNSIGNED NOT NULL,
file MEDIUMBLOB NOT NULL);

```

After creating a table, you might need to modify the `max_allowed_packet` system variable. This variable determines how large of a packet (that is, a single row) can be sent to the MySQL server. By default, the server only accepts a maximum size of 1MB from the client application. If you intend to exceed 1MB in your file transfers, increase this number.

The `max_allowed_packet` option can be modified using the MySQL Workbench **Server Administration** screen. Adjust the Maximum permitted option in the **Data / Memory size** section of the Networking tab to an appropriate setting. After adjusting the value, click the **Apply** button and restart the server using the **Startup / Shutdown** screen of MySQL Workbench. You can also adjust this value directly in the `my.cnf` file (add a line that reads `max_allowed_packet=xxM`), or use the `SET max_allowed_packet=xxM;` syntax from within MySQL.

Try to be conservative when setting `max_allowed_packet`, as transfers of BLOB data can take some time to complete. Try to set a value that will be adequate for your intended use and increase the value if necessary.

22.2.5.10.2. Writing a File to the Database

To write a file to a database, we need to convert the file to a byte array, then use the byte array as a parameter to an `INSERT` query.

The following code opens a file using a `FileStream` object, reads it into a byte array, and inserts it into the `file` table:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

Dim SQL As String

Dim FileSize As UInt32
Dim rawData() As Byte
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    fs = New FileStream("c:\image.png", FileMode.Open, FileAccess.Read)
    FileSize = fs.Length

    rawData = New Byte(FileSize) {}
    fs.Read(rawData, 0, FileSize)
    fs.Close()

    conn.Open()

    SQL = "INSERT INTO file VALUES(NULL, @FileName, @FileSize, @File)"

    cmd.Connection = conn
    cmd.CommandText = SQL
    cmd.Parameters.AddWithValue("@FileName", strFileName)
    cmd.Parameters.AddWithValue("@FileSize", FileSize)
    cmd.Parameters.AddWithValue("@File", rawData)

    cmd.ExecuteNonQuery()

    MessageBox.Show("File Inserted into database successfully!", _
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)

    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();
```

```

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    fs = new FileStream(@"c:\image.png", FileMode.Open, FileAccess.Read);
    FileSize = fs.Length;

    rawData = new byte[FileSize];
    fs.Read(rawData, 0, FileSize);
    fs.Close();

    conn.Open();

    SQL = "INSERT INTO file VALUES(NULL, @FileName, @FileSize, @File)";

    cmd.Connection = conn;
    cmd.CommandText = SQL;
    cmd.Parameters.AddWithValue("@FileName", strFileName);
    cmd.Parameters.AddWithValue("@FileSize", FileSize);
    cmd.Parameters.AddWithValue("@File", rawData);

    cmd.ExecuteNonQuery();

    MessageBox.Show("File Inserted into database successfully!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

The `Read` method of the `FileStream` object is used to load the file into a byte array which is sized according to the `Length` property of the `FileStream` object.

After assigning the byte array as a parameter of the `MySqlCommand` object, the `ExecuteNonQuery` method is called and the `BLOB` is inserted into the `file` table.

22.2.5.10.3. Reading a BLOB from the Database to a File on Disk

Once a file is loaded into the `file` table, we can use the `MySqlDataReader` class to retrieve it.

The following code retrieves a row from the `file` table, then loads the data into a `FileStream` object to be written to disk:

Visual Basic Example

```

Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myData As MySqlDataReader
Dim SQL As String
Dim rawData() As Byte
Dim FileSize As UInt32
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _

```

```

    & "pwd=12345;" _
    & "database=test"

SQL = "SELECT file_name, file_size, file FROM file"

Try
    conn.Open()

    cmd.Connection = conn
    cmd.CommandText = SQL

    myData = cmd.ExecuteReader

    If Not myData.HasRows Then Throw New Exception("There are no BLOBs to save")

    myData.Read()

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"))
    rawData = New Byte(FileSize) {}

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize)

    fs = New FileStream("C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write)
    fs.Write(rawData, 0, FileSize)
    fs.Close()

    MessageBox.Show("File successfully written to disk!", "Success!", MessageBoxButtons.OK, MessageBoxIcon.Ast

    myData.Close()
    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;
MySQL.Data.MySqlClient.MySqlDataReader myData;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

SQL = "SELECT file_name, file_size, file FROM file";

try
{
    conn.Open();

    cmd.Connection = conn;
    cmd.CommandText = SQL;

    myData = cmd.ExecuteReader();

    if (! myData.HasRows)
        throw new Exception("There are no BLOBs to save");

    myData.Read();

```

```

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"));
    rawData = new byte[FileSize];

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, (int)FileSize);

    fs = new FileStream(@"C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write);
    fs.Write(rawData, 0, (int)FileSize);
    fs.Close();

    MessageBox.Show("File successfully written to disk!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    myData.Close();
    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

After connecting, the contents of the `file` table are loaded into a `MySqlDataReader` object. The `GetBytes` method of the `MySqlDataReader` is used to load the `BLOB` into a byte array, which is then written to disk using a `FileStream` object.

The `GetOrdinal` method of the `MySqlDataReader` can be used to determine the integer index of a named column. Use of the `GetOrdinal` method prevents errors if the column order of the `SELECT` query is changed.

22.2.5.11. Using the Connector/Net Interceptor Classes

An interceptor is a software design pattern that provides a transparent way to extend or modify some aspect of a program, similar to a user exit. No recompiling is required. With Connector/Net, the interceptors are enabled and disabled by updating the connection string to refer to different sets of interceptor classes that you instantiate.

Connector/Net includes the following interceptor classes:

- The `BaseCommandInterceptor` lets you perform additional operations when a program issues a SQL command. For example, you can examine the SQL statement for logging or debugging purposes, substitute your own result set to implement a caching mechanism, and so on. Depending on the use case, your code can supplement the SQL command or replace it entirely.

The `BaseCommandInterceptor` class has these methods that you can override:

```

public virtual bool ExecuteScalar(string sql, ref object returnValue);
public virtual bool ExecuteNonQuery(string sql, ref int returnValue);
public virtual bool ExecuteReader(string sql, CommandBehavior behavior, ref MySqlDataReader returnValue);
public virtual void Init(MySqlConnection connection);

```

If your interceptor overrides one of the `Execute...` methods, set the `returnValue` output parameter and return `true` if you handled the event, or `false` if you did not handle the event. The SQL command is processed normally only when all command interceptors return `false`.

The connection passed to the `Init` method is the connection that is attached to this interceptor.

- The `BaseExceptionInterceptor` lets you perform additional operations when a program encounters a SQL exception. The exception interception mechanism is modeled after the Connector/J model. You can code an interceptor class and connect it to an existing program without recompiling, and intercept exceptions when they are created. You can then change the exception type and optionally attach information to it. This capability lets you turn on and off logging and debugging code without hardcoding

anything in the application. This technique applies to exceptions raised at the SQL level, not to lower-level system or I/O errors.

You develop an exception interceptor first by creating a subclass of the `BaseExceptionInterceptor` class. You must override the `InterceptException()` method. You can also override the `Init()` method to do some one-time initialization.

Each exception interceptor has 2 methods:

```
public abstract Exception InterceptException(Exception exception,
    MySqlConnection connection);
public virtual void Init(MySqlConnection connection);
```

The connection passed to `Init()` is the connection that is attached to this interceptor.

Each interceptor is required to override `InterceptException` and return an exception. It can return the exception it is given, or it can wrap it in a new exception. We currently do not offer the ability to suppress the exception.

Here are examples of using the FQN (fully qualified name) on the connection string:

```
MySqlConnection c1 = new MySqlConnection(@"server=localhost;pooling=false;
commandinterceptors=CommandApp.MyCommandInterceptor,CommandApp");

MySqlConnection c2 = new MySqlConnection(@"server=localhost;pooling=false;
exceptioninterceptors=ExceptionStackTraceTest.MyExceptionInterceptor,ExceptionStackTraceTest");
```

In this example, the command interceptor is called `CommandApp.MyCommandInterceptor` and exists in the `CommandApp` assembly. The exception interceptor is called `ExceptionStackTraceTest.MyExceptionInterceptor` and exists in the `ExceptionStackTraceTest` assembly.

To shorten the connection string, you can register your exception interceptors in your `app.config` or `web.config` file like this:

```
<configSections>
<section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration,
MySql.Data"/>
</configSections>
<MySQL>
<CommandInterceptors>
  <add name="myC" type="CommandApp.MyCommandInterceptor,CommandApp" />
</CommandInterceptors>
</MySQL>

<configSections>
<section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration,
MySql.Data"/>
</configSections>
<MySQL>
<ExceptionInterceptors>
  <add name="myE"
type="ExceptionStackTraceTest.MyExceptionInterceptor,ExceptionStackTraceTest" />
</ExceptionInterceptors>
</MySQL>
```

Once you have done that, your connection strings can look like these:

```
MySqlConnection c1 = new MySqlConnection(@"server=localhost;pooling=false;
commandinterceptors=myC");
```

```
MySQLConnection c2 = new MySQLConnection(@"server=localhost;pooling=false;  
exceptioninterceptors=myE");
```

22.2.5.12. Handling Date and Time Information in Connector/Net

MySQL and the .NET languages handle date and time information differently, with MySQL allowing dates that cannot be represented by a .NET data type, such as '0000-00-00 00:00:00'. These differences can cause problems if not properly handled.

The following sections demonstrate how to properly handle date and time information when using Connector/Net.

22.2.5.12.1. Fractional Seconds

Connector/Net 6.5 and higher support the fractional seconds feature introduced in MySQL 5.6.4. Fractional seconds could always be specified in a date literal or passed back and forth as parameters and return values, but the fractional part was always stripped off when stored in a table column. In MySQL 5.6.4 and higher, the fractional part is now preserved in data stored and retrieved through SQL. For fractional second handling in MySQL 5.6.4 and higher, see [Fractional Seconds in Time Values](#). For the behavior of fractional seconds prior to MySQL 5.6.4, see [Section 11.3.6, "Fractional Seconds in Time Values"](#).

To use the more precise date and time types, specify a value from 1 to 6 when creating the table column, for example `TIME(3)` or `DATETIME(6)`, representing the number of digits of precision after the decimal point. Specifying a precision of 0 leaves the fractional part out entirely. In your C# or Visual Basic code, refer to the `Millisecond` member to retrieve the fractional second value from the `MySQLDateTime` object returned by the `GetMySQLDateTime` function. The `DateTime` object returned by the `GetDateTime` function also contains the fractional value, but only the first 3 digits.

For related code examples, see the following blog post: https://blogs.oracle.com/MySQLOnWindows/entry/milliseconds_value_support_on_datetime

22.2.5.12.2. Problems when Using Invalid Dates

The differences in date handling can cause problems for developers who use invalid dates. Invalid MySQL dates cannot be loaded into native .NET `DateTime` objects, including `NULL` dates.

Because of this issue, .NET `DataSet` objects cannot be populated by the `Fill` method of the `MySQLDataAdapter` class as invalid dates will cause a `System.ArgumentOutOfRangeException` exception to occur.

22.2.5.12.3. Restricting Invalid Dates

The best solution to the date problem is to restrict users from entering invalid dates. This can be done on either the client or the server side.

Restricting invalid dates on the client side is as simple as always using the .NET `DateTime` class to handle dates. The `DateTime` class will only allow valid dates, ensuring that the values in your database are also valid. The disadvantage of this is that it is not useful in a mixed environment where .NET and non .NET code are used to manipulate the database, as each application must perform its own date validation.

Users of MySQL 5.0.2 and higher can use the new `traditional` SQL mode to restrict invalid date values. For information on using the `traditional` SQL mode, see [Section 5.1.7, "Server SQL Modes"](#).

22.2.5.12.4. Handling Invalid Dates

Although it is strongly recommended that you avoid the use of invalid dates within your .NET application, it is possible to use invalid dates by means of the `MySQLDateTime` data type.

The `MySqlDateTime` data type supports the same date values that are supported by the MySQL server. The default behavior of Connector/Net is to return a .NET `DateTime` object for valid date values, and return an error for invalid dates. This default can be modified to cause Connector/Net to return `MySqlDateTime` objects for invalid dates.

To instruct Connector/Net to return a `MySqlDateTime` object for invalid dates, add the following line to your connection string:

```
Allow Zero Datetime=True
```

Please note that the use of the `MySqlDateTime` class can still be problematic. The following are some known issues:

1. Data binding for invalid dates can still cause errors (zero dates like 0000-00-00 do not seem to have this problem).
2. The `ToString` method return a date formatted in the standard MySQL format (for example, 2005-02-23 08:50:25). This differs from the `ToString` behavior of the .NET `DateTime` class.
3. The `MySqlDateTime` class supports NULL dates, while the .NET `DateTime` class does not. This can cause errors when trying to convert a `MySqlDateTime` to a `DateTime` if you do not check for NULL first.

Because of the known issues, the best recommendation is still to use only valid dates in your application.

22.2.5.12.5. Handling NULL Dates

The .NET `DateTime` data type cannot handle `NULL` values. As such, when assigning values from a query to a `DateTime` variable, you must first check whether the value is in fact `NULL`.

When using a `MySqlDataReader`, use the `.IsDBNull` method to check whether a value is `NULL` before making the assignment:

Visual Basic Example

```
If Not myReader.IsDBNull(myReader.GetOrdinal("mytime")) Then
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"))
Else
    myTime = DateTime.MinValue
End If
```

C# Example

```
if (! myReader.IsDBNull(myReader.GetOrdinal("mytime")))
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"));
else
    myTime = DateTime.MinValue;
```

`NULL` values will work in a data set and can be bound to form controls without special handling.

22.2.5.13. Using the `MySqlBulkLoader` Class

MySQL Connector/Net features a bulk loader class that wraps the MySQL statement `LOAD DATA INFILE`. This gives MySQL Connector/Net the ability to load a data file from a local or remote host to the server. The class concerned is `MySqlBulkLoader`. This class has various methods, the main one being `load` to cause the specified file to be loaded to the server. Various parameters can be set to control how the data file is processed. This is achieved through setting various properties of the class. For example, the field separator used, such as comma or tab, can be specified, along with the record terminator, such as newline.

The following code shows a simple example of using the [MySQLBulkLoader](#) class. First an empty table needs to be created, in this case in the `test` database:

```
CREATE TABLE Career (  
    Name VARCHAR(100) NOT NULL,  
    Age INTEGER,  
    Profession VARCHAR(200)  
);
```

A simple tab-delimited data file is also created (it could use any other field delimiter such as comma):

```
Table Career in Test Database  
Name Age Profession  
  
Tony 47 Technical Writer  
Ana 43 Nurse  
Fred 21 IT Specialist  
Simon 45 Hairy Biker
```

Note that with this test file the first three lines will need to be ignored, as they do not contain table data. This can be achieved using the [NumberOfLinesToSkip](#) property. This file can then be loaded and used to populate the `Career` table in the `test` database:

```
using System;  
using System.Text;  
using MySql.Data;  
using MySql.Data.MySqlClient;  
  
namespace ConsoleApplication1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
  
            string connStr = "server=localhost;user=root;database=test;port=3306;password=*****";  
            MySqlConnection conn = new MySqlConnection(connStr);  
  
            MySQLBulkLoader bl = new MySQLBulkLoader(conn);  
            bl.TableName = "Career";  
            bl.FieldTerminator = "\t";  
            bl.LineTerminator = "\n";  
            bl.FileName = "c:/career_data.txt";  
            bl.NumberOfLinesToSkip = 3;  
  
            try  
            {  
                Console.WriteLine("Connecting to MySQL...");  
                conn.Open();  
  
                // Upload data from file  
                int count = bl.Load();  
                Console.WriteLine(count + " lines uploaded.");  
  
                string sql = "SELECT Name, Age, Profession FROM Career";  
                MySqlCommand cmd = new MySqlCommand(sql, conn);  
                MySqlDataReader rdr = cmd.ExecuteReader();  
  
                while (rdr.Read())  
                {  
                    Console.WriteLine(rdr[0] + " -- " + rdr[1] + " -- " + rdr[2]);  
                }  
  
                rdr.Close();  
  
                conn.Close();  
            }  
        }  
    }  
}
```

```
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
    Console.WriteLine("Done.");
}
}
```

Further information on [LOAD DATA INFILE](#) can be found in [Section 13.2.6, "LOAD DATA INFILE Syntax"](#). Further information on [MySqlBulkLoader](#) can be found in the reference documentation that was included with your connector.

22.2.5.14. Using the MySQL Connector/Net Trace Source Object

MySQL Connector/Net 6.2 introduced support for .NET 2.0 compatible tracing, using [TraceSource](#) objects.

The .NET 2.0 tracing architecture consists of four main parts:

- *Source* - This is the originator of the trace information. The source is used to send trace messages. The name of the source provided by MySQL Connector/Net is [mysql](#).
- *Switch* - This defines the level of trace information to emit. Typically, this is specified in the [app.config](#) file, so that it is not necessary to recompile an application to change the trace level.
- *Listener* - Trace listeners define where the trace information will be written to. Supported listeners include, for example, the Visual Studio Output window, the Windows Event Log, and the console.
- *Filter* - Filters can be attached to listeners. Filters determine the level of trace information that will be written. While a switch defines the level of information that will be written to all listeners, a filter can be applied on a per-listener basis, giving finer grained control of trace information.

To use tracing a [TraceSource](#) object first needs to be created. To create a [TraceSource](#) object in MySQL Connector/Net you would use code similar to the following:

```
TraceSource ts = new TraceSource("mysql");
```

To enable trace messages, configure a trace switch. There are three main switch classes, [BooleanSwitch](#), [SourceSwitch](#), and [TraceSwitch](#). Trace switches also have associated with them a trace level enumeration, these are [Off](#), [Error](#), [Warning](#), [Info](#), and [Verbose](#). The following code snippet illustrates creating a switch:

```
ts.Switch = new SourceSwitch("MySwitch", "Verbose");
```

This creates a [SourceSwitch](#), called [MySwitch](#), and sets the trace level to [Verbose](#), meaning that all trace messages will be written.

It is convenient to be able to change the trace level without having to recompile the code. This is achieved by specifying the trace level in application configuration file, [app.config](#). You then simply need to specify the desired trace level in the configuration file and restart the application. The trace source is configured within the [system.diagnostics](#) section of the file. The following XML snippet illustrates this:

```
<configuration>
  ...
  <system.diagnostics>
    <sources>
```

```

    <source name="mysql" switchName="MySwitch"
           switchType="System.Diagnostics.SourceSwitch" />
    ...
</sources>
<switches>
  <add name="MySwitch" value="Verbose" />
  ...
</switches>
</system.diagnostics>
...
</configuration>

```

By default, trace information is written to the Output window of Microsoft Visual Studio. There are a wide range of listeners that can be attached to the trace source, so that trace messages can be written out to various destinations. You can also create custom listeners to allow trace messages to be written to other destinations as mobile devices and web services. A commonly used example of a listener is [ConsoleTraceListener](#), which writes trace messages to the console.

To add a listener at run time, use code such as the following:

```
ts.Listeners.Add(new ConsoleTraceListener());
```

Then, call methods on the trace source object to generate trace information. For example, the [TraceInformation\(\)](#), [TraceEvent\(\)](#), or [TraceData\(\)](#) methods can be used.

The [TraceInformation\(\)](#) method simply prints a string passed as a parameter. The [TraceEvent\(\)](#) method, as well as the optional informational string, requires a [TraceEventType](#) value to be passed to indicate the trace message type, and also an application specific ID. The [TraceEventType](#) can have a value of [Verbose](#), [Information](#), [Warning](#), [Error](#), and [Critical](#). Using the [TraceData\(\)](#) method you can pass any object, for example an exception object, instead of a message.

To ensure that these generated trace messages gets flushed from the trace source buffers to listeners, invoke the [Flush\(\)](#) method. When you are finished using a trace source, call the [Close\(\)](#) method. The [Close\(\)](#) method first calls [Flush\(\)](#), to ensure any remaining data is written out. It then frees up resources, and closes the listeners associated with the trace source.

```

ts.TraceInformation("Informational message");
ts.TraceEvent(TraceEventType.Error, 3, "Optional error message");
ts.TraceData(TraceEventType.Error, 3, ex); // pass exception object
ts.Flush();
...
ts.Close();

```

22.2.5.14.1. Viewing MySQL Trace Information

This section describes how to set up your application to view MySQL trace information.

The first thing you need to do is create a suitable [app.config](#) file for your application. An example is shown in the following code:

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <source name="mysql" switchName="SourceSwitch"
             switchType="System.Diagnostics.SourceSwitch" >
        <listeners>
          <add name="console" />
          <remove name="Default" />
        </listeners>
      </source>
    </sources>
  </system.diagnostics>
</configuration>

```

```
</source>
</sources>
<switches>
  <!-- You can set the level at which tracing is to occur -->
  <add name="SourceSwitch" value="Verbose" />
  <!-- You can turn tracing off -->
  <!--add name="SourceSwitch" value="Off" -->
</switches>
<sharedListeners>
  <add name="console"
    type="System.Diagnostics.ConsoleTraceListener"
    initializeData="false"/>
</sharedListeners>
</system.diagnostics>
</configuration>
```

This ensures a suitable trace source is created, along with a switch. The switch level in this case is set to [Verbose](#) to display the maximum amount of information.

In the application the only other step required is to add `logging=true` to the connection string. An example application could be:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using MySql.Data;
using MySql.Data.MySqlClient;
using MySql.Web;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=world;port=3306;password=*****;logging=true";
            MySqlConnection conn = new MySqlConnection(connStr);
            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'";
                MySqlCommand cmd = new MySqlCommand(sql, conn);
                MySqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Console.WriteLine(rdr[0] + " -- " + rdr[1]);
                }

                rdr.Close();

                conn.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            Console.WriteLine("Done.");
        }
    }
}
```

```
}

```

This simple application will then generate the following output:

```
Connecting to MySQL...
mysql Information: 1 : 1: Connection Opened: connection string = 'server=localhost;User Id=root;database=world;password=*****;logging=True'
mysql Information: 3 : 1: Query Opened: SHOW VARIABLES
mysql Information: 4 : 1: Resultset Opened: field(s) = 2, affected rows = -1, inserted id = -1
mysql Information: 5 : 1: Resultset Closed. Total rows=272, skipped rows=0, size (bytes)=7058
mysql Information: 6 : 1: Query Closed
mysql Information: 3 : 1: Query Opened: SHOW COLLATION
mysql Information: 4 : 1: Resultset Opened: field(s) = 6, affected rows = -1, inserted id = -1
mysql Information: 5 : 1: Resultset Closed. Total rows=127, skipped rows=0, size (bytes)=4102
mysql Information: 6 : 1: Query Closed
mysql Information: 3 : 1: Query Opened: SET character_set_results=NULL
mysql Information: 4 : 1: Resultset Opened: field(s) = 0, affected rows = 0, inserted id = 0
mysql Information: 5 : 1: Resultset Closed. Total rows=0, skipped rows=0, size (bytes)=0
mysql Information: 6 : 1: Query Closed
mysql Information: 10 : 1: Set Database: world
mysql Information: 3 : 1: Query Opened: SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'
mysql Information: 4 : 1: Resultset Opened: field(s) = 2, affected rows = -1, inserted id = -1
American Samoa -- George W. Bush
Australia -- Elisabeth II
...
Wallis and Futuna -- Jacques Chirac
Vanuatu -- John Bani
United States Minor Outlying Islands -- George W. Bush
mysql Information: 5 : 1: Resultset Closed. Total rows=28, skipped rows=0, size (bytes)=788
mysql Information: 6 : 1: Query Closed
Done.
mysql Information: 2 : 1: Connection Closed
```

The first number displayed in the trace message corresponds to the MySQL event type:

Event	Description
1	ConnectionOpened: connection string
2	ConnectionClosed:
3	QueryOpened: mysql server thread id, query text
4	ResultOpened: field count, affected rows (-1 if select), inserted id (-1 if select)
5	ResultClosed: total rows read, rows skipped, size of resultset in bytes
6	QueryClosed:
7	StatementPrepared: prepared sql, statement id
8	StatementExecuted: statement id, mysql server thread id
9	StatementClosed: statement id
10	NonQuery: [varies]
11	UsageAdvisorWarning: usage advisor flag. NoIndex = 1, BadIndex = 2, SkippedRows = 3, SkippedColumns = 4, FieldConversion = 5.
12	Warning: level, code, message
13	Error: error number, error message

The second number displayed in the trace message is the connection count.

Although this example uses the `ConsoleTraceListener`, any of the other standard listeners could have been used. Another possibility is to create a custom listener that uses the information passed using the `TraceEvent` method. For example, a custom trace listener could be created to perform active monitoring of the MySQL event messages, rather than simply writing these to an output device.

It is also possible to add listeners to the MySQL Trace Source at run time. This can be done with the following code:

```
MySQLTrace.Listeners.Add(new ConsoleTraceListener());
```

MySQL Connector/Net 6.3.2 introduced the ability to switch tracing on and off at run time. This can be achieved using the calls `MySQLTrace.EnableQueryAnalyzer(string host, int postInterval)` and `MySQLTrace.DisableQueryAnalyzer()`. The parameter `host` is the URL of the MySQL Enterprise Monitor server to monitor. The parameter `postInterval` is how often to post the data to MySQL Enterprise Monitor, in seconds.

22.2.5.14.2. Building Custom Listeners

To build custom listeners that work with the MySQL Connector/Net Trace Source, it is necessary to understand the key methods used, and the event data formats used.

The main method involved in passing trace messages is the `TraceSource.TraceEvent` method. This has the prototype:

```
public void TraceEvent(
    TraceEventType eventType,
    int id,
    string format,
    params Object[] args
)
```

This trace source method will process the list of attached listeners and call the listener's `TraceListener.TraceEvent` method. The prototype for the `TraceListener.TraceEvent` method is as follows:

```
public virtual void TraceEvent(
    TraceEventCache eventCache,
    string source,
    TraceEventType eventType,
    int id,
    string format,
    params Object[] args
)
```

The first three parameters are used in the standard as [defined by Microsoft](#). The last three parameters contain MySQL-specific trace information. Each of these parameters is now discussed in more detail.

`int id`

This is a MySQL-specific identifier. It identifies the MySQL event type that has occurred, resulting in a trace message being generated. This value is defined by the `MySQLTraceEventType` public enum contained in the MySQL Connector/Net code:

```
public enum MySQLTraceEventType : int
{
    ConnectionOpened = 1,
    ConnectionClosed,
    QueryOpened,
    ResultOpened,
    ResultClosed,
    QueryClosed,
```

```

StatementPrepared,
StatementExecuted,
StatementClosed,
NonQuery,
UsageAdvisorWarning,
Warning,
Error
}

```

The MySQL event type also determines the contents passed using the parameter `params Object[] args`. The nature of the `args` parameters are described in further detail in the following material.

string format

This is the format string that contains zero or more format items, which correspond to objects in the `args` array. This would be used by a listener such as `ConsoleTraceListener` to write a message to the output device.

params Object[] args

This is a list of objects that depends on the MySQL event type, `id`. However, the first parameter passed using this list is always the driver id. The driver id is a unique number that is incremented each time the connector is opened. This enables groups of queries on the same connection to be identified. The parameters that follow driver id depend on the MySQL event id, and are as follows:

MySQL-specific event type	Arguments (params Object[] args)
ConnectionOpened	Connection string
ConnectionClosed	No additional parameters
QueryOpened	mysql server thread id, query text
ResultOpened	field count, affected rows (-1 if select), inserted id (-1 if select)
ResultClosed	total rows read, rows skipped, size of resultset in bytes
QueryClosed	No additional parameters
StatementPrepared	prepared sql, statement id
StatementExecuted	statement id, mysql server thread id
StatementClosed	statement id
NonQuery	Varies
UsageAdvisorWarning	usage advisor flag. NoIndex = 1, BadIndex = 2, SkippedRows = 3, SkippedColumns = 4, FieldConversion = 5.
Warning	level, code, message
Error	error number, error message

This information will allow you to create custom trace listeners that can actively monitor the MySQL-specific events.

22.2.5.15. Binary/Nonbinary Issues

There are certain situations where MySQL will return incorrect metadata about one or more columns. More specifically, the server will sometimes report that a column is binary when it is not and vice versa. In these situations, it becomes practically impossible for the connector to be able to correctly identify the correct metadata.

Some examples of situations that may return incorrect metadata are:

- Execution of `SHOW PROCESSLIST`. Some of the columns will be returned as binary even though they only hold string data.
- When a temporary table is used to process a resultset, some columns may be returned with incorrect binary flags.
- Some server functions such `DATE_FORMAT` will incorrectly return the column as binary.

With the availability of `BINARY` and `VARBINARY` data types, it is important that we respect the metadata returned by the server. However, we are aware that some existing applications may break with this change, so we are creating a connection string option to enable or disable it. By default, Connector/Net 5.1 respects the binary flags returned by the server. You might need to make small changes to your application to accommodate this change.

In the event that the changes required to your application would be too large, adding `'respect binary flags=false'` to your connection string causes the connector to use the prior behavior: any column that is marked as string, regardless of binary flags, will be returned as string. Only columns that are specifically marked as a `BLOB` will be returned as `BLOB`.

22.2.5.16. Character Set Considerations for Connector/Net

Treating Binary Blobs As UTF8

MySQL doesn't currently support 4-byte UTF8 sequences. This makes it difficult to represent some multi-byte languages such as Japanese. To try and alleviate this, Connector/Net now supports a mode where binary blobs can be treated as strings.

To do this, you set the `'Treat Blobs As UTF8'` connection string keyword to `yes`. This is all that needs to be done to enable conversion of all binary blobs to UTF8 strings. To convert only some of your BLOB columns, you can make use of the `'BlobAsUTF8IncludePattern'` and `'BlobAsUTF8ExcludePattern'` keywords. Set these to a regular expression pattern that matches the column names to include or exclude respectively.

When the regular expression patterns both match a single column, the include pattern is applied before the exclude pattern. The result, in this case, would be that the column would be excluded. Also, be aware that this mode does not apply to columns of type `BINARY` or `VARBINARY` and also do not apply to nonbinary `BLOB` columns.

Currently, this mode only applies to reading strings out of MySQL. To insert 4-byte UTF8 strings into blob columns, use the `.NET Encoding.GetBytes` function to convert your string to a series of bytes. You can then set this byte array as a parameter for a `BLOB` column.

22.2.5.17. Using Connector/Net with Crystal Reports

Crystal Reports is a common tool used by Windows application developers to perform reporting and document generation. In this section we will show how to use Crystal Reports XI with MySQL and Connector/Net.

22.2.5.17.1. Creating a Data Source

When creating a report in Crystal Reports there are two options for accessing the MySQL data while designing your report.

The first option is to use Connector/ODBC as an ADO data source when designing your report. You will be able to browse your database and choose tables and fields using drag and drop to build your report. The disadvantage of this approach is that additional work must be performed within your application to produce a data set that matches the one expected by your report.

The second option is to create a data set in VB.NET and save it as XML. This XML file can then be used to design a report. This works quite well when displaying the report in your application, but is less versatile at design time because you must choose all relevant columns when creating the data set. If you forget a column you must re-create the data set before the column can be added to the report.

The following code can be used to create a data set from a query and write it to disk:

Visual Basic Example

```
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"

Try
    conn.Open()
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myData.WriteXml("C:\dataset.xml", XmlWriteMode.WriteSchema)
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myData.WriteXml(@"C:\dataset.xml", XmlWriteMode.WriteSchema);
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

The resulting XML file can be used as an ADO.NET XML datasource when designing your report.

If you choose to design your reports using Connector/ODBC, it can be downloaded from dev.mysql.com.

22.2.5.17.2. Creating the Report

For most purposes, the Standard Report wizard helps with the initial creation of a report. To start the wizard, open Crystal Reports and choose the **New > Standard Report** option from the File menu.

The wizard first prompts you for a data source. If you use Connector/ODBC as your data source, use the OLEDB provider for ODBC option from the OLE DB (ADO) tree instead of the ODBC (RDO) tree when choosing a data source. If using a saved data set, choose the ADO.NET (XML) option and browse to your saved data set.

The remainder of the report creation process is done automatically by the wizard.

After the report is created, choose the Report Options... entry of the File menu. Un-check the Save Data With Report option. This prevents saved data from interfering with the loading of data within our application.

22.2.5.17.3. Displaying the Report

To display a report we first populate a data set with the data needed for the report, then load the report and bind it to the data set. Finally we pass the report to the crViewer control for display to the user.

The following references are needed in a project that displays a report:

- CrystalDecisions.CrystalReports.Engine
- CrystalDecisions.ReportSource
- CrystalDecisions.Shared
- CrystalDecisions.Windows.Forms

The following code assumes that you created your report using a data set saved using the code shown in [Section 22.2.5.17.1, "Creating a Data Source"](#), and have a crViewer control on your form named `myViewer`.

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = _
    "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()
```

```

cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
    & "country.name, country.population, country.continent " _
    & "FROM country, city ORDER BY country.continent, country.name"
cmd.Connection = conn

myAdapter.SelectCommand = cmd
myAdapter.Fill(myData)

myReport.Load(".\world_report.rpt")
myReport.SetDataSource(myData)
myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.SetDataSource(myData);
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

A new data set is generated using the same query used to generate the previously saved data set. Once the data set is filled, a ReportDocument is used to load the report file and bind it to the data set. The ReportDocument is then passed as the ReportSource of the crViewer.

This same approach is taken when a report is created from a single table using Connector/ODBC. The data set replaces the table used in the report and the report is displayed properly.

When a report is created from multiple tables using Connector/ODBC, a data set with multiple tables must be created in our application. This enables each table in the report data source to be replaced with a report in the data set.

We populate a data set with multiple tables by providing multiple `SELECT` statements in our `MySqlCommand` object. These `SELECT` statements are based on the SQL query shown in Crystal Reports in the Database menu's Show SQL Query option. Assume the following query:

```
SELECT `country`.`Name`, `country`.`Continent`, `country`.`Population`, `city`.`Name`, `city`.`Population`
FROM `world`.`country` `country` LEFT OUTER JOIN `world`.`city` `city` ON `country`.`Code`=`city`.`CountryCode`
ORDER BY `country`.`Continent`, `country`.`Name`, `city`.`Name`
```

This query is converted to two `SELECT` queries and displayed with the following code:

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"

Try
    conn.Open()
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER BY countrycode, name;" _
        & "SELECT name, population, code, continent FROM country ORDER BY continent, name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0))
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1))
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER " +
```

```

        "BY countrycode, name; SELECT name, population, code, continent FROM " +
        "country ORDER BY continent, name";
        cmd.Connection = conn;

        myAdapter.SelectCommand = cmd;
        myAdapter.Fill(myData);

        myReport.Load(@".\world_report.rpt");
        myReport.Database.Tables(0).SetDataSource(myData.Tables(0));
        myReport.Database.Tables(1).SetDataSource(myData.Tables(1));
        myViewer.ReportSource = myReport;
    }
    catch (MySql.Data.MySqlClient.MySqlException ex)
    {
        MessageBox.Show(ex.Message, "Report could not be created",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

It is important to order the `SELECT` queries in alphabetic order, as this is the order the report will expect its source tables to be in. One `SetDataSource` statement is needed for each table in the report.

This approach can cause performance problems because Crystal Reports must bind the tables together on the client-side, which will be slower than using a pre-saved data set.

22.2.5.18. ASP.NET Provider Model

MySQL Connector/Net provides support for the ASP.NET 2.0 provider model. This model enables application developers to focus on the business logic of their application instead of having to recreate such boilerplate items as membership and roles support.

MySQL Connector/Net supplies the following providers:

- Membership Provider
- Role Provider
- Profile Provider
- Session State Provider (MySQL Connector/Net 6.1 and later)

The following tables show the supported providers, their default provider and the corresponding MySQL provider.

Membership Provider

Default Provider	MySQL Provider
System.Web.Security.SqlMembershipProvider	MySql.Web.Security.MySQLMembershipProvider

Role Provider

Default Provider	MySQL Provider
System.Web.Security.SqlRoleProvider	MySql.Web.Security.MySQLRoleProvider

Profile Provider

Default Provider	MySQL Provider
System.Web.Profile.SqlProfileProvider	MySql.Web.Profile.MySQLProfileProvider

SessionState Provider

Default Provider	MySQL Provider
System.Web.SessionState.InProcSessionStateStore	MySql.Web.SessionState.MySqlSessionStateStore



Note

The MySQL Session State provider uses slightly different capitalization on the class name compared to the other MySQL providers.

Installing The Providers

The installation of Connector/Net 5.1 or later will install the providers and register them in your machine's .NET configuration file, `machine.config`. The additional entries created will result in the `system.web` section appearing similar to the following code:

```
<system.web>
  <processModel autoConfig="true" />
  <httpHandlers />
  <membership>
    <providers>
      <add name="AspNetSqlMembershipProvider" type="System.Web.Security.SqlMembershipProvider, System.Web, Ver
      <add name="MySQLMembershipProvider" type="MySql.Web.Security.MySQLMembershipProvider, MySql.Web, Version
    </providers>
  </membership>
  <profile>
    <providers>
      <add name="AspNetSqlProfileProvider" connectionStringName="LocalSqlServer" applicationName="/" type="Sys
      <add name="MySQLProfileProvider" type="MySql.Web.Profile.MySQLProfileProvider, MySql.Web, Version=6.1.1.
    </providers>
  </profile>
  <roleManager>
    <providers>
      <add name="AspNetSqlRoleProvider" connectionStringName="LocalSqlServer" applicationName="/" type="System
      <add name="AspNetWindowsTokenRoleProvider" applicationName="/" type="System.Web.Security.WindowsTokenRol
      <add name="MySQLRoleProvider" type="MySql.Web.Security.MySQLRoleProvider, MySql.Web, Version=6.1.1.0, Cu
    </providers>
  </roleManager>
</system.web>
```

Each provider type can have multiple provider implementations. The default provider can also be set here using the `defaultProvider` attribute, but usually this is set in the `web.config` file either manually or by using the ASP.NET configuration tool.

At time of writing, the `MySqlSessionStateStore` is not added to `machine.config` at install time, and so add the following:

```
<sessionState>
  <providers>
    <add name="MySqlSessionStateStore" type="MySql.Web.SessionState.MySqlSessionStateStore, MySql.Web, Version
  </providers>
</sessionState>
```

The SessionState Provider uses the `customProvider` attribute, rather than `defaultProvider`, to set the provider as the default. A typical `web.config` file might contain:

```
<system.web>
  <membership defaultProvider="MySQLMembershipProvider" />
  <roleManager defaultProvider="MySQLRoleProvider" />
  <profile defaultProvider="MySQLProfileProvider" />
  <sessionState customProvider="MySqlSessionStateStore" />
  <compilation debug="false">
```

...

This sets the MySQL Providers as the defaults to be used in this web application.

The providers are implemented in the file `mysql.web.dll` and this file can be found in your MySQL Connector/Net installation folder. There is no need to run any type of SQL script to set up the database schema, as the providers create and maintain the proper schema automatically.

Using The Providers

The easiest way to start using the providers is to use the ASP.NET configuration tool that is available on the Solution Explorer toolbar when you have a website project loaded.

In the web pages that open, you can select the MySQL membership and roles providers by picking a custom provider for each area.

When the provider is installed, it creates a dummy connection string named `LocalMySqlServer`. Although this has to be done so that the provider will work in the ASP.NET configuration tool, you override this connection string in your `web.config` file. You do this by first removing the dummy connection string and then adding in the proper one, as shown in the following example:

```
<connectionStrings>
  <remove name="LocalMySqlServer" />
  <add name="LocalMySqlServer" connectionString="server=xxx;uid=xxx;pwd=xxx;database=xxx;" />
</connectionStrings>
```

Note the database to connect to must be specified.

Rather than manually editing configuration files, consider using the MySQL Website Configuration tool to configure your desired provider setup. From MySQL Connector/Net 6.1.1 onwards, all providers can be selected and configured from this wizard. The tool modifies your `website.config` file to the desired configuration. A tutorial on doing this is available in the following section [Section 22.2.3.12, "MySQL Website Configuration Tool"](#).

A tutorial demonstrating how to use the Membership and Role Providers can be found in the following section [Section 22.2.4.2, "Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider"](#).

Deployment

To use the providers on a production server, distribute the `MySql.Data` and the `MySql.Web` assemblies, and either register them in the remote systems Global Assembly Cache or keep them in your application's `bin/` directory.

22.2.5.19. Working with Partial Trust / Medium Trust

.NET applications operate under a given trust level. Normal desktop applications operate under full trust, while web applications that are hosted in shared environments are normally run under the partial trust level (also known as "medium trust"). Some hosting providers host shared applications in their own app pools and allow the application to run under full trust, but this configuration is relatively rare. The Connector/Net support for partial trust has improved over time to simplify the configuration and deployment process for hosting providers.

22.2.5.19.1. Evolution of Partial Trust Support Across Connector/Net Versions

The partial trust support for Connector/Net has improved rapidly throughout the 6.5.x and 6.6.x versions. The latest enhancements do require some configuration changes in existing deployments. Here is a summary of the changes for each version.

6.6.4 and Above: Library Can Be Inside or Outside GAC

Now you can install the `MySql.Data.dll` library in the Global Assembly Cache (GAC) as explained in [Section 22.2.5.19.2, “Configuring Partial Trust with Connector/Net Library Installed in GAC”](#), or in a `bin` or `lib` folder inside the project or solution as explained in [Section 22.2.5.19.3, “Configuring Partial Trust with Connector/Net Library Not Installed in GAC”](#). If the library is not in the GAC, the only protocol supported is TCP/IP.

6.5.1 and Above: Partial Trust Requires Library in the GAC

Connector/Net 6.5 fully enables our provider to run in a partial trust environment when the library is installed in the Global Assembly Cache (GAC). The new `MySqlClientPermission` class, derived from the .NET `DBDataPermission` class, helps to simplify the permission setup.

5.0.8 / 5.1.3 and Above: Partial Trust Requires Socket Permissions

Starting with these versions, Connector/Net can be used under partial trust hosting that has been modified to allow the use of sockets for communication. By default, partial trust does not include `SocketPermission`. Connector/Net uses sockets to talk with the MySQL server, so the hosting provider must create a new trust level that is an exact clone of partial trust but that has the following permissions added:

- `System.Net.SocketPermission`
- `System.Security.Permissions.ReflectionPermission`
- `System.Net.DnsPermission`
- `System.Security.Permissions.SecurityPermission`

Prior to 5.0.8 / 5.1.3: Partial Trust Not Supported

Connector/Net versions prior to 5.0.8 and 5.1.3 were not compatible with partial trust hosting.

22.2.5.19.2. Configuring Partial Trust with Connector/Net Library Installed in GAC

If the library is installed in the GAC, you must include the connection option `includesecurityasserts=true` in your connection string. This is a new requirement as of Connector/Net 6.6.4.

The following list shows steps and code fragments needed to run a Connector/Net application in a partial trust environment. For illustration purposes, we use the Pipe Connections protocol in this example.

1. Install Connector/Net: version 6.6.1 or higher, or 6.5.4 or higher.
2. After installing the library, make the following configuration changes:

In the `SecurityClasses` section, add a definition for the `MySqlClientPermission` class, including the version to use.

```
<configuration>
  <mscorlib>
    <security>
      <policy>
        <PolicyLevel version="1">
          <SecurityClasses>
            ....
            <SecurityClass Name="MySqlClientPermission" Description="MySql.Data.MySqlClient.MySqlClientPerm
```

Scroll down to the `ASP.Net` section:

```
<PermissionSet class="NamedPermissionSet" version="1" Name="ASP.Net">
```

Add a new entry for the detailed configuration of the `MySQLClientPermission` class:

```
<IPermission class="MySQLClientPermission" version="1" Unrestricted="true"/>
```

Note: This configuration is the most generalized way that includes all keywords.

3. Configure the MySQL server to accept pipe connections, by adding the `--enable-named-pipe` option on the command line. If you need more information about this, see [Section 2.3, "Installing MySQL on Microsoft Windows"](#).
4. Confirm that the hosting provider has installed the Connector/Net library (`MySQL.Data.dll`) in the GAC.
5. Optionally, the hosting provider can avoid granting permissions globally by using the new `MySQLClientPermission` class in the trust policies. (The alternative is to globally enable the permissions `System.Net.SocketPermission`, `System.Security.Permissions.ReflectionPermission`, `System.Net.DnsPermission`, and `System.Security.Permissions.SecurityPermission`.)
6. Create a simple web application using Visual Studio 2010.
7. Add the reference in your application for the `MySQL.Data.MySqlClient` library.
8. Edit your `web.config` file so that your application runs using a Medium trust level:

```
<system.web>
  <trust level="Medium"/>
</system.web>
```

9. Add the `MySQL.Data.MySqlClient` namespace to your server-code page.
10. Define the connection string, in slightly different ways depending on the Connector/Net version.

Only for 6.6.4 or later: To use the connections inside any web application that will run in Medium trust, add the new `includesecurityasserts` option to the connection string. `includesecurityasserts=true` that makes the library request the following permissions when required: `SocketPermissions`, `ReflectionPermissions`, `DnsPermissions`, `SecurityPermissions` among others that are not granted in Medium trust levels.

For Connector/Net 6.6.3 or earlier: No special setting for security is needed within the connection string.

```
MySQLConnectionStringBuilder myconnString = new MySQLConnectionStringBuilder("server=localhost;User Id=
myconnString.PipeName = "MySQL55";
myconnString.ConnectionProtocol = MySQLConnectionProtocol.Pipe;
// Following attribute is a new requirement when the library is in the GAC.
// Could also be done by adding includesecurityasserts=true; to the string literal
// in the constructor above.
// Not needed with Connector/Net 6.6.3 and earlier.myconnString.IncludeSecurityAsserts = true;
```

11. Define the `MySQLConnection` to use:

```
MySQLConnection myconn = new MySQLConnection(myconnString.ConnectionString);
myconn.Open();
```

12. Retrieve some data from your tables:

```
MySQLCommand cmd = new MySQLCommand("Select * from products", myconn);
MySQLDataAdapter da = new MySQLDataAdapter(cmd);
DataSet1 tds = new DataSet1();
da.Fill(tds, tds.Tables[0].TableName);
GridView1.DataSource = tds;
GridView1.DataBind();
myconn.Close();
```

13. Run the program. It should execute successfully, without requiring any special code or encountering any security problems.

22.2.5.19.3. Configuring Partial Trust with Connector/Net Library Not Installed in GAC

When deploying a web application to a Shared Hosted environment, where this environment is configured to run all their .NET applications under a partial or medium trust level, you might not be able to install the Connector/Net library in the GAC. Instead, you put a reference to the library in the `bin` or `lib` folder inside the project or solution. In this case, you configure the security in a different way than when the library is in the GAC.

Connector/Net is commonly used by applications that run in Windows environments where the default communication for the protocol is used via sockets or by TCP/IP. For this protocol to operate is necessary have the required socket permissions in the web configuration file as follows:

1. Open the medium trust policy web configuration file, which should be under this folder:

```
%windir%\Microsoft.NET\Framework\{version}\CONFIG\web_mediumtrust.config
```

Use `Framework64` in the path instead of `Framework` if you are using a 64-bit installation of the framework.

2. Locate the `SecurityClasses` tag:

```
<SecurityClass Name="SocketPermission"
Description="System.Net.SocketPermission, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
```

3. Scroll down and look for the following `PermissionSet`:

```
<PermissionSet version="1" Name="ASP.Net">
```

4. Add the following inside this `PermissionSet`:

```
<IPermission class="SocketPermission" version="1" Unrestricted="true" />
```

This configuration lets you use the driver with the default Windows protocol TCP/IP without having any security issues. This approach only supports the TCP/IP protocol, so you cannot use any other type of connection.

Also, since the `MySQLClientPermissions` class is not added to the medium trust policy, you cannot use it. This configuration is the minimum required in order to work with Connector/Net without the GAC.

22.2.6. Connector/Net Connection String Options Reference

For usage information about connection strings, see [Section 22.2.5.2, "Creating a Connector/Net Connection String"](#). The first table list options that apply generally to all server configurations. The options related to systems using a connection pool are split into a separate table.

General Options

Table 22.20. Connector/Net Connection String Options - General

Name	Default	Description
<code>Allow Batch, AllowBatch</code>	true	When true, multiple SQL statements can be sent with one command execution. Note: starting with MySQL 4.1.1, batch statements should be separated by the server-defined separator character. Statements sent to earlier versions of MySQL should be separated by ';'.
<code>Allow User Variables, AllowUserVariables</code>	false	Setting this to <code>true</code> indicates that the provider expects user variables in the SQL. This option was added in Connector/Net version 5.2.2.
<code>Allow Zero Datetime, AllowZeroDateTime</code>	false	If set to <code>True</code> , <code>MySqlDataReader.GetValue()</code> returns a <code>MySqlDateTime</code> object for date or datetime columns that have disallowed values, such as zero datetime values, and a <code>System.DateTime</code> object for valid values. If set to <code>False</code> (the default setting) it causes a <code>System.DateTime</code> object to be returned for all valid values and an exception to be thrown for disallowed values, such as zero datetime values.
<code>Auto Enlist, AutoEnlist</code>	true	If <code>AutoEnlist</code> is set to <code>true</code> , which is the default, a connection opened using <code>TransactionScope</code> participates in this scope, it commits when the scope commits and rolls back if <code>TransactionScope</code> does not commit. However, this feature is considered security sensitive and therefore cannot be used in a medium trust environment.
<code>BlobAsUTF8ExcludePattern</code>	null	A POSIX-style regular expression that matches the names of BLOB columns that do not contain UTF-8 character data. See Section 22.2.5.16, "Character Set Considerations for Connector/Net" for usage details.
<code>BlobAsUTF8IncludePattern</code>	null	A POSIX-style regular expression that matches the names of BLOB columns containing UTF-8 character data. See Section 22.2.5.16, "Character Set Considerations for Connector/Net" for usage details.
<code>Certificate File, CertificateFile</code>	null	This option specifies the path to a certificate file in PKCS #12 format (<code>.pfx</code>). For an example of usage, see Section 22.2.4.7, "Tutorial: Using SSL with MySQL Connector/Net" . Was introduced with 6.2.1.
<code>Certificate Password, CertificatePassword</code>	null	Specifies a password that is used in conjunction with a certificate specified using the option <code>CertificateFile</code> . For an example of usage, see Section 22.2.4.7, "Tutorial: Using SSL with MySQL Connector/Net" . Was introduced with 6.2.1.
<code>Certificate Store Location, CertificateStoreLocation</code>	null	Enables you to access a certificate held in a personal store, rather than use a certificate file and password combination. For an example of usage, see Section 22.2.4.7, "Tutorial: Using SSL with MySQL Connector/Net" . Was introduced with 6.2.1.

Connector/Net Connection String Options Reference

Name	Default	Description
<code>CertificateThumbprint</code> , <code>CertificateThumbprint</code>	null	Specifies a certificate thumbprint to ensure correct identification of a certificate contained within a personal store. For an example of usage, see Section 22.2.4.7 , “ Tutorial: Using SSL with MySQL Connector/Net ”. Was introduced with 6.2.1.
<code>CharSet</code> , <code>Character Set</code> , <code>CharacterSet</code>		Specifies the character set that should be used to encode all queries sent to the server. Resultsets are still returned in the character set of the result data.
<code>Check Parameters</code> , <code>CheckParameters</code>	true	Indicates if stored routine parameters should be checked against the server.
<code>Command Interceptors</code> , <code>CommandInterceptors</code>		The list of interceptors that can intercept SQL command operations.
<code>Connect Timeout</code> , <code>Connection Timeout</code> , <code>ConnectionTimeout</code>	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
<code>Convert Zero Datetime</code> , <code>ConvertZeroDateTime</code>	false	True to have <code>MySqlDataReader.GetValue()</code> and <code>MySqlDataReader.GetDateTime()</code> return <code>DateTime.MinValue</code> for date or datetime columns that have disallowed values.
<code>Default Command Timeout</code> , <code>DefaultCommandTimeout</code>	30	Sets the default value of the command timeout to be used. This does not supercede the individual command timeout property on an individual command object. If you set the command timeout property, that will be used. This option was added in Connector/Net 5.1.4
<code>Default Table Cache Age</code> , <code>DefaultTableCacheAge</code>	60	Specifies how long a <code>TableDirect</code> result should be cached, in seconds. For usage information about table caching, see Section 22.2.5.7 , “ Using Connector/Net with Table Caching ”. This option was added in Connector/Net 6.4.
<code>enableSessionExpireCallback</code>	false	When set to <code>true</code> , causes the session-expiry scanner to raise the <code>session_end</code> event before deleting the session data stored in the <code>my_aspnet_sessions</code> table, when a session times out. Enable this option to write additional application-specific cleanup code to handle the <code>session_end</code> event of the <code>global.asax</code> class, before the stored data of the session gets deleted. Within the <code>session_end</code> method, any other required cleanup can be done. This option was added in Connector/Net 6.4.5; it is not yet available in Connector/Net 6.5.x releases.
<code>Encrypt</code> , <code>UseSSL</code>	false	For Connector/Net 5.0.3 and later, when <code>true</code> , SSL encryption is used for all data sent between the client and server if the server has a certificate installed. Recognized values are <code>true</code> , <code>false</code> , <code>yes</code> , and <code>no</code> . In versions before 5.0.3, this option had no effect. From version 6.2.1, this option is deprecated and is replaced by <code>SSL Mode</code> . The option still works if used. If this option is set to true, it is equivalent to <code>SSL Mode = Preferred</code> .

Connector/Net Connection String Options Reference

Name	Default	Description
Exception Interceptors , ExceptionInterceptors		The list of interceptors that can triage thrown MySQLException exceptions.
Functions Return String , FunctionsReturnString	false	Causes the connector to return binary/varbinary values as strings, if they do not have a tablename in the metadata.
Host , Server , Data Source , DataSource , Address , Addr , Network Address	localhost	The name or network address of the instance of MySQL to which to connect. Multiple hosts can be specified separated by commas. This can be useful where multiple MySQL servers are configured for replication and you are not concerned about the precise server you are connecting to. No attempt is made by the provider to synchronize writes to the database, so take care when using this option. In Unix environment with Mono, this can be a fully qualified path to a MySQL socket file. With this configuration, the Unix socket is used instead of the TCP/IP socket. Currently, only a single socket name can be given, so accessing MySQL in a replicated environment using Unix sockets is not currently supported.
Ignore Prepare , IgnorePrepare	true	When true, instructs the provider to ignore any calls to MySQLCommand.Prepare() . This option is provided to prevent issues with corruption of the statements when used with server-side prepared statements. If you use server-side prepare statements, set this option to false. This option was added in Connector/Net 5.0.3 and Connector/Net 1.0.9.
includesecurityasserts , include security asserts	false	Must be set to <code>true</code> when using the MySQLClientPermissions class in a partial trust environment, with the library installed in the GAC of the hosting environment. This requirement is new for partial-trust applications in Connector/Net 6.6.4 and higher. See Section 22.2.5.19, "Working with Partial Trust / Medium Trust" for details.
Initial Catalog , Database	mysql	The case-sensitive name of the database to use initially.
Interactive , Interactive Session , InteractiveSession	false	If set to true, the client is interactive. An interactive client is one where the server variable <code>CLIENT_INTERACTIVE</code> is set. If an interactive client is set, the <code>wait_timeout</code> variable is set to the value of <code>interactive_timeout</code> . The client will then time out after this period of inactivity. For more details, see Section 5.1.4, "Server System Variables" in the MySQL Reference Manual.
Integrated Security , IntegratedSecurity	no	Use Windows authentication when connecting to server. By default, it is turned off. To enable, specify a value of <code>yes</code> . (You can also use the value <code>sspi</code> as an alternative to <code>yes</code> .) For details, see Section 22.2.5.5, "Using the Windows Native Authentication Plugin" . This option was introduced in Connector/Net 6.4.4.
Keep Alive , Keepalive	0	For TCP connections, idle connection time measured in seconds, before the first keepalive packet is sent. A value of 0 indicates that keepalive is not used.

Connector/Net Connection String Options Reference

Name	Default	Description
<code>Logging</code>	false	When true, various pieces of information is output to any configured TraceListeners. See Section 22.2.5.14, "Using the MySQL Connector/Net Trace Source Object" for further details.
<code>Old Guids, OldGuids</code>	false	This option was introduced in Connector/Net 6.1.1. The backend representation of a GUID type was changed from <code>BINARY(16)</code> to <code>CHAR(36)</code> . This was done to allow developers to use the server function <code>UUID() [1201]</code> to populate a GUID table - <code>UUID()</code> generates a 36-character string. Developers of older applications can add <code>'Old Guids=true'</code> to the connection string to use a GUID of data type <code>BINARY(16)</code> .
<code>Old Syntax, OldSyntax, Use Old Syntax, UseOldSyntax</code>	false	This option was deprecated in Connector/Net 5.2.2. All code should now be written using the '@' symbol as the parameter marker.
<code>Password, pwd</code>		The password for the MySQL account being used.
<code>Persist Security Info, PersistSecurityInfo</code>	false	When set to <code>false</code> or <code>no</code> (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values, including the password. Recognized values are <code>true</code> , <code>false</code> , <code>yes</code> , and <code>no</code> .
<code>Pipe Name, Pipe, PipeName</code>	mysql	When set to the name of a named pipe, the <code>MySqlConnection</code> attempts to connect to MySQL on that named pipe. This setting only applies to the Windows platform.
<code>Port</code>	3306	The port MySQL is using to listen for connections. This value is ignored if Unix socket is used.
<code>Procedure Cache Size, ProcedureCacheSize, procedure cache, procedurecache</code>	25	Sets the size of the stored procedure cache. By default, Connector/Net stores the metadata (input/output data types) about the last 25 stored procedures used. To disable the stored procedure cache, set the value to zero (0). This option was added in Connector/Net 5.0.2 and Connector/Net 1.0.9.
<code>Protocol, Connection Protocol, ConnectionProtocol</code>	socket	Specifies the type of connection to make to the server. Values can be: <code>socket</code> or <code>tcp</code> for a socket connection, <code>pipe</code> for a named pipe connection, <code>unix</code> for a Unix socket connection, <code>memory</code> to use MySQL shared memory.
<code>Replication</code>	false	Indicates if this connection is to use replicated servers.
<code>Respect Binary Flags, RespectBinaryFlags</code>	true	Setting this option to <code>false</code> means that Connector/Net ignores a column's binary flags as set by the server. This option was added in Connector/Net version 5.1.3.
<code>Shared Memory Name, SharedMemoryName</code>	MYSQL	The name of the shared memory object to use for communication if the connection protocol is set to <code>memory</code> .

Name	Default	Description
Sql Server Mode, sqlservermode	false	Allow SQL Server syntax. When set to <code>true</code> , enables Connector/Net to support square brackets around symbols instead of backticks. This enables Visual Studio wizards that bracket symbols with <code>[]</code> to work with Connector/Net. This option incurs a performance hit, so should only be used if necessary. This option was added in version 6.3.1.
SSL Mode, SslMode	None	<p>This option has the following values:</p> <ul style="list-style-type: none"> • None - do not use SSL. • Preferred - use SSL if the server supports it, but allow connection in all cases. • Required - Always use SSL. Deny connection if server does not support SSL. • VerifyCA - Always use SSL. Validate the CA but tolerate name mismatch. • VerifyFull - Always use SSL. Fail if the host name is not correct. <p>This option was introduced in MySQL Connector/Net 6.2.1.</p>
Table Cache, tablecache, tablecaching	false	Enables or disables caching of <code>TableDirect</code> commands. A value of <code>true</code> enables the cache while <code>false</code> disables it. For usage information about table caching, see Section 22.2.5.7, “Using Connector/Net with Table Caching” . This option was added in Connector/Net 6.4.
Treat BLOBs as UTF8, TreatBlobsAsUTF8	false	
Treat Tiny As Boolean, TreatTinyAsBoolean	true	Setting this value to <code>false</code> causes <code>TINYINT(1)</code> to be treated as an <code>INT</code> . See Section 11.1.1, “Numeric Type Overview” for a further explanation of the <code>TINYINT</code> and <code>BOOL</code> data types.
Use Affected Rows, UseAffectedRows	false	When <code>true</code> , the connection reports changed rows instead of found rows. This option was added in Connector/Net version 5.2.6.
Use Procedure Bodies, UseProcedureBodies, procedure bodies	true	When set to <code>true</code> , the default value, MySQL Connector/Net expects the body of the procedure to be viewable. This enables it to determine the parameter types and order. Set the option to <code>false</code> when the user connecting to the database does not have the SELECT [697] privileges for the <code>mysql.proc</code> (stored procedures) table, or cannot view <code>INFORMATION_SCHEMA.ROUTINES</code> . In this case, MySQL Connector/Net cannot determine the types and order of the parameters, and must be alerted to this fact by setting this option to <code>false</code> . When set to <code>false</code> , MySQL Connector/Net does not rely on

Name	Default	Description
		this information being available when the procedure is called. Because MySQL Connector/Net will not be able to determine this information, explicitly set the types of all the parameters before the call and add the parameters to the command in the same order as they appear in the procedure definition. This option was added in MySQL Connector/Net 5.0.4 and MySQL Connector/Net 1.0.10.
<code>User Id, UserID, Username, Uid, User name, User</code>		The MySQL login account being used.
<code>Compress, Use Compression, UseCompression</code>	false	<p>Setting this option to <code>true</code> enables compression of packets exchanged between the client and the server. This exchange is defined by the MySQL client/server protocol.</p> <p>Compression is used if both client and server support ZLIB compression, and the client has requested compression using this option.</p> <p>A compressed packet header is: packet length (3 bytes), packet number (1 byte), and Uncompressed Packet Length (3 bytes). The Uncompressed Packet Length is the number of bytes in the original, uncompressed packet. If this is zero, the data in this packet has not been compressed. When the compression protocol is in use, either the client or the server may compress packets. However, compression will not occur if the compressed length is greater than the original length. Thus, some packets will contain compressed data while other packets will not.</p>
<code>Use Usage Advisor, Usage Advisor, UseUsageAdvisor</code>	false	Logs inefficient database operations.
<code>Use Performance Monitor, UsePerformanceMonitor, userperfmon, perfmon</code>	false	Indicates that performance counters should be updated during execution.

Connection Pooling Options

The following table lists the valid names for options related to connection pooling within the `ConnectionString`. For more information about connection pooling, see [Section 22.2.5.4, “Using Connector/Net with Connection Pooling”](#).

Table 22.21. Connector/Net Connection String Options - Connection Pooling

Name	Default	Description
<code>Cache Server Properties, CacheServerProperties</code>	false	Specifies whether server variable settings are updated by a <code>SHOW VARIABLES</code> command each time a pooled connection is returned. Enabling this setting speeds up connections in a connection pool environment. Your application is not informed of any changes to configuration variables made by other connections. This option was added in Connector/Net 6.3.

Name	Default	Description
Connection Lifetime , ConnectionLifeTime	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by Connection Lifetime . This is useful in clustered configurations to force load balancing between a running server and a server just brought online. A value of zero (0) causes pooled connections to have the maximum connection timeout.
Connection Reset , ConnectionReset	false	If true, the connection state is reset when it is retrieved from the pool. The default value of false avoids making an additional server round trip when obtaining a connection, but the connection state is not reset.
Maximum Pool Size , Max Pool Size , maximumpoolsize	100	The maximum number of connections allowed in the pool.
Minimum Pool Size , Min Pool Size , MinimumPoolSize	0	The minimum number of connections allowed in the pool.
Pooling	true	When true , the MySQLConnection object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are true , false , yes , and no .

22.2.7. Connector/Net API Reference

This section of the manual contains a complete reference to the Connector/Net ADO.NET component, automatically generated from the embedded documentation.

22.2.7.1. [MySQL.Data.MySqlClient](#) Namespace

[Namespace hierarchy](#)

Classes

Class	Description
BaseCommandInterceptor	Provides a means of enhancing or replacing SQL commands through the connection string rather than recompiling.
BaseExceptionInterceptor	Provides a means of enabling and disabling exception handling through the connection string rather than recompiling.
MySQLClientPermission	Derived from the .NET DBDataPermission class. For usage information, see Section 22.2.5.19 , " Working with Partial Trust / Medium Trust ".
MySQLCommand	
MySQLCommandBuilder	
MySQLConnection	
MySQLDataAdapter	
MySQLDataReader	Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.

Class	Description
MySqlError	Collection of error codes that can be returned by the server
MySqlException	The exception that is thrown when MySQL returns an error. This class cannot be inherited.
MySqlHelper	Helper class that makes it easier to work with the provider.
MySqlInfoMessageEventArgs	Provides data for the InfoMessage event. This class cannot be inherited.
MySqlParameter	Represents a parameter to a MySqlCommand , and optionally, its mapping to DataSet columns. This class cannot be inherited.
MySqlParameterCollection	Represents a collection of parameters relevant to a MySqlCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.
MySqlRowUpdatedEventArgs	Provides data for the RowUpdated event. This class cannot be inherited.
MySqlRowUpdatingEventArgs	Provides data for the RowUpdating event. This class cannot be inherited.
MySqlTransaction	

Delegates

Delegate	Description
MySqlInfoMessageEventHandler	Represents the method that will handle the InfoMessage event of a MySqlConnection .
MySqlRowUpdatedEventHandler	Represents the method that will handle the RowUpdated event of a MySqlDataAdapter .
MySqlRowUpdatingEventHandler	Represents the method that will handle the RowUpdating event of a MySqlDataAdapter .

Enumerations

Enumeration	Description
MySqlDbType	Specifies MySQL-specific data type of a field, property, for use in a MySqlParameter .
MySqlErrorCode	

22.2.7.1.1. [MySql.Data.MySqlClientHierarchy](#)

See Also

[MySql.Data.MySqlClient Namespace](#)

22.2.7.1.2. [BaseCommandInterceptor](#)

The [BaseCommandInterceptor](#) class has these methods that you can override:

```
public virtual bool ExecuteScalar(string sql, ref object returnValue);
```

```
public virtual bool ExecuteNonQuery(string sql, ref int returnValue);
public virtual bool ExecuteReader(string sql, CommandBehavior behavior, ref MySqlDataReader returnValue);
public virtual void Init(MySqlConnection connection);
```

If your interceptor overrides one of the `Execute...` methods, set the `returnValue` output parameter and return `true` if you handled the event, or `false` if you did not handle the event. The SQL command is processed normally only when all command interceptors return `false`.

The connection passed to the `Init` method is the connection that is attached to this interceptor.

For full usage and examples, see [Section 22.2.5.11, "Using the Connector/Net Interceptor Classes"](#).

22.2.7.1.3. BaseExceptionInterceptor

You develop an exception interceptor first by creating a subclass of the `BaseExceptionInterceptor` class. You must override the `InterceptException()` method. You can also override the `Init()` method to do some one-time initialization.

Each exception interceptor has 2 methods:

```
public abstract Exception InterceptException(Exception exception,
    MySqlConnection connection);
public virtual void Init(MySqlConnection connection);
```

The connection passed to `Init()` is the connection that is attached to this interceptor.

Each interceptor is required to override `InterceptException` and return an exception. It can return the exception it is given, or it can wrap it in a new exception. We currently do not offer the ability to suppress the exception.

For full usage and examples, see [Section 22.2.5.11, "Using the Connector/Net Interceptor Classes"](#).

22.2.7.1.4. MySqlCommand Class

For a list of all members of this type, see [MySqlCommand Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommand_
    Inherits Component_
    Implements IDbCommand, ICloneable
```

Syntax: C#

```
public sealed class MySqlCommand : Component, IDbCommand, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: `MySql.Data` (in `MySql.Data.dll`)

See Also

[MySqlCommand Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1. MySqlCommand Members[MySqlCommand overview](#)**Public Instance Constructors**

MySqlCommand	Overloaded. Initializes a new instance of the MySqlCommand class.
------------------------------	---

Public Instance Properties

CommandText	
CommandTimeout	
CommandType	
Connection	
Container (inherited from Component)	Gets the IContainer that contains the Component.
IsPrepared	
Parameters	
Site (inherited from Component)	Gets or sets the ISite of the Component.
Transaction	
UpdatedRowSource	

Public Instance Methods

Cancel	Attempts to cancel the execution of a MySqlCommand. This operation is not supported.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
CreateParameter	Creates a new instance of a MySqlParameter object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExecuteNonQuery	
ExecuteReader	Overloaded.
ExecuteScalar	
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Prepare	

ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.
-------------------------------------	--

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
-------------------------------------	---

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1. [MySqlCommand](#) Constructor

Initializes a new instance of the [MySqlCommand](#) class.

Overload List

Initializes a new instance of the [MySqlCommand](#) class.

- [public MySqlCommand\(\);](#)
- [public MySqlCommand\(string\);](#)
- [public MySqlCommand\(string, MySqlConnection\);](#)
- [public MySqlCommand\(string, MySqlConnection, MySqlTransaction\);](#)

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.1. [MySqlCommand](#) Constructor ()

Initializes a new instance of the [MySqlCommand](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommand();
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand Constructor Overload List](#)

22.2.7.1.4.1.1.2. [MySqlCommand](#) Constructor (String)**Syntax: Visual Basic**

```
Overloads Public Sub New( _  
    ByVal cmdText As String _  
)
```

Syntax: C#

```
public MySqlCommand(  
    string cmdText  
);
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand Constructor Overload List](#)

22.2.7.1.4.1.1.3. [MySqlCommand](#) Constructor**Syntax: Visual Basic**

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection _  
)
```

Syntax: C#

```
public MySqlCommand(  
    string cmdText,  
    MySqlConnection connection  
);
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand Constructor Overload List](#)

22.2.7.1.4.1.1.3.1. [MySqlConnection](#) Class

For a list of all members of this type, see [MySqlConnection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_  
    Inherits Component_  
    Implements IDbConnection, ICloneable
```

Syntax: C#

```
public sealed class MySqlConnection : Component, IDbConnection, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlConnection Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1. [MySqlConnection](#) Members

[MySqlConnection overview](#)

Public Instance Constructors

 MySqlConnection 	Overloaded. Initializes a new instance of the MySqlConnection class.
-----------------------------------	--

Public Instance Properties

ConnectionString	
ConnectionTimeout	
Container (inherited from Component)	Gets the IContainer that contains the Component.
Database	
DataSource	Gets the name of the MySQL server to which to connect.
ServerThread	Returns the ID of the server thread this connection is executing on.
ServerVersion	
Site (inherited from Component)	Gets or sets the ISite of the Component.
State	
UseCompression	Indicates if this connection should use compression when communicating with the server.

Public Instance Methods

BeginTransaction	Overloaded.
ChangeDatabase	
Close	
CreateCommand	
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Open	
Ping	Ping
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
InfoMessage	
StateChange	

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.1. [MySQLConnection](#) Constructor

Initializes a new instance of the [MySQLConnection](#) class.

Overload List

Initializes a new instance of the [MySQLConnection](#) class.

- [public MySQLConnection\(\);](#)
- [public MySQLConnection\(string\);](#)

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.1.1. [MySQLConnection](#) Constructor

Initializes a new instance of the [MySQLConnection](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySQLConnection();
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLConnection Constructor Overload List](#)

22.2.7.1.4.1.1.3.1.1.1.2. [MySQLConnection](#) Constructor**Syntax: Visual Basic**

```
Overloads Public Sub New( _
    ByVal connectionString As String _
)
```

Syntax: C#

```
public MySQLConnection(
    string connectionString
);
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLConnection Constructor Overload List](#)

22.2.7.1.4.1.1.3.1.1.2. ConnectionString Property

Syntax: Visual Basic

```
NotOverridable Public Property ConnectionString As String _  
-  
    Implements IDbConnection.ConnectionString
```

Syntax: C#

```
public string ConnectionString {get; set;}
```

Implements

IDbConnection.ConnectionString

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.3. ConnectionTimeout Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property ConnectionTimeout As Integer _  
-  
    Implements IDbConnection.ConnectionTimeout
```

Syntax: C#

```
public int ConnectionTimeout {get;}
```

Implements

IDbConnection.ConnectionTimeout

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.4. Database Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Database As String _  
-  
    Implements IDbConnection.Database
```

Syntax: C#

```
public string Database {get;}
```

Implements

IDbConnection.Database

See Also

[MySQLConnection Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.5. DataSource Property

Gets the name of the MySQL server to which to connect.

Syntax: Visual Basic

```
Public ReadOnly Property DataSource As String
```

Syntax: C#

```
public string DataSource {get;}
```

See Also

[MySQLConnection Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.6. ServerThread Property

Returns the ID of the server thread this connection is executing on

Syntax: Visual Basic

```
Public ReadOnly Property ServerThread As Integer
```

Syntax: C#

```
public int ServerThread {get;}
```

See Also

[MySQLConnection Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.7. ServerVersion Property

Syntax: Visual Basic

```
Public ReadOnly Property ServerVersion As String
```

Syntax: C#

```
public string ServerVersion {get;}
```

See Also

[MySQLConnection Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.8. State Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property State As ConnectionState _  
_ Implements IDbConnection.State
```

Syntax: C#

```
public System.Data.ConnectionState State {get;}
```

Implements

IDbConnection.State

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.9. UseCompression Property

Indicates if this connection should use compression when communicating with the server.

Syntax: Visual Basic

```
Public ReadOnly Property UseCompression As Boolean
```

Syntax: C#

```
public bool UseCompression {get;}
```

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.10. BeginTransaction Method

Overload List

- [public MySqlConnection BeginTransaction\(\);](#)
- [public MySqlConnection BeginTransaction\(IsolationLevel\);](#)

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.10.1. MySqlConnection.BeginTransaction Method

Syntax: Visual Basic

```
Overloads Public Function BeginTransaction() As MySqlTransaction
```

Syntax: C#

```
public MySqlTransaction BeginTransaction();
```

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlConnection.BeginTransaction Overload List](#)

22.2.7.1.4.1.1.3.1.1.10.1.1. MySqlTransaction Class

For a list of all members of this type, see [MySqlTransaction Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlTransaction_  
    Implements IDbTransaction, IDisposable
```

Syntax: C#

```
public sealed class MySqlTransaction : IDbTransaction, IDisposable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlTransaction Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.10.1.1.1. [MySqlTransaction](#) Members

[MySqlTransaction overview](#)

Public Instance Properties

Connection	Gets the MySqlConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
IsolationLevel	Specifies the IsolationLevel for this transaction.

Public Instance Methods

Commit	
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Rollback	
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySqlTransaction Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.10.1.1.1.1. Connection Property

Gets the [MySqlConnection](#) object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Syntax: Visual Basic

```
Public ReadOnly Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get;}
```

Property Value

The [MySqlConnection](#) object associated with this transaction.

Remarks

A single application may have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by [BeginTransaction](#).

See Also

[MySqlTransaction Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.10.1.1.1.2. IsolationLevel Property

Specifies the IsolationLevel for this transaction.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsolationLevel As IsolationLevel _  
-  
    Implements IDbTransaction.IsolationLevel
```

Syntax: C#

```
public System.Data.IsolationLevel IsolationLevel {get;}
```

Property Value

The IsolationLevel for this transaction. The default is ReadCommitted.

Implements

IDbTransaction.IsolationLevel

Remarks

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.

See Also

[MySqlTransaction Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.10.1.1.1.3. [MySqlTransaction.Commit](#) Method

Syntax: Visual Basic

```
NotOverridable Public Sub Commit() _  
-  
    Implements IDbTransaction.Commit
```

Syntax: C#

```
public void Commit();
```

Implements

IDbTransaction.Commit

See Also

[MySQLTransaction Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.10.1.1.1.4. [MySQLTransaction.Rollback](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Sub Rollback() _  
_   
_ Implements IDbTransaction.Rollback
```

Syntax: C#

```
public void Rollback();
```

Implements

IDbTransaction.Rollback

See Also

[MySQLTransaction Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.10.2. [MySQLConnection.BeginTransaction](#) Method**Syntax: Visual Basic**

```
Overloads Public Function BeginTransaction( _  
ByVal iso As IsolationLevel _  
) As MySQLTransaction
```

Syntax: C#

```
public MySQLTransaction BeginTransaction(  
IsolationLeveliso  
);
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLConnection.BeginTransaction Overload List](#)

22.2.7.1.4.1.1.3.1.1.11. [MySQLConnection.ChangeDatabase](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Sub ChangeDatabase( _  
ByVal databaseName As String _  
) _  
_   
_ Implements IDbConnection.ChangeDatabase
```

Syntax: C#

```
public void ChangeDatabase(  
stringdatabaseName  
);
```

Implements

IDbConnection.ChangeDatabase

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.12. [MySQLConnection.Close](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Sub Close() _  
-  
    Implements IDbConnection.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDbConnection.Close

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.13. [MySQLConnection.CreateCommand](#) Method**Syntax: Visual Basic**

```
Public Function CreateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand CreateCommand();
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.14. [MySQLConnection.Open](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Sub Open() _  
-  
    Implements IDbConnection.Open
```

Syntax: C#

```
public void Open();
```

Implements

IDbConnection.Open

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.15. [MySQLConnection.Ping](#) Method

Ping

Syntax: Visual Basic

```
Public Function Ping() As Boolean
```

Syntax: C#

```
public bool Ping();
```

Return Value**See Also**

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16. [MySQLConnection.InfoMessage](#) Event**Syntax: Visual Basic**

```
Public Event InfoMessage As MySQLInfoMessageEventHandler
```

Syntax: C#

```
public event MySQLInfoMessageEventHandler InfoMessage;
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1. [MySQLInfoMessageEventHandler](#) Delegate

Represents the method that will handle the [InfoMessage](#) event of a [MySQLConnection](#).

Syntax: Visual Basic

```
Public Delegate Sub MySQLInfoMessageEventHandler( _  
    ByVal sender As Object, _  
    ByVal args As MySQLInfoMessageEventArgs _  
)
```

Syntax: C#

```
public delegate void MySQLInfoMessageEventHandler(  
    object sender,  
    MySQLInfoMessageEventArgs args  
);
```

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1.1. [MySQLInfoMessageEventArgs](#) Class

Provides data for the [InfoMessage](#) event. This class cannot be inherited.

For a list of all members of this type, see [MySQLInfoMessageEventArgs Members](#).

Syntax: Visual Basic

```
Public Class MySqlInfoMessageEventArgs_
    Inherits EventArgs
```

Syntax: C#

```
public class MySqlInfoMessageEventArgs : EventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlInfoMessageEventArgs Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1.1.1. [MySqlInfoMessageEventArgs Members](#)

[MySqlInfoMessageEventArgs overview](#)

Public Instance Constructors

MySqlInfoMessageEventArgs Constructor	Initializes a new instance of the MySqlInfoMessageEventArgs class.
---	--

Public Instance Fields

errors	
------------------------	--

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlInfoMessageEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1.1.1.1. [MySqlInfoMessageEventArgs](#) Constructor

Initializes a new instance of the [MySqlInfoMessageEventArgs](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlInfoMessageEventArgs();
```

See Also

[MySqlInfoMessageEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1.1.1.2. [MySqlInfoMessageEventArgs.errors](#) Field**Syntax: Visual Basic**

```
Public errors As MySqlError()
```

Syntax: C#

```
public MySqlError[] errors;
```

See Also

[MySqlInfoMessageEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1.1.1.2.1. [MySqlError](#) Class

Collection of error codes that can be returned by the server

For a list of all members of this type, see [MySqlError Members](#) .

Syntax: Visual Basic

```
Public Class MySqlError
```

Syntax: C#

```
public class MySqlError
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlError Members, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.1.3.1.1.16.1.1.1.2.1.1.1. [MySqlError](#) Members[MySqlError overview](#)**Public Instance Constructors**

MySqlError Constructor	
--	--

Public Instance Properties

Code	Error code
Level	Error level
Message	Error message

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also[MySqlError Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.1.3.1.1.16.1.1.1.2.1.1.1.1. [MySqlError](#) Constructor**Syntax: Visual Basic**

```
Public Sub New( _
    ByVal level As String, _
    ByVal code As Integer, _
    ByVal message As String _
)
```

Syntax: C#

```
public MySqlError(
    stringlevel,
    intcode,
    stringmessage
);
```

Parameters

- `level`:
- `code`:
- `message`:

See Also

[MySqlError Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1.1.1.2.1.1.2. Code Property

Error code

Syntax: Visual Basic

```
Public ReadOnly Property Code As Integer
```

Syntax: C#

```
public int Code {get;}
```

See Also

[MySqlError Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1.1.1.2.1.1.3. Level Property

Error level

Syntax: Visual Basic

```
Public ReadOnly Property Level As String
```

Syntax: C#

```
public string Level {get;}
```

See Also

[MySqlError Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.16.1.1.1.2.1.1.4. Message Property

Error message

Syntax: Visual Basic

```
Public ReadOnly Property Message As String
```

Syntax: C#

```
public string Message {get;}
```

See Also

[MySqlError Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.3.1.1.17. [MySQLConnection.StateChange](#) Event

Syntax: Visual Basic

```
Public Event StateChange As StateChangeEventHandler
```

Syntax: C#

```
public event StateChangeEventHandler StateChange;
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.1.4. [MySQLCommand](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySQLConnection, _  
    ByVal transaction As MySQLTransaction _  
)
```

Syntax: C#

```
public MySQLCommand(  
    stringcmdText,  
    MySQLConnectionconnection,  
    MySQLTransactiontransaction  
) ;
```

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLCommand Constructor Overload List](#)

22.2.7.1.4.1.2. [CommandText](#) Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandText As String _  
_   
_ Implements IDbCommand.CommandText
```

Syntax: C#

```
public string CommandText {get; set;}
```

Implements

[IDbCommand.CommandText](#)

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.3. [CommandTimeout](#) Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandTimeout As Integer _  
_
```

```
Implements IDbCommand.CommandTimeout
```

Syntax: C#

```
public int CommandTimeout {get; set;}
```

Implements

IDbCommand.CommandTimeout

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.4. CommandType Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandType As CommandType _  
_ Implements IDbCommand.CommandType
```

Syntax: C#

```
public System.Data.CommandType CommandType {get; set;}
```

Implements

IDbCommand.CommandType

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.5. Connection Property

Syntax: Visual Basic

```
Public Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get; set;}
```

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.6. IsPrepared Property

Syntax: Visual Basic

```
Public ReadOnly Property IsPrepared As Boolean
```

Syntax: C#

```
public bool IsPrepared {get;}
```

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7. Parameters Property

Syntax: Visual Basic

```
Public ReadOnly Property Parameters As MySqlParameterCollection
```

Syntax: C#

```
public MySqlParameterCollection Parameters {get;}
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1. [MySqlParameterCollection](#) Class

Represents a collection of parameters relevant to a [MySqlCommand](#) as well as their respective mappings to columns in a DataSet. This class cannot be inherited.

For a list of all members of this type, see [MySqlParameterCollection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameterCollection_
    Inherits MarshalByRefObject_
    Implements IDataParameterCollection, IList, ICollection, IEnumerable
```

Syntax: C#

```
public sealed class MySqlParameterCollection : MarshalByRefObject, IDataParameterCollection, IList, ICollection
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlParameterCollection Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1. [MySqlParameterCollection](#) Members

[MySqlParameterCollection overview](#)

Public Instance Constructors

MySqlParameterCollection Constructor	Initializes a new instance of the MySqlParameterCollection class.
--	---

Public Instance Properties

Count	Gets the number of MySqlParameter objects in the collection.
-----------------------	--

Item	Overloaded. Gets the MySQLParameter with a specified attribute. In C#, this property is the indexer for the MySQLParameterCollection class.
----------------------	---

Public Instance Methods

Add	Overloaded. Adds the specified MySQLParameter object to the MySQLParameterCollection .
Clear	Removes all items from the collection.
Contains	Overloaded. Gets a value indicating whether a MySQLParameter exists in the collection.
CopyTo	Copies MySQLParameter objects from the MySQLParameterCollection to the specified array.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IndexOf	Overloaded. Gets the location of a MySQLParameter in the collection.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Insert	Inserts a MySQLParameter into the collection at the specified index.
Remove	Removes the specified MySQLParameter from the collection.
RemoveAt	Overloaded. Removes the specified MySQLParameter from the collection.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.1. [MySQLParameterCollection](#) Constructor

Initializes a new instance of the [MySQLParameterCollection](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlParameterCollection();
```

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.2. Count Property

Gets the number of [MySqlParameter](#) objects in the collection.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Count As Integer _
_
_ Implements ICollection.Count
```

Syntax: C#

```
public int Count {get;}
```

Implements

[ICollection.Count](#)

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3. Item Property

Gets the [MySqlParameter](#) with a specified attribute. In C#, this property is the indexer for the [MySqlParameterCollection](#) class.

Overload List

Gets the [MySqlParameter](#) at the specified index.

- [public MySqlParameter this\[int\] {get; set;}](#)

Gets the [MySqlParameter](#) with the specified name.

- [public MySqlParameter this\[string\] {get; set;}](#)

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1. [MySqlParameter](#) Class

Represents a parameter to a [MySqlCommand](#), and optionally, its mapping to [DataSet](#) columns. This class cannot be inherited.

For a list of all members of this type, see [MySqlParameter Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameter_
_ Inherits MarshalByRefObject_
_ Implements IDataParameter, IDbDataParameter, ICloneable
```

Syntax: C#

```
public sealed class MySqlParameter : MarshalByRefObject, IDataParameter, IDbDataParameter, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlParameter Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1. [MySqlParameter](#) Members

[MySqlParameter overview](#)

Public Instance Constructors

MySqlParameter	Overloaded. Initializes a new instance of the MySqlParameter class.
--------------------------------	---

Public Instance Properties

DbType	Gets or sets the DbType of the parameter.
Direction	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySQL version 4.1 and earlier, input-only is the only valid choice.
IsNullable	Gets or sets a value indicating whether the parameter accepts null values.
IsUnsigned	
MySqlDbType	Gets or sets the MySqlDbType of the parameter.
ParameterName	Gets or sets the name of the MySqlParameter .
Precision	Gets or sets the maximum number of digits used to represent the Value property.
Scale	Gets or sets the number of decimal places to which Value is resolved.
Size	Gets or sets the maximum size, in bytes, of the data within the column.
SourceColumn	Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the Value .
SourceVersion	Gets or sets the DataRowVersion to use when loading Value .
Value	Gets or sets the value of the parameter.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ToString	Overridden. Gets a string containing the ParameterName .

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.1. [MySQLParameter](#) Constructor

Initializes a new instance of the [MySQLParameter](#) class.

Overload List

Initializes a new instance of the [MySQLParameter](#) class.

- [public MySQLParameter\(\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and the data type.

- [public MySQLParameter\(string,MySQLDbType\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySQLDbType](#), and the size.

- [public MySQLParameter\(string,MySQLDbType,int\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the type of the parameter, the size of the parameter, a [ParameterDirection](#), the precision of the parameter, the scale of the parameter, the source column, a [DataRowVersion](#) to use, and the value of the parameter.

- [public MySQLParameter\(string,MySQLDbType,int,ParameterDirection,bool,byte,byte,string,DataRowVersion,object\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySQLDbType](#), the size, and the source column name.

- [public MySQLParameter\(string,MySQLDbType,int,string\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and a value of the new [MySQLParameter](#).

- `public MySqlParameter(string,object);`

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.1.1. `MySqlParameter` Constructor ()

Initializes a new instance of the `MySqlParameter` class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlParameter();
```

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

22.2.7.1.4.1.7.1.1.3.1.1.1.2. `MySqlParameter` Constructor

Initializes a new instance of the `MySqlParameter` class with the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType _  
)
```

Syntax: C#

```
public MySqlParameter(  
    stringparameterName,  
    MySqlDbTypedbType  
);
```

Parameters

- `parameterName`: The name of the parameter to map.
- `dbType`: One of the [MySqlDbType](#) values.

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

22.2.7.1.4.1.7.1.1.3.1.1.2.1. `MySqlDbType` Enumeration

Specifies MySQL-specific data type of a field, property, for use in a `MySqlParameter`.

Syntax: Visual Basic

```
Public Enum MySqlDbType
```

Syntax: C#

```
public enum MySqlDbType
```

Members

Member Name	Description
Newdate	Obsolete. Use Datetime or Date type.
Timestamp	A timestamp. The range is '1970-01-01 00:00:01.000000' to '2038-01-19 03:14:07.999999'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Time	The range is '-838:59:59.000000' to '838:59:59.000000'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Date	The supported range is '1000-01-01' to '9999-12-31'.
Datetime	The supported range is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Year	A year in 2- or 4-digit format (default is 4-digit). The allowable values are 1901 to 2155, 0000 in the 4-digit year format, and 1970-2069 if you use the 2-digit format (70-69).
TinyBlob	A BLOB column with a maximum length of 255 (2 ⁸ - 1) characters.
Blob	A BLOB column with a maximum length of 65535 (2 ¹⁶ - 1) characters.
MediumBlob	A BLOB column with a maximum length of 16777215 (2 ²⁴ - 1) characters.
LongBlob	A BLOB column with a maximum length of 4294967295 or 4G (2 ³² - 1) characters.
Int16	A 16-bit signed integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.
Int24	Specifies a 24 (3 byte) signed or unsigned value.
Int32	A 32-bit signed integer.
Int64	A 64-bit signed integer.
Byte	The signed range is -128 to 127. The unsigned range is 0 to 255.
Float	A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.
Double	A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.
UByte	An 8-bit unsigned value.

UInt16	A 16-bit unsigned value.
UInt24	A 24-bit unsigned value.
UInt32	A 32-bit unsigned value.
UInt64	A 64-bit unsigned value.
Decimal	A fixed precision and scale numeric value between $-10^{38}-1$ and $10^{38}-1$.
NewDecimal	New Decimal
Set	A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', ... A SET can have a maximum of 64 members.
String	Obsolete. Use VarChar type.
VarChar	A variable-length string containing 0 to 255 characters.
VarString	A variable-length string containing 0 to 65535 characters.
Enum	An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', ..., NULL or the special "" error value. An ENUM can have a maximum of 65535 distinct values.
Geometry	
Bit	Bit-field data type.
TinyText	A nonbinary string column supporting a maximum length of 255 ($2^8 - 1$) characters.
Text	A nonbinary string column supporting a maximum length of 65535 ($2^{16} - 1$) characters.
MediumText	A nonbinary string column supporting a maximum length of 16777215 ($2^{24} - 1$) characters.
LongText	A nonbinary string column supporting a maximum length of 4294967295 ($2^{32} - 1$) characters.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.3. [MySqlParameter](#) Constructor (String, MySqlDbType, Int32)

Initializes a new instance of the [MySqlParameter](#) class with the parameter name, the [MySqlDbType](#), and the size.

Syntax: Visual Basic

```
Overloads Public Sub New( _
```

```

    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer _
)

```

Syntax: C#

```

public MySqlParameter(
    stringparameterName,
    MySqlDbTypeDbType,
    intsize
);

```

Parameters

- `parameterName`: The name of the parameter to map.
- `dbType`: One of the [MySqlDbType](#) values.
- `size`: The length of the parameter.

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

22.2.7.1.4.1.7.1.1.3.1.1.4. [MySqlParameter](#) Constructor

Initializes a new instance of the [MySqlParameter](#) class with the parameter name, the type of the parameter, the size of the parameter, a [ParameterDirection](#), the precision of the parameter, the scale of the parameter, the source column, a [DataRowVersion](#) to use, and the value of the parameter.

Syntax: Visual Basic

```

Overloads Public Sub New( _
    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer, _
    ByVal direction As ParameterDirection, _
    ByVal isNullable As Boolean, _
    ByVal precision As Byte, _
    ByVal scale As Byte, _
    ByVal sourceColumn As String, _
    ByVal sourceVersion As DataRowVersion, _
    ByVal value As Object _
)

```

Syntax: C#

```

public MySqlParameter(
    stringparameterName,
    MySqlDbTypeDbType,
    intsize,
    ParameterDirectiondirection,
    boolisNullable,
    byteprecision,
    bytescale,
    stringsourceColumn,
    DataRowVersionsourceVersion,
    objectvalue
);

```

Parameters

- `parameterName`: The name of the parameter to map.

- `dbType`: One of the [MySqlDbType](#) values.
- `size`: The length of the parameter.
- `direction`: One of the [ParameterDirection](#) values.
- `isNullable`: true if the value of the field can be null, otherwise false.
- `precision`: The total number of digits to the left and right of the decimal point to which [Value](#) is resolved.
- `scale`: The total number of decimal places to which [Value](#) is resolved.
- `sourceColumn`: The name of the source column.
- `sourceVersion`: One of the [DataRowVersion](#) values.
- `value`: An Object that is the value of the [MySqlParameter](#).

Exceptions

Exception Type	Condition
ArgumentException	

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

22.2.7.1.4.1.7.1.1.3.1.1.4.1. Value Property

Gets or sets the value of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property Value As Object _
-
    Implements IDataParameter.Value
```

Syntax: C#

```
public object Value {get; set;}
```

Implements

[IDataParameter.Value](#)

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.5. [MySqlParameter](#) Constructor

Initializes a new instance of the [MySqlParameter](#) class with the parameter name, the [MySqlDbType](#), the size, and the source column name.

Syntax: Visual Basic

```
Overloads Public Sub New( _
```

```

    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer, _
    ByVal sourceColumn As String _
)

```

Syntax: C#

```

public MySqlParameter(
    stringparameterName,
    MySqlDbTypedbType,
    intsize,
    stringsourceColumn
);

```

Parameters

- `parameterName`: The name of the parameter to map.
- `dbType`: One of the [MySqlDbType](#) values.
- `size`: The length of the parameter.
- `sourceColumn`: The name of the source column.

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

22.2.7.1.4.1.7.1.1.3.1.1.1.6. [MySqlParameter](#) Constructor

Initializes a new instance of the [MySqlParameter](#) class with the parameter name and a value of the new [MySqlParameter](#).

Syntax: Visual Basic

```

Overloads Public Sub New( _
    ByVal parameterName As String, _
    ByVal value As Object _
)

```

Syntax: C#

```

public MySqlParameter(
    stringparameterName,
    objectvalue
);

```

Parameters

- `parameterName`: The name of the parameter to map.
- `value`: An Object that is the value of the [MySqlParameter](#).

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

22.2.7.1.4.1.7.1.1.3.1.1.2. DbType Property

Gets or sets the DbType of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property DbType As DbType _  
-  
    Implements IDataParameter.DbType
```

Syntax: C#

```
public System.Data.DbType DbType {get; set;}
```

Implements

IDataParameter.DbType

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.3. Direction Property

Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySQL version 4.1 and earlier, input-only is the only valid choice.

Syntax: Visual Basic

```
NotOverridable Public Property Direction As ParameterDirection _  
-  
    Implements IDataParameter.Direction
```

Syntax: C#

```
public System.Data.ParameterDirection Direction {get; set;}
```

Implements

IDataParameter.Direction

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.4. IsNullable Property

Gets or sets a value indicating whether the parameter accepts null values.

Syntax: Visual Basic

```
NotOverridable Public Property IsNullable As Boolean _  
-  
    Implements IDataParameter.IsNullable
```

Syntax: C#

```
public bool IsNullable {get; set;}
```

Implements

IDataParameter.IsNullable

See Also

[MySQLParameter Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.5. IsUnsigned Property

Syntax: Visual Basic

```
Public Property IsUnsigned As Boolean
```

Syntax: C#

```
public bool IsUnsigned {get; set;}
```

See Also

[MySQLParameter Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.6. MySQLDbType Property

Gets or sets the MySQLDbType of the parameter.

Syntax: Visual Basic

```
Public Property MySQLDbType As MySQLDbType
```

Syntax: C#

```
public MySQLDbType MySQLDbType {get; set;}
```

See Also

[MySQLParameter Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.7. ParameterName Property

Gets or sets the name of the MySQLParameter.

Syntax: Visual Basic

```
NotOverridable Public Property ParameterName As String _  
_ Implements IDataParameter.ParameterName
```

Syntax: C#

```
public string ParameterName {get; set;}
```

Implements

IDataParameter.ParameterName

See Also

[MySQLParameter Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.8. Precision Property

Gets or sets the maximum number of digits used to represent the [Value](#) property.

Syntax: Visual Basic

```
NotOverridable Public Property Precision As Byte _  
-  
  Implements IDbDataParameter.Precision
```

Syntax: C#

```
public byte Precision {get; set;}
```

Implements

IDbDataParameter.Precision

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.9. Scale Property

Gets or sets the number of decimal places to which [Value](#) is resolved.

Syntax: Visual Basic

```
NotOverridable Public Property Scale As Byte _  
-  
  Implements IDbDataParameter.Scale
```

Syntax: C#

```
public byte Scale {get; set;}
```

Implements

IDbDataParameter.Scale

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.10. Size Property

Gets or sets the maximum size, in bytes, of the data within the column.

Syntax: Visual Basic

```
NotOverridable Public Property Size As Integer _  
-  
  Implements IDbDataParameter.Size
```

Syntax: C#

```
public int Size {get; set;}
```

Implements

IDbDataParameter.Size

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.11. SourceColumn Property

Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the [Value](#).

Syntax: Visual Basic

```
NotOverridable Public Property SourceColumn As String _  
-  
    Implements IDataParameter.SourceColumn
```

Syntax: C#

```
public string SourceColumn {get; set;}
```

Implements

IDataParameter.SourceColumn

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.12. SourceVersion Property

Gets or sets the DataRowVersion to use when loading [Value](#).

Syntax: Visual Basic

```
NotOverridable Public Property SourceVersion As DataRowVersion _  
-  
    Implements IDataParameter.SourceVersion
```

Syntax: C#

```
public System.Data.DataRowVersion SourceVersion {get; set;}
```

Implements

IDataParameter.SourceVersion

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.1.1.13. MySQLParameter.ToString Method

Overridden. Gets a string containing the [ParameterName](#).

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

Return Value

See Also

[MySQLParameter Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.3.2. Item Property (Int32)

Gets the [MySQLParameter](#) at the specified index.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _  
    ByVal index As Integer _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter this[  
    int index  
] {get; set;}
```

See Also

[MySQLParameterCollection Class, MySql.Data.MySqlClient Namespace, MySQLParameterCollection.Item Overload List](#)

22.2.7.1.4.1.7.1.1.3.3. Item Property (String)

Gets the [MySQLParameter](#) with the specified name.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _  
    ByVal name As String _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter this[  
    string name  
] {get; set;}
```

See Also

[MySQLParameterCollection Class, MySql.Data.MySqlClient Namespace, MySQLParameterCollection.Item Overload List](#)

22.2.7.1.4.1.7.1.1.4. Add Method

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#).

Overload List

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#).

- `public MySQLParameter Add(MySQLParameter);`

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#).

- `public int Add(object);`

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the parameter name and the data type.

- [public MySqlConnection Add\(string,MySqlDbType\);](#)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, and the column length.

- [public MySqlConnection Add\(string,MySqlDbType,int\);](#)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, the column length, and the source column name.

- [public MySqlConnection Add\(string,MySqlDbType,int,string\);](#)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) given the specified parameter name and value.

- [public MySqlConnection Add\(string,object\);](#)

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.4.1. [MySqlParameterCollection.Add](#) Method

Adds the specified [MySqlParameter](#) object to the [MySqlParameterCollection](#).

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal value As MySqlConnection _  
) As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Add(  
    MySqlConnection value  
);
```

Parameters

- **value:** The [MySqlParameter](#) to add to the collection.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameterCollection.Add Overload List](#)

22.2.7.1.4.1.7.1.1.4.2. [MySqlParameterCollection.Add](#) Method

Adds the specified [MySqlParameter](#) object to the [MySqlParameterCollection](#).

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Add( _  
    ByVal value As Object _  
) As Integer _  
—  
Implements IList.Add
```

Syntax: C#

```
public int Add(  
    object value  
);
```

Parameters

- **value**: The [MySQLParameter](#) to add to the collection.

Return Value

The index of the new [MySQLParameter](#) object.

Implements

[IList.Add](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLParameterCollection.Add Overload List](#)

22.2.7.1.4.1.7.1.1.4.3. MySQLParameterCollection.Add Method

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySQLDbType _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(  
    string parameterName,  
    MySQLDbType dbType  
);
```

Parameters

- **parameterName**: The name of the parameter.
- **dbType**: One of the [MySQLDbType](#) values.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLParameterCollection.Add Overload List](#)

22.2.7.1.4.1.7.1.1.4.4. MySQLParameterCollection.Add Method

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) with the parameter name, the data type, and the column length.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType, _  
    ByVal size As Integer _  
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(  
    stringparameterName,  
    MySqlDbTypedbType,  
    intsize  
);
```

Parameters

- `parameterName`: The name of the parameter.
- `dbType`: One of the [MySqlDbType](#) values.
- `size`: The length of the column.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameterCollection.Add Overload List](#)

22.2.7.1.4.1.7.1.1.4.5. [MySqlParameterCollection.Add](#) Method

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, the column length, and the source column name.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType, _  
    ByVal size As Integer, _  
    ByVal sourceColumn As String _  
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(  
    stringparameterName,  
    MySqlDbTypedbType,  
    intsize,  
    stringsourceColumn  
);
```

Parameters

- `parameterName`: The name of the parameter.
- `dbType`: One of the [MySqlDbType](#) values.

- `size`: The length of the column.
- `sourceColumn`: The name of the source column.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLParameterCollection.Add Overload List](#)

22.2.7.1.4.1.7.1.1.4.6. [MySQLParameterCollection.Add](#) Method

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the specified parameter name and value.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal value As Object _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(  
    stringparameterName,  
    objectvalue  
);
```

Parameters

- `parameterName`: The name of the parameter.
- `value`: The [Value](#) of the [MySQLParameter](#) to add to the collection.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLParameterCollection.Add Overload List](#)

22.2.7.1.4.1.7.1.1.5. [MySQLParameterCollection.Clear](#) Method

Removes all items from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Clear() _  
-  
    Implements IList.Clear
```

Syntax: C#

```
public void Clear();
```

Implements

IList.Clear

See Also[MySQLParameterCollection Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.6. Contains Method

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

Overload List

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

- `public bool Contains(object);`

Gets a value indicating whether a [MySQLParameter](#) with the specified parameter name exists in the collection.

- `public bool Contains(string);`

See Also[MySQLParameterCollection Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.7.1.1.6.1. [MySQLParameterCollection.Contains](#) Method

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _  
    ByVal value As Object _  
) As Boolean _  
-  
    Implements IList.Contains
```

Syntax: C#

```
public bool Contains(  
    objectvalue  
);
```

Parameters

- `value`: The value of the [MySQLParameter](#) object to find.

Return Value

true if the collection contains the [MySQLParameter](#) object; otherwise, false.

Implements

IList.Contains

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.Contains Overload List](#)

22.2.7.1.4.1.7.1.1.6.2. [MySQLParameterCollection.Contains](#) Method

Gets a value indicating whether a [MySQLParameter](#) with the specified parameter name exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _  
    ByVal name As String _  
) As Boolean _  
-  
    Implements IDataParameterCollection.Contains
```

Syntax: C#

```
public bool Contains(  
    stringname  
);
```

Parameters

- **name**: The name of the [MySQLParameter](#) object to find.

Return Value

true if the collection contains the parameter; otherwise, false.

Implements

[IDataParameterCollection.Contains](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.Contains Overload List](#)

22.2.7.1.4.1.7.1.1.7. [MySQLParameterCollection.CopyTo](#) Method

Copies [MySQLParameter](#) objects from the [MySQLParameterCollection](#) to the specified array.

Syntax: Visual Basic

```
NotOverridable Public Sub CopyTo( _  
    ByVal array As Array, _  
    ByVal index As Integer _  
) _  
-  
    Implements ICollection.CopyTo
```

Syntax: C#

```
public void CopyTo(  
    Arrayarray,  
    intindex  
);
```

Parameters

- [array](#):
- [index \[839\]](#):

Implements

[ICollection.CopyTo](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.8. IndexOf Method

Gets the location of a [MySQLParameter](#) in the collection.

Overload List

Gets the location of a [MySQLParameter](#) in the collection.

- `public int IndexOf(object);`

Gets the location of the [MySQLParameter](#) in the collection with a specific parameter name.

- `public int IndexOf(string);`

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.8.1. `MySQLParameterCollection.IndexOf` Method

Gets the location of a [MySQLParameter](#) in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _  
    ByVal value As Object _  
) As Integer _  
—  
Implements IList.IndexOf
```

Syntax: C#

```
public int IndexOf(  
    object value  
);
```

Parameters

- `value`: The [MySQLParameter](#) object to locate.

Return Value

The zero-based location of the [MySQLParameter](#) in the collection.

Implements

[IList.IndexOf](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.IndexOf Overload List](#)

22.2.7.1.4.1.7.1.1.8.2. [MySQLParameterCollection.IndexOf](#) Method

Gets the location of the [MySQLParameter](#) in the collection with a specific parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _  
    ByVal parameterName As String _  
) As Integer _  
-  
    Implements IDataParameterCollection.IndexOf
```

Syntax: C#

```
public int IndexOf(  
    string parameterName  
);
```

Parameters

- `parameterName`: The name of the [MySQLParameter](#) object to retrieve.

Return Value

The zero-based location of the [MySQLParameter](#) in the collection.

Implements

[IDataParameterCollection.IndexOf](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.IndexOf Overload List](#)

22.2.7.1.4.1.7.1.1.9. [MySQLParameterCollection.Insert](#) Method

Inserts a [MySQLParameter](#) into the collection at the specified index.

Syntax: Visual Basic

```
NotOverridable Public Sub Insert( _  
    ByVal index As Integer, _  
    ByVal value As Object _  
) _  
-  
    Implements IList.Insert
```

Syntax: C#

```
public void Insert(  
    int index,  
    object value  
);
```

Parameters

- [index \[839\]](#):
- [value](#):

Implements

[IList.Insert](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.10. [MySQLParameterCollection.Remove](#) Method

Removes the specified [MySQLParameter](#) from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Remove( _  
    ByVal value As Object _  
) _  
—  
Implements IList.Remove
```

Syntax: C#

```
public void Remove(  
    objectvalue  
);
```

Parameters

- [value](#):

Implements

[IList.Remove](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.11. [RemoveAt](#) Method

Removes the specified [MySQLParameter](#) from the collection.

Overload List

Removes the specified [MySQLParameter](#) from the collection using a specific index.

- [public void RemoveAt\(int\);](#)

Removes the specified [MySQLParameter](#) from the collection using the parameter name.

- [public void RemoveAt\(string\);](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.7.1.1.11.1. [MySQLParameterCollection.RemoveAt](#) Method

Removes the specified [MySQLParameter](#) from the collection using a specific index.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _  
    ByVal index As Integer _  
) _  
-  
    Implements IList.RemoveAt
```

Syntax: C#

```
public void RemoveAt(  
    int index  
);
```

Parameters

- [index](#) [839]: The zero-based index of the parameter.

Implements

IList.RemoveAt

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.RemoveAt Overload List](#)

22.2.7.1.4.1.7.1.1.11.2. [MySQLParameterCollection.RemoveAt](#) Method

Removes the specified [MySQLParameter](#) from the collection using the parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _  
    ByVal name As String _  
) _  
-  
    Implements IDataParameterCollection.RemoveAt
```

Syntax: C#

```
public void RemoveAt(  
    string name  
);
```

Parameters

- [name](#): The name of the [MySQLParameter](#) object to retrieve.

Implements

IDataParameterCollection.RemoveAt

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.RemoveAt Overload List](#)

22.2.7.1.4.1.8. Transaction Property

Syntax: Visual Basic

```
Public Property Transaction As MySqlTransaction
```

Syntax: C#

```
public MySqlTransaction Transaction {get; set;}
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.9. UpdatedRowSource Property

Syntax: Visual Basic

```
NotOverridable Public Property UpdatedRowSource As UpdateRowSource _  
-  
Implements IDbCommand.UpdatedRowSource
```

Syntax: C#

```
public System.Data.UpdateRowSource UpdatedRowSource {get; set;}
```

Implements

IDbCommand.UpdatedRowSource

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.10. MySqlCommand.Cancel Method

Attempts to cancel the execution of a MySqlCommand. This operation is not supported.

Syntax: Visual Basic

```
NotOverridable Public Sub Cancel() _  
-  
Implements IDbCommand.Cancel
```

Syntax: C#

```
public void Cancel();
```

Implements

IDbCommand.Cancel

Remarks

Cancelling an executing command is currently not supported on any version of MySQL.

Exceptions

Exception Type	Condition
----------------	-----------

NotSupportedException	This operation is not supported.
-----------------------	----------------------------------

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.11. [MySqlCommand.CreateParameter](#) Method

Creates a new instance of a [MySqlParameter](#) object.

Syntax: Visual Basic

```
Public Function CreateParameter() As MySqlParameter
```

Syntax: C#

```
public MySqlParameter CreateParameter();
```

Return Value

A [MySqlParameter](#) object.

Remarks

This method is a strongly-typed version of `CreateParameter`.

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.12. [MySqlCommand.ExecuteNonQuery](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function ExecuteNonQuery() As Integer _  
—  
Implements IDbCommand.ExecuteNonQuery
```

Syntax: C#

```
public int ExecuteNonQuery();
```

Implements

`IDbCommand.ExecuteNonQuery`

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13. `ExecuteReader` Method**Overload List**

- [public MySqlDataReader ExecuteReader\(\)](#);
- [public MySqlDataReader ExecuteReader\(CommandBehavior\)](#);

See Also

[MySqlCommand Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.13.1. [MySqlCommand.ExecuteReader](#) Method**Syntax: Visual Basic**

```
Overloads Public Function ExecuteReader() As MySqlDataReader
```

Syntax: C#

```
public MySqlDataReader ExecuteReader();
```

See Also

[MySqlCommand Class, MySql.Data.MySqlClient Namespace, MySqlCommand.ExecuteReader Overload List](#)

22.2.7.1.4.1.13.1.1. [MySqlDataReader](#) Class

Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.

For a list of all members of this type, see [MySqlDataReader Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataReader_
    Inherits MarshalByRefObject_
    Implements IEnumerable, IDataReader, IDisposable, IDataRecord
```

Syntax: C#

```
public sealed class MySqlDataReader : MarshalByRefObject, IEnumerable, IDataReader, IDisposable, IDataRecord
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDataReader Members, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1. [MySqlDataReader](#) Members

[MySqlDataReader overview](#)

Public Instance Properties

 Depth	Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.
------------------------	--

FieldCount	Gets the number of columns in the current row.
HasRows	Gets a value indicating whether the <code>MySqlDataReader</code> contains one or more rows.
IsClosed	Gets a value indicating whether the data reader is closed.
Item	Overloaded. Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the <code>MySqlDataReader</code> class.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Public Instance Methods

Close	Closes the <code>MySqlDataReader</code> object.
CreateObjRef (inherited from <code>MarshalByRefObject</code>)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from <code>Object</code>)	Determines whether the specified <code>Object</code> is equal to the current <code>Object</code> .
GetBoolean	Gets the value of the specified column as a Boolean.
GetByte	Gets the value of the specified column as a byte.
GetBytes	Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.
GetChar	Gets the value of the specified column as a single character.
GetChars	Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.
GetDataTypeName	Gets the name of the source data type.
GetDateTime	
GetDecimal	
GetDouble	
GetFieldType	Gets the <code>Type</code> that is the data type of the object.
GetFloat	
GetGuid	
GetHashCode (inherited from <code>Object</code>)	Serves as a hash function for a particular type. <code>GetHashCode</code> is suitable for use in hashing algorithms and data structures like a hash table.
GetInt16	
GetInt32	
GetInt64	
GetLifetimeService (inherited from <code>MarshalByRefObject</code>)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.

GetMySqlDateTime	
GetName	Gets the name of the specified column.
GetOrdinal	Gets the column ordinal, given the name of the column.
GetSchemaTable	Returns a DataTable that describes the column metadata of the MySqlDataReader.
GetString	
GetTimeSpan	
GetType (inherited from Object)	Gets the Type of the current instance.
GetUInt16	
GetUInt32	
GetUInt64	
GetValue	Gets the value of the specified column in its native format.
GetValues	Gets all attribute columns in the collection for the current row.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
IsDBNull	Gets a value indicating whether the column contains non-existent or missing values.
NextResult	Advances the data reader to the next result, when reading the results of batch SQL statements.
Read	Advances the MySqlDataReader to the next record.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.1. Depth Property

Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Depth As Integer _
-
    Implements IDataReader.Depth
```

Syntax: C#

```
public int Depth {get;}
```

Implements

IDataReader.Depth

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.2. FieldCount Property

Gets the number of columns in the current row.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property FieldCount As Integer _  
-  
    Implements IDataRecord.FieldCount
```

Syntax: C#

```
public int FieldCount {get;}
```

Implements

IDataRecord.FieldCount

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.3. HasRows Property

Gets a value indicating whether the MySQLDataReader contains one or more rows.

Syntax: Visual Basic

```
Public ReadOnly Property HasRows As Boolean
```

Syntax: C#

```
public bool HasRows {get;}
```

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.4. IsClosed Property

Gets a value indicating whether the data reader is closed.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsClosed As Boolean _  
-  
    Implements IDataReader.IsClosed
```

Syntax: C#

```
public bool IsClosed {get;}
```

Implements

IDataReader.IsClosed

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.5. Item Property

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Overload List

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

- `public object this[int] {get;}`

Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

- `public object this[string] {get;}`

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.5.1. Item Property (Int32)

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _  
    ByVal i As Integer _  
) _  
-  
    Implements IDataRecord.Item As Object _  
-  
    Implements IDataRecord.Item
```

Syntax: C#

```
public object this[  
    inti  
] {get;}
```

Implements

`IDataRecord.Item`

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlDataReader.Item Overload List](#)

22.2.7.1.4.1.13.1.1.1.5.2. Item Property (String)

Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _  
    ByVal name As String _  
) _  
-
```

```
Implements IDataRecord.Item As Object _  
-  
Implements IDataRecord.Item
```

Syntax: C#

```
public object this[  
stringname  
] {get;}
```

Implements

IDataRecord.Item

See Also[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLDataReader.Item Overload List](#)

22.2.7.1.4.1.13.1.1.1.6. RecordsAffected Property

Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property RecordsAffected As Integer _  
-  
Implements IDataReader.RecordsAffected
```

Syntax: C#

```
public int RecordsAffected {get;}
```

Implements

IDataReader.RecordsAffected

See Also[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.7. MySQLDataReader.Close Method

Closes the MySQLDataReader object.

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _  
-  
Implements IDataReader.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDataReader.Close

See Also[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.8. [MySqlDataReader.GetBoolean](#) Method

Gets the value of the specified column as a Boolean.

Syntax: Visual Basic

```
NotOverridable Public Function GetBoolean( _  
    ByVal i As Integer _  
) As Boolean _  
—  
    Implements IDataRecord.GetBoolean
```

Syntax: C#

```
public bool GetBoolean(  
    inti  
);
```

Parameters

- *i*:

Return Value

Implements

IDataRecord.GetBoolean

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.9. [MySqlDataReader.GetByte](#) Method

Gets the value of the specified column as a byte.

Syntax: Visual Basic

```
NotOverridable Public Function GetByte( _  
    ByVal i As Integer _  
) As Byte _  
—  
    Implements IDataRecord.GetByte
```

Syntax: C#

```
public byte GetByte(  
    inti  
);
```

Parameters

- *i*:

Return Value

Implements

IDataRecord.GetByte

See Also

[MySqlDataReader Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.13.1.1.1.10. [MySqlDataReader.GetBytes](#) Method

Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetBytes( _  
    ByVal i As Integer, _  
    ByVal dataIndex As Long, _  
    ByVal buffer As Byte(), _  
    ByVal bufferSize As Integer, _  
    ByVal length As Integer _  
    ) As Long _  
-  
    Implements IDataRecord.GetBytes
```

Syntax: C#

```
public long GetBytes(  
    inti,  
    longdataIndex,  
    byte[]buffer,  
    intbufferIndex,  
    intlength  
    );
```

Parameters

- **i**: The zero-based column ordinal.
- **dataIndex**: The index within the field from which to begin the read operation.
- **buffer**: The buffer into which to read the stream of bytes.
- **bufferIndex**: The index for buffer to begin the read operation.
- **length**: The maximum length to copy into the buffer.

Return Value

The actual number of bytes read.

Implements

IDataRecord.GetBytes

See Also

[MySqlDataReader Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.11. [MySqlDataReader.GetChar](#) Method

Gets the value of the specified column as a single character.

Syntax: Visual Basic

```
NotOverridable Public Function GetChar( _  
    ByVal i As Integer _
```

```
) As Char _  
-  
Implements IDataRecord.GetChar
```

Syntax: C#

```
public char GetChar(  
    inti  
);
```

Parameters

- `i`:

Return Value**Implements**

IDataRecord.GetChar

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.12. [MySqlDataReader.GetChars](#) Method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetChars( _  
    ByVal i As Integer, _  
    ByVal fieldOffset As Long, _  
    ByVal buffer As Char(), _  
    ByVal bufferoffset As Integer, _  
    ByVal length As Integer _  
    ) As Long _  
-  
Implements IDataRecord.GetChars
```

Syntax: C#

```
public long GetChars(  
    inti,  
    longfieldOffset,  
    char[]buffer,  
    intbufferoffset,  
    intlength  
);
```

Parameters

- `i`:
- `fieldOffset`:
- `buffer`:
- `bufferoffset`:
- `length`:

Return Value**Implements**

IDataRecord.GetChars

See Also[MySQLDataReader Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.13.1.1.1.13. [MySQLDataReader.GetDataTypeName](#) Method

Gets the name of the source data type.

Syntax: Visual Basic

```
NotOverridable Public Function GetDataTypeName( _  
    ByVal i As Integer _  
) As String _  
-  
    Implements IDataRecord.GetDataTypeName
```

Syntax: C#

```
public string GetDataTypeName(  
    inti  
);
```

Parameters

- `i`:

Return Value**Implements**

IDataRecord.GetDataTypeName

See Also[MySQLDataReader Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.13.1.1.1.14. [MySQLDataReader.GetDateTime](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetDateTime( _  
    ByVal index As Integer _  
) As Date _  
-  
    Implements IDataRecord.GetDateTime
```

Syntax: C#

```
public DateTime GetDateTime(  
    int index  
);
```

Implements

IDataRecord.GetDateTime

See Also

[MySqlDataReader Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.15. [MySqlDataReader.GetDecimal](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetDecimal( _  
    ByVal index As Integer _  
) As Decimal _  
-  
    Implements IDataRecord.GetDecimal
```

Syntax: C#

```
public decimal GetDecimal(  
    int index  
);
```

Implements

IDataRecord.GetDecimal

See Also

[MySqlDataReader Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.16. [MySqlDataReader.GetDouble](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetDouble( _  
    ByVal index As Integer _  
) As Double _  
-  
    Implements IDataRecord.GetDouble
```

Syntax: C#

```
public double GetDouble(  
    int index  
);
```

Implements

IDataRecord.GetDouble

See Also

[MySqlDataReader Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.17. [MySqlDataReader.GetFieldType](#) Method

Gets the Type that is the data type of the object.

Syntax: Visual Basic

```
NotOverridable Public Function GetFieldType( _  
    ByVal i As Integer _  
) As Type _  
-  
    Implements IDataRecord.GetFieldType
```

Syntax: C#

```
public Type GetFieldType(  
    int i  
);
```

Parameters

- `i`:

Return Value**Implements**

IDataRecord.GetFieldType

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.18. [MySQLDataReader.GetFloat](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetFloat( _  
    ByVal index As Integer _  
) As Single _  
—  
    Implements IDataRecord.GetFloat
```

Syntax: C#

```
public float GetFloat(  
    int index  
);
```

Implements

IDataRecord.GetFloat

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.19. [MySQLDataReader.GetGuid](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetGuid( _  
    ByVal index As Integer _  
) As Guid _  
—  
    Implements IDataRecord.GetGuid
```

Syntax: C#

```
public Guid GetGuid(  
    int index  
);
```

Implements

IDataRecord.GetGuid

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.20. [MySqlDataReader.GetInt16](#) Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt16( _  
    ByVal index As Integer _  
) As Short _  
-  
    Implements IDataRecord.GetInt16
```

Syntax: C#

```
public short GetInt16(  
    int index  
);
```

Implements

IDataRecord.GetInt16

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.21. [MySqlDataReader.GetInt32](#) Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt32( _  
    ByVal index As Integer _  
) As Integer _  
-  
    Implements IDataRecord.GetInt32
```

Syntax: C#

```
public int GetInt32(  
    int index  
);
```

Implements

IDataRecord.GetInt32

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.22. [MySqlDataReader.GetInt64](#) Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt64( _  
    ByVal index As Integer _  
) As Long _  
-  
    Implements IDataRecord.GetInt64
```

Syntax: C#

```
public long GetInt64(  
    int index  
);
```

Implements

IDataRecord.GetInt64

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.23. [MySQLDataReader.GetMySqlDateTime](#) Method**Syntax: Visual Basic**

```
Public Function GetMySqlDateTime( _  
    ByVal index As Integer _  
) As MySqlDateTime
```

Syntax: C#

```
public MySqlDateTime GetMySqlDateTime(  
    int index  
);
```

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.24. [MySQLDataReader.GetName](#) Method

Gets the name of the specified column.

Syntax: Visual Basic

```
NotOverridable Public Function GetName( _  
    ByVal i As Integer _  
) As String _  
—  
    Implements IDataRecord.GetName
```

Syntax: C#

```
public string GetName(  
    int i  
);
```

Parameters

- `i`:

Return Value**Implements**

IDataRecord.GetName

See Also

[MySqlDataReader Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.13.1.1.1.25. [MySqlDataReader.GetOrdinal](#) Method

Gets the column ordinal, given the name of the column.

Syntax: Visual Basic

```
NotOverridable Public Function GetOrdinal( _  
    ByVal name As String _  
) As Integer _  
-  
    Implements IDataRecord.GetOrdinal
```

Syntax: C#

```
public int GetOrdinal(  
    stringname  
);
```

Parameters

- `name`:

Return Value**Implements**

`IDataRecord.GetOrdinal`

See Also[MySqlDataReader Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.13.1.1.1.26. [MySqlDataReader.GetSchemaTable](#) Method

Returns a `DataTable` that describes the column metadata of the `MySqlDataReader`.

Syntax: Visual Basic

```
NotOverridable Public Function GetSchemaTable() As DataTable _  
-  
    Implements IDataReader.GetSchemaTable
```

Syntax: C#

```
public DataTable GetSchemaTable();
```

Return Value**Implements**

`IDataReader.GetSchemaTable`

See Also[MySqlDataReader Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.13.1.1.1.27. [MySqlDataReader.GetString](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetString( _  
    ByVal index As Integer _  
    ) As String _  
-  
    Implements IDataRecord.GetString
```

Syntax: C#

```
public string GetString(  
    int index  
);
```

Implements

IDataRecord.GetString

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.28. [MySQLDataReader.GetString](#) Method**Syntax: Visual Basic**

```
Public Function GetString( _  
    ByVal index As Integer _  
    ) As TimeSpan
```

Syntax: C#

```
public TimeSpan GetString(  
    int index  
);
```

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.29. [MySQLDataReader.GetUInt16](#) Method**Syntax: Visual Basic**

```
Public Function GetUInt16( _  
    ByVal index As Integer _  
    ) As UInt16
```

Syntax: C#

```
public ushort GetUInt16(  
    int index  
);
```

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.30. [MySQLDataReader.GetUInt32](#) Method**Syntax: Visual Basic**

```
Public Function GetUInt32( _  
    ByVal index As Integer _
```

```
) As UInt32
```

Syntax: C#

```
public uint GetUInt32(  
    int index  
);
```

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.31. [MySqlDataReader.GetUInt64](#) Method**Syntax: Visual Basic**

```
Public Function GetUInt64( _  
    ByVal index As Integer _  
) As UInt64
```

Syntax: C#

```
public ulong GetUInt64(  
    int index  
);
```

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.32. [MySqlDataReader.GetValue](#) Method

Gets the value of the specified column in its native format.

Syntax: Visual Basic

```
NotOverridable Public Function GetValue( _  
    ByVal i As Integer _  
) As Object _  
—  
    Implements IDataRecord.GetValue
```

Syntax: C#

```
public object GetValue(  
    int i  
);
```

Parameters

- *i*:

Return Value**Implements**

[IDataRecord.GetValue](#)

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.33. [MySQLDataReader](#).[GetValues](#) Method

Gets all attribute columns in the collection for the current row.

Syntax: Visual Basic

```
NotOverridable Public Function GetValues( _  
    ByVal values As Object() _  
) As Integer _  
-  
    Implements IDataRecord.GetValues
```

Syntax: C#

```
public int GetValues(  
    object[] values  
);
```

Parameters

- `values`:

Return Value

Implements

`IDataRecord.GetValues`

See Also

[MySQLDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.34. [MySQLDataReader](#).[IsDBNull](#) Method

Gets a value indicating whether the column contains non-existent or missing values.

Syntax: Visual Basic

```
NotOverridable Public Function IsDBNull( _  
    ByVal i As Integer _  
) As Boolean _  
-  
    Implements IDataRecord.IsDBNull
```

Syntax: C#

```
public bool IsDBNull(  
    inti  
);
```

Parameters

- `i`:

Return Value

Implements

`IDataRecord.IsDBNull`

See Also

[MySQLDataReader Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.4.1.13.1.1.1.35. [MySQLDataReader.NextResult](#) Method

Advances the data reader to the next result, when reading the results of batch SQL statements.

Syntax: Visual Basic

```
NotOverridable Public Function NextResult() As Boolean _  
-  
    Implements IDataReader.NextResult
```

Syntax: C#

```
public bool NextResult();
```

Return Value**Implements**

IDataReader.NextResult

See Also

[MySQLDataReader Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.1.1.1.36. [MySQLDataReader.Read](#) Method

Advances the MySQLDataReader to the next record.

Syntax: Visual Basic

```
NotOverridable Public Function Read() As Boolean _  
-  
    Implements IDataReader.Read
```

Syntax: C#

```
public bool Read();
```

Return Value**Implements**

IDataReader.Read

See Also

[MySQLDataReader Class, MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.13.2. [MySQLCommand.ExecuteReader](#) Method**Syntax: Visual Basic**

```
Overloads Public Function ExecuteReader( _  
    ByVal behavior As CommandBehavior _  
) As MySQLDataReader
```

Syntax: C#

```
public MySQLDataReader ExecuteReader(  
    CommandBehaviorbehavior  
);
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand.ExecuteReader Overload List](#)

22.2.7.1.4.1.14. [MySqlCommand.ExecuteScalar](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function ExecuteScalar() As Object _  
    _  
    Implements IDbCommand.ExecuteScalar
```

Syntax: C#

```
public object ExecuteScalar();
```

Implements

IDbCommand.ExecuteScalar

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.4.1.15. [MySqlCommand.Prepare](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Sub Prepare() _  
    _  
    Implements IDbCommand.Prepare
```

Syntax: C#

```
public void Prepare();
```

Implements

IDbCommand.Prepare

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5. [MySqlCommandBuilder](#) Class

For a list of all members of this type, see [MySqlCommandBuilder Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommandBuilder_  
    Inherits Component
```

Syntax: C#

```
public sealed class MySqlCommandBuilder : Component
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlCommandBuilder Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1. MySqlCommandBuilder Members

[MySqlCommandBuilder overview](#)

Public Static (Shared) Methods

DeriveParameters	Overloaded. Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.
----------------------------------	--

Public Instance Constructors

MySqlCommandBuilder	Overloaded. Initializes a new instance of the MySqlCommandBuilder class.
-------------------------------------	--

Public Instance Properties

Container (inherited from Component)	Gets the IContainer that contains the Component.
DataAdapter	
QuotePrefix	
QuoteSuffix	
Site (inherited from Component)	Gets or sets the ISite of the Component.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetDeleteCommand	
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetInsertCommand	
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.

GetType (inherited from Object)	Gets the Type of the current instance.
GetUpdateCommand	
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
RefreshSchema	
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
-------------------------------------	---

See Also

[MySQLCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.1. DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

Overload List

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

- [public static void DeriveParameters\(MySqlCommand\);](#)
- [public static void DeriveParameters\(MySqlCommand,bool\);](#)

See Also

[MySQLCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.1.1. [MySQLCommandBuilder.DeriveParameters](#) Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _
    ByVal command As MySqlCommand _
)
```

Syntax: C#

```
public static void DeriveParameters(
    MySqlCommand command
);
```

Parameters

- `command`: The MySqlCommand referencing the stored procedure from which the parameter information is to be derived. The derived parameters are added to the Parameters collection of the MySqlCommand.

Exceptions

Exception Type	Condition
InvalidOperationException	The command text is not a valid stored procedure name.

See Also

[MySqlCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySqlCommandBuilder.DeriveParameters Overload List](#)

22.2.7.1.5.1.1.2. MySqlCommandBuilder.DeriveParameters Method

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _
    ByVal command As MySqlCommand, _
    ByVal useProc As Boolean _
)
```

Syntax: C#

```
public static void DeriveParameters(
    MySqlCommand command,
    bool useProc
);
```

See Also

[MySqlCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySqlCommandBuilder.DeriveParameters Overload List](#)

22.2.7.1.5.1.2. MySqlCommandBuilder Constructor

Initializes a new instance of the [MySqlCommandBuilder](#) class.

Overload List

Initializes a new instance of the [MySqlCommandBuilder](#) class.

- `public MySqlCommandBuilder();`
- `public MySqlCommandBuilder(MySqlDataAdapter);`
- `public MySqlCommandBuilder(MySqlDataAdapter, bool);`
- `public MySqlCommandBuilder(bool);`

See Also

[MySqlCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.1. MySqlCommandBuilder Constructor

Initializes a new instance of the [MySqlCommandBuilder](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommandBuilder();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

22.2.7.1.5.1.2.2. MySqlCommandBuilder Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal adapter As MySqlDataAdapter _  
)
```

Syntax: C#

```
public MySqlCommandBuilder(  
    MySqlDataAdapter adapter  
);
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

22.2.7.1.5.1.2.2.1. MySqlDataAdapter Class

For a list of all members of this type, see [MySqlDataAdapter Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataAdapter_  
    Inherits DbDataAdapter
```

Syntax: C#

```
public sealed class MySqlDataAdapter : DbDataAdapter
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDataAdapter Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1. MySqlDataAdapter Members

[MySqlDataAdapter overview](#)

Public Instance Constructors

MySqlDataAdapter	Overloaded. Initializes a new instance of the <code>MySqlDataAdapter</code> class.
----------------------------------	--

Public Instance Properties

<code>AcceptChangesDuringFill</code> (inherited from <code>DataAdapter</code>)	Gets or sets a value indicating whether <code>AcceptChanges</code> is called on a <code>DataRow</code> after it is added to the <code>DataTable</code> during any of the Fill operations.
<code>AcceptChangesDuringUpdate</code> (inherited from <code>DataAdapter</code>)	Gets or sets whether <code>AcceptChanges</code> is called during a <code>Update</code> .
<code>Container</code> (inherited from <code>Component</code>)	Gets the <code>IContainer</code> that contains the <code>Component</code> .
<code>ContinueUpdateOnError</code> (inherited from <code>DataAdapter</code>)	Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.
DeleteCommand	Overloaded.
<code>FillLoadOption</code> (inherited from <code>DataAdapter</code>)	Gets or sets the <code>LoadOption</code> that determines how the adapter fills the <code>DataTable</code> from the <code>DbDataReader</code> .
InsertCommand	Overloaded.
<code>MissingMappingAction</code> (inherited from <code>DataAdapter</code>)	Determines the action to take when incoming data does not have a matching table or column.
<code>MissingSchemaAction</code> (inherited from <code>DataAdapter</code>)	Determines the action to take when existing <code>DataSetschema</code> does not match incoming data.
<code>ReturnProviderSpecificTypes</code> (inherited from <code>DataAdapter</code>)	Gets or sets whether the <code>FillMethod</code> should return provider-specific values or common CLS-compliant values.
SelectCommand	Overloaded.
<code>Site</code> (inherited from <code>Component</code>)	Gets or sets the <code>ISite</code> of the <code>Component</code> .
<code>TableMappings</code> (inherited from <code>DataAdapter</code>)	Gets a collection that provides the master mapping between a source table and a <code>DataTable</code> .
<code>UpdateBatchSize</code> (inherited from <code>DbDataAdapter</code>)	Gets or sets a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
UpdateCommand	Overloaded.

Public Instance Methods

<code>CreateObjRef</code> (inherited from <code>MarshalByRefObject</code>)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
<code>Dispose</code> (inherited from <code>Component</code>)	Releases all resources used by the <code>Component</code> .
<code>Equals</code> (inherited from <code>Object</code>)	Determines whether the specified <code>Object</code> is equal to the current <code>Object</code> .
<code>Fill</code> (inherited from <code>DbDataAdapter</code>)	Overloaded. Adds or refreshes rows in the <code>DataSet</code> to match those in the data source using

	the DataSetname, and creates a DataTablenamed "Table."
FillSchema (inherited from DbDataAdapter)	Overloaded. Configures the schema of the specified DataTablebased on the specified SchemaType.
GetFillParameters (inherited from DbDataAdapter)	Gets the parameters set by the user when executing an SQL SELECT statement.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ResetFillLoadOption (inherited from DataAdapter)	Resets FillLoadOptionto its default state and causes Fillto honor AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill (inherited from DataAdapter)	Determines whether the AcceptChangesDuringFillproperty should be persisted.
ShouldSerializeFillLoadOption (inherited from DataAdapter)	Determines whether the FillLoadOptionproperty should be persisted.
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.
Update (inherited from DbDataAdapter)	Overloaded. Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
FillError (inherited from DataAdapter)	Returned when an error occurs during a fill operation.
RowUpdated	Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.
RowUpdating	Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Protected Internal Instance Properties

FillCommandBehavior (inherited from DbDataAdapter)	Gets or sets the behavior of the command used to fill the data adapter.
--	---

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.1. [MySQLDataAdapter](#) Constructor

Initializes a new instance of the [MySQLDataAdapter](#) class.

Overload List

Initializes a new instance of the [MySQLDataAdapter](#) class.

- [public MySQLDataAdapter\(\);](#)
- [public MySQLDataAdapter\(MySqlCommand\);](#)
- [public MySQLDataAdapter\(string, MySqlConnection\);](#)
- [public MySQLDataAdapter\(string, string\);](#)

See Also

[MySQLDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.1.1. [MySQLDataAdapter](#) Constructor

Initializes a new instance of the [MySQLDataAdapter](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySQLDataAdapter();
```

See Also

[MySQLDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLDataAdapter Constructor Overload List](#)

22.2.7.1.5.1.2.2.1.1.1.2. [MySQLDataAdapter](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommand As MySqlCommand _  
)
```

Syntax: C#

```
public MySQLDataAdapter(  
    MySqlCommand selectCommand  
) ;
```

See Also

[MySQLDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLDataAdapter Constructor Overload List](#)

22.2.7.1.5.1.2.2.1.1.1.3. [MySQLDataAdapter](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommandText As String, _
```

```
    ByVal connection As MySqlConnection _
  )
```

Syntax: C#

```
public MySqlDataAdapter(
    stringselectCommandText,
    MySqlConnectionconnection
);
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlDataAdapter Constructor Overload List](#)

22.2.7.1.5.1.2.2.1.1.4. [MySqlDataAdapter](#) Constructor**Syntax: Visual Basic**

```
Overloads Public Sub New( _
    ByVal selectCommandText As String, _
    ByVal selectConnString As String _
)
```

Syntax: C#

```
public MySqlDataAdapter(
    stringselectCommandText,
    stringselectConnString
);
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlDataAdapter Constructor Overload List](#)

22.2.7.1.5.1.2.2.1.1.2. DeleteCommand Property

Syntax: Visual Basic

```
Overloads Public Property DeleteCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand DeleteCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.3. InsertCommand Property

Syntax: Visual Basic

```
Overloads Public Property InsertCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand InsertCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.4. SelectCommand Property

Syntax: Visual Basic

```
Overloads Public Property SelectCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand SelectCommand {get; set;}
```

See Also

[MySqlCommandAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.5. UpdateCommand Property

Syntax: Visual Basic

```
Overloads Public Property UpdateCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand UpdateCommand {get; set;}
```

See Also

[MySqlCommandAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.6. MySqlCommandAdapter.RowUpdated Event

Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdated As MySqlRowUpdatedEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatedEventHandler RowUpdated;
```

Event Data

The event handler receives an argument of type [MySqlRowUpdatedEventArgs](#) containing data related to this event. The following [MySqlRowUpdatedEventArgsproperties](#) provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand executed when Update is called.
Errors	Gets any errors generated by the .NET Framework data provider when the Commandwas executed.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row	Gets the DataRow sent through an Update.
RowCount	Gets the number of rows processed in a batch of updated records.

StatementType	Gets the type of SQL statement executed.
Status	Gets the UpdateStatus of the Commandproperty.
TableMapping	Gets the DataTableMappingsent through an Update.

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.6.1. [MySqlRowUpdatedEventHandler](#) Delegate

Represents the method that will handle the RowUpdatedevent of a [MySqlDataAdapter](#).

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatedEventHandler( _
    ByVal sender As Object, _
    ByVal e As MySqlRowUpdatedEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatedEventHandler(
    objectsender,
    MySqlRowUpdatedEventArgs
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.6.1.1. [MySqlRowUpdatedEventArgs](#) Class

Provides data for the RowUpdated event. This class cannot be inherited.

For a list of all members of this type, see [MySqlRowUpdatedEventArgs Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatedEventArgs_
    Inherits RowUpdatedEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatedEventArgs : RowUpdatedEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlRowUpdatedEventArgs Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.6.1.1.1. [MySqlRowUpdatedEventArgs](#) Members

[MySqlRowUpdatedEventArgs](#) overview

Public Instance Constructors

MySqlRowUpdatedEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatedEventArgs class.
--	---

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand executed when Update is called.
Errors (inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command was executed.
RecordsAffected (inherited from RowUpdatedEventArgs)	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row (inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an Update .
RowCount (inherited from RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.
StatementType (inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status (inherited from RowUpdatedEventArgs)	Gets the UpdateStatus of the Command property.
TableMapping (inherited from RowUpdatedEventArgs)	Gets the DataTableMapping sent through an Update .

Public Instance Methods

CopyToRows (inherited from RowUpdatedEventArgs)	Overloaded. Copies references to the modified rows into the provided array.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object .
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object .

See Also

[MySqlRowUpdatedEventArgs](#) Class, [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.6.1.1.1.1. [MySqlRowUpdatedEventArgs](#) Constructor

Initializes a new instance of the [MySqlRowUpdatedEventArgs](#) class.

Syntax: Visual Basic

```
Public Sub New( _
    ByVal row As DataRow, _
    ByVal command As IDbCommand, _
    ByVal statementType As StatementType, _
    ByVal tableMapping As DataTableMapping _
)
```

Syntax: C#

```
public MySqlRowUpdatedEventArgs(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
);
```

Parameters

- **row**: The DataRow sent through an Update.
- **command**: The IDbCommand executed when Update is called.
- **statementType**: One of the StatementType values that specifies the type of query executed.
- **tableMapping**: The DataTableMapping sent through an Update.

See Also

[MySqlRowUpdatedEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.6.1.1.1.2. Command Property

Gets or sets the MySqlCommand executed when Update is called.

Syntax: Visual Basic

```
Overloads Public ReadOnly Property Command As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand Command {get;}
```

See Also

[MySqlRowUpdatedEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.7. [MySqlDataAdapter.RowUpdating](#) Event

Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdating As MySqlRowUpdatingEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatingEventHandler RowUpdating;
```

Event Data

The event handler receives an argument of type [MySQLRowUpdatingEventArgs](#) containing data related to this event. The following [MySQLRowUpdatingEventArgsproperties](#) provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand to execute when performing the Update.
Errors	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType	Gets the type of SQL statement to execute.
Status	Gets or sets the UpdateStatus of the Commandproperty.
TableMapping	Gets the DataTableMapping to send through the Update.

See Also

[MySQLDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.7.1. [MySQLRowUpdatingEventHandler](#) Delegate

Represents the method that will handle the RowUpdatingevent of a [MySQLDataAdapter](#).

Syntax: Visual Basic

```
Public Delegate Sub MySQLRowUpdatingEventHandler( _
    ByVal sender As Object, _
    ByVal e As MySQLRowUpdatingEventArgs _
)
```

Syntax: C#

```
public delegate void MySQLRowUpdatingEventHandler(
    objectsender,
    MySQLRowUpdatingEventArgs
);
```

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.7.1.1. [MySQLRowUpdatingEventArgs](#) Class

Provides data for the RowUpdating event. This class cannot be inherited.

For a list of all members of this type, see [MySQLRowUpdatingEventArgs Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatingEventArgs_
    Inherits RowUpdatingEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatingEventArgs : RowUpdatingEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlRowUpdatingEventArgs Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.2.2.1.1.7.1.1.1. [MySqlRowUpdatingEventArgs](#) Members

[MySqlRowUpdatingEventArgs](#) overview

Public Instance Constructors

MySqlRowUpdatingEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatingEventArgs class.
---	--

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand to execute when performing the Update.
Errors (inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row (inherited from RowUpdatingEventArgs)	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType (inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status (inherited from RowUpdatingEventArgs)	Gets or sets the UpdateStatus of the Commandproperty.
TableMapping (inherited from RowUpdatingEventArgs)	Gets the DataTableMapping to send through the Update.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object .
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.


```
Overloads Public Sub New( _  
    ByVal adapter As MySqlDataAdapter, _  
    ByVal lastOneWins As Boolean _  
)
```

Syntax: C#

```
public MySqlCommandBuilder(  
    MySqlDataAdapter adapter,  
    boollastOneWins  
) ;
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

22.2.7.1.5.1.2.4. [MySqlCommandBuilder](#) Constructor**Syntax: Visual Basic**

```
Overloads Public Sub New( _  
    ByVal lastOneWins As Boolean _  
)
```

Syntax: C#

```
public MySqlCommandBuilder(  
    boollastOneWins  
) ;
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

22.2.7.1.5.1.3. [DataAdapter](#) Property**Syntax: Visual Basic**

```
Public Property DataAdapter As MySqlDataAdapter
```

Syntax: C#

```
public MySqlDataAdapter DataAdapter {get; set;}
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.4. [QuotePrefix](#) Property**Syntax: Visual Basic**

```
Public Property QuotePrefix As String
```

Syntax: C#

```
public string QuotePrefix {get; set;}
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.5. QuoteSuffix Property

Syntax: Visual Basic

```
Public Property QuoteSuffix As String
```

Syntax: C#

```
public string QuoteSuffix {get; set;}
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.6. MySqlCommandBuilder.GetDeleteCommand Method

Syntax: Visual Basic

```
Public Function GetDeleteCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetDeleteCommand();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.7. MySqlCommandBuilder.GetInsertCommand Method

Syntax: Visual Basic

```
Public Function GetInsertCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetInsertCommand();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.8. MySqlCommandBuilder.GetUpdateCommand Method

Syntax: Visual Basic

```
Public Function GetUpdateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetUpdateCommand();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.5.1.9. MySqlCommandBuilder.RefreshSchema Method

Syntax: Visual Basic

```
Public Sub RefreshSchema()
```

Syntax: C#

```
public void RefreshSchema();
```

See Also

[MySQLCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.6. MySqlCommandException Class

The exception that is thrown when MySQL returns an error. This class cannot be inherited.

For a list of all members of this type, see [MySqlCommandException Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommandException_
    Inherits SystemException
```

Syntax: C#

```
public sealed class MySqlCommandException : SystemException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySqlCommandException Members](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.6.1. MySqlCommandException Members

[MySqlCommandException overview](#)

Public Instance Properties

Data (inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exceptioninstance that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.
Number	Gets a number that identifies the type of error. This number corresponds to the error numbers given in Section C.3, "Server Error Codes and Messages" .

Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Exception)	Gets the runtime type of the current instance.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

See Also

[MySqlException Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.6.1.1. Number Property

Gets a number that identifies the type of error.

Syntax: Visual Basic

```
Public ReadOnly Property Number As Integer
```

Syntax: C#

```
public int Number {get;}
```

See Also

[MySqlException Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.7. MySqlHelper Class

Helper class that makes it easier to work with the provider.

For a list of all members of this type, see [MySqlHelper Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlHelper
```

Syntax: C#

```
public sealed class MySqlHelper
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlHelper Members](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.7.1. [MySqlHelper](#) Members

[MySqlHelper overview](#)

Public Static (Shared) Methods

ExecuteDataRow	Executes a single SQL statement and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteDataset	Overloaded. Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteNonQuery	Overloaded. Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.
ExecuteReader	Overloaded. Executes a single command against a MySQL database.
ExecuteScalar	Overloaded. Execute a single command against a MySQL database.
UpdateDataSet	Updates the given table with data from the given DataSet

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLHelper Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.7.1.1. [MySQLHelper.ExecuteDataRow](#) Method

Executes a single SQL statement and returns the first row of the resultset. A new [MySQLConnection](#) object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Public Shared Function ExecuteDataRow( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray parms As MySqlParameter() _  
) As DataRow
```

Syntax: C#

```
public static DataRow ExecuteDataRow(  
    stringconnectionString,  
    stringcommandText,  
    params MySqlParameter[]parms  
);
```

Parameters

- [connectionString](#): Settings to be used for the connection
- [commandText](#): Command to execute
- [parms](#): Parameters to use for the command

Return Value

[DataRow](#) containing the first row of the resultset

See Also[MySQLHelper Class, MySql.Data.MySqlClient Namespace](#)22.2.7.1.7.1.2. [ExecuteDataset](#) Method

Executes a single SQL statement and returns the resultset in a [DataSet](#). The state of the [MySQLConnection](#) object remains unchanged after execution of this method.

Overload List

Executes a single SQL statement and returns the resultset in a [DataSet](#). The state of the [MySQLConnection](#) object remains unchanged after execution of this method.

- [public static DataSet ExecuteDataset\(MySqlConnection,string\);](#)

Executes a single SQL statement and returns the resultset in a [DataSet](#). The state of the [MySQLConnection](#) object remains unchanged after execution of this method.

- [public static DataSet ExecuteDataset\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Executes a single SQL statement and returns the resultset in a [DataSet](#). A new [MySQLConnection](#) object is created, opened, and closed during this method.

- [public static DataSet ExecuteDataset\(string,string\);](#)

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- [public static DataSet ExecuteDataset\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.7.1.2.1. [MySqlHelper.ExecuteDataset](#) Method

Executes a single SQL statement and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    MySqlConnectionconnection,  
    stringcommandText  
);
```

Parameters

- `connection`: [MySqlConnection](#) object to use
- `commandText`: Command to execute

Return Value

DataSetcontaining the resultset

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteDataset Overload List](#)

22.2.7.1.7.1.2.2. [MySqlHelper.ExecuteDataset](#) Method

Executes a single SQL statement and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    MySqlConnectionconnection,  
    stringcommandText,  
    params MySqlParameter[]commandParameters  
);
```

Parameters

- `connection`: [MySQLConnection](#) object to use
- `commandText`: Command to execute
- `commandParameters`: Parameters to use for the command

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLHelper.ExecuteDataset Overload List](#)

22.2.7.1.7.1.2.3. MySQLHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. A new MySQLConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connectionString As String, _  
    ByVal commandText As String _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    string connectionString,  
    string commandText  
);
```

Parameters

- `connectionString`: Settings to be used for the connection
- `commandText`: Command to execute

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLHelper.ExecuteDataset Overload List](#)

22.2.7.1.7.1.2.4. MySQLHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. A new MySQLConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
    string connectionString,
    string commandText,
    params MySqlParameter[] commandParameters
);
```

Parameters

- `connectionString`: Settings to be used for the connection
- `commandText`: Command to execute
- `commandParameters`: Parameters to use for the command

Return Value

DataSet containing the resultset

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteDataset Overload List](#)

22.2.7.1.7.1.3. ExecuteNonQuery Method

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

Overload List

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

- [public static int ExecuteNonQuery\(MySqlConnection, string, params MySqlParameter\[\]\)](#);

Executes a single command against a MySQL database. A new [MySqlConnection](#) is created using the [ConnectionString](#) given.

- [public static int ExecuteNonQuery\(string, string, params MySqlParameter\[\]\)](#);

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.7.1.3.1. [MySqlHelper.ExecuteNonQuery](#) Method

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _
    ByVal connection As MySqlConnection, _
    ByVal commandText As String, _
    ParamArray commandParameters As MySqlParameter() _
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(
```

```

MySQLConnection connection,
string commandText,
    params MySqlParameter[] commandParameters
);

```

Parameters

- `connection`: [MySQLConnection](#) object to use
- `commandText`: SQL statement to be executed
- `commandParameters`: Array of [MySqlParameter](#) objects to use with the command.

Return Value

See Also

[MySQLHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLHelper.ExecuteNonQuery Overload List](#)

22.2.7.1.7.1.3.2. [MySQLHelper.ExecuteNonQuery](#) Method

Executes a single command against a MySQL database. A new [MySQLConnection](#) is created using the [ConnectionString](#) given.

Syntax: Visual Basic

```

Overloads Public Shared Function ExecuteNonQuery( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ParamArray parms As MySqlParameter() _
) As Integer

```

Syntax: C#

```

public static int ExecuteNonQuery(
    string connectionString,
    string commandText,
    params MySqlParameter[] parms
);

```

Parameters

- `connectionString`: [ConnectionString](#) to use
- `commandText`: SQL statement to be executed
- `parms`: Array of [MySqlParameter](#) objects to use with the command.

Return Value

See Also

[MySQLHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLHelper.ExecuteNonQuery Overload List](#)

22.2.7.1.7.1.4. [ExecuteReader](#) Method

Executes a single command against a MySQL database.

Overload List

Executes a single command against a MySQL database.

- [public static MySqlDataReader ExecuteReader\(string,string\);](#)

Executes a single command against a MySQL database.

- [public static MySqlDataReader ExecuteReader\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.7.1.4.1. [MySqlHelper.ExecuteReader](#) Method

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _  
    ByVal connectionString As String, _  
    ByVal commandText As String _  
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(  
stringconnectionString,  
stringcommandText  
);
```

Parameters

- [connectionString](#): Settings to use for this command
- [commandText](#): Command text to use

Return Value

[MySqlDataReader](#) object ready to read the results of the command

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteReader Overload List](#)

22.2.7.1.7.1.4.2. [MySqlHelper.ExecuteReader](#) Method

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(  
stringconnectionString,  
stringcommandText,  
params MySqlParameter[]commandParameters  
);
```

Parameters

- `connectionString`: Settings to use for this command
- `commandText`: Command text to use
- `commandParameters`: Array of [MySqlParameter](#) objects to use with the command

Return Value

[MySqlDataReader](#) object ready to read the results of the command

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteReader Overload List](#)

22.2.7.1.7.1.5. ExecuteScalar Method

Execute a single command against a MySQL database.

Overload List

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(MySqlConnection,string\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(string,string\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

22.2.7.1.7.1.5.1. [MySqlHelper.ExecuteScalar](#) Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
    MySqlConnectionconnection,  
    stringcommandText  
);
```

Parameters

- `connection`: [MySqlConnection](#) object to use

- `commandText`: Command text to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteScalar Overload List](#)

22.2.7.1.7.1.5.2. `MySqlHelper.ExecuteScalar` Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
    MySqlConnection connection,  
    string commandText,  
    params MySqlParameter[] commandParameters  
);
```

Parameters

- `connection`: [MySqlConnection](#) object to use
- `commandText`: Command text to use for the command
- `commandParameters`: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteScalar Overload List](#)

22.2.7.1.7.1.5.3. `MySqlHelper.ExecuteScalar` Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connectionString As String, _  
    ByVal commandText As String _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
    string connectionString,  
    string commandText
```

```
);
```

Parameters

- `connectionString`: Settings to use for the update
- `commandText`: Command text to use for the update

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteScalar Overload List](#)

22.2.7.1.7.1.5.4. `MySqlHelper.ExecuteScalar` Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ParamArray commandParameters As MySqlParameter() _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
    string connectionString,
    string commandText,
    params MySqlParameter[] commandParameters
);
```

Parameters

- `connectionString`: Settings to use for the command
- `commandText`: Command text to use for the command
- `commandParameters`: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteScalar Overload List](#)

22.2.7.1.7.1.6. `MySqlHelper.UpdateDataSet` Method

Updates the given table with data from the given DataSet

Syntax: Visual Basic

```
Public Shared Sub UpdateDataSet( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ByVal ds As DataSet, _
```

```
    ByVal tablename As String _
)
```

Syntax: C#

```
public static void UpdateDataSet(
    stringconnectionString,
    stringcommandText,
    DataSetds,
    stringtablename
);
```

Parameters

- `connectionString`: Settings to use for the update
- `commandText`: Command text to use for the update
- `ds`: DataSetcontaining the new data to use in the update
- `tablename`: Tablename in the data set to update

See Also

[MySQLHelper Class](#), [MySQL.Data.MySqlClient Namespace](#)

22.2.7.1.8. MySQLErrorCode Enumeration**Syntax: Visual Basic**

```
Public Enum MySQLErrorCode
```

Syntax: C#

```
public enum MySQLErrorCode
```

Members

Member Name	Description
PacketTooLarge	
PasswordNotAllowed	
DuplicateKeyEntry	
HostNotPrivileged	
PasswordNoMatch	
AnonymousUser	
DuplicateKey	
KeyNotFound	
DuplicateKeyName	

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

22.2.7.2. [MySql.Data.Types Namespace](#)

[Namespace hierarchy](#)

Classes

Class	Description
MySqlConnectionException	Summary description for MySqlConnectionException.
MySqlDateTime	Summary description for MySqlDateTime.
MySqlValue	

22.2.7.2.1. [MySql.Data.TypesHierarchy](#)

See Also

[MySql.Data.Types Namespace](#)

22.2.7.2.2. [MySqlConnectionException Class](#)

Summary description for MySqlConnectionException.

For a list of all members of this type, see [MySqlConnectionException Members](#) .

Syntax: Visual Basic

```
Public Class MySqlConnectionException_
    Inherits ApplicationException
```

Syntax: C#

```
public class MySqlConnectionException : ApplicationException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.Types](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlConnectionException Members](#), [MySql.Data.Types Namespace](#)

22.2.7.2.2.1. [MySqlConnectionException Members](#)

[MySqlConnectionException overview](#)

Public Instance Constructors

MySqlConnectionException Constructor	Ctor
--	------

Public Instance Properties

Data (inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the ExceptionInstance that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Exception)	Gets the runtime type of the current instance.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT , a coded numeric value that is assigned to a specific exception.
------------------------------------	--

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySQLConversionException Class](#), [MySQL.Data.Types Namespace](#)

22.2.7.2.2.1.1. [MySQLConversionException](#) Constructor

Syntax: Visual Basic

```
Public Sub New( _
    ByVal msg As String _
)
```

Syntax: C#

```
public MySQLConversionException(
    stringmsg
);
```

See Also

[MySQLConversionException Class](#), [MySQL.Data.Types Namespace](#)

22.2.7.2.3. [MySQLDateTime](#) Class

Summary description for MySQLDateTime.

For a list of all members of this type, see [MySQLDateTime Members](#) .

Syntax: Visual Basic

```
Public Class MySQLDateTime_
    Inherits MySQLValue_
    Implements IConvertible, IComparable
```

Syntax: C#

```
public class MySQLDateTime : MySQLValue, IConvertible, IComparable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.Types](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLDateTime Members](#), [MySQL.Data.Types Namespace](#)

22.2.7.2.3.1. [MySQLDateTime](#) Members

[MySQLDateTime overview](#)

Public Static (Shared) Type Conversions

Explicit MySQLDateTime to DateTime Conversion	
---	--

Public Instance Properties

Day	Returns the day portion of this datetime
---------------------	--

Hour	Returns the hour portion of this datetime
IsNull (inherited from MySqlValue)	
IsValidDateTime	Indicates if this object contains a value that can be represented as a DateTime
Minute	Returns the minute portion of this datetime
Month	Returns the month portion of this datetime
Second	Returns the second portion of this datetime
Millisecond	Returns the millisecond portion of this datetime
ValueAsObject (inherited from MySqlValue)	Returns the value of this field as an object
Year	Returns the year portion of this datetime

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object .
GetDateTime	Returns this value as a DateTime
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Returns a MySQL-specific string representation of this value

Protected Instance Fields

classType (inherited from MySqlValue)	The system type represented by this value
dbType (inherited from MySqlValue)	The generic dbtype of this value
isNull (inherited from MySqlValue)	Is this value null
mysqlDbType (inherited from MySqlValue)	The specific MySQL db type
mysqlTypeName (inherited from MySqlValue)	The MySQL-specific typename of this value
objectValue (inherited from MySqlValue)	

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object .

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.1. [MySqlDateTime](#) Explicit [MySqlDateTime](#) to [DateTime](#) Conversion**Syntax: Visual Basic**

```
MySqlDateTime.op_Explicit(val)
```

Syntax: C#

```
public static explicit operator DateTime(  
    MySqlDateTime val  
);
```

Parameters

- `val`:

Return Value**See Also**

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.2. Day Property

Returns the day portion of this datetime

Syntax: Visual Basic

```
Public Property Day As Integer
```

Syntax: C#

```
public int Day {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.3. Hour Property

Returns the hour portion of this datetime

Syntax: Visual Basic

```
Public Property Hour As Integer
```

Syntax: C#

```
public int Hour {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4. IsNull Property

Syntax: Visual Basic

```
Public Property IsNull As Boolean
```

Syntax: C#

```
public bool IsNull {get; set;}
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1. [MySqlValue](#) Class

For a list of all members of this type, see [MySqlValue Members](#) .

Syntax: Visual Basic

```
MustInherit Public Class MySqlValue
```

Syntax: C#

```
public abstract class MySqlValue
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.Types](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlValue Members](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1. [MySqlValue](#) Members

[MySqlValue overview](#)

Protected Static (Shared) Fields

numberFormat	
------------------------------	--

Public Instance Constructors

MySqlValue Constructor	Initializes a new instance of the MySqlValue class.
--	---

Public Instance Properties

IsNull	
ValueAsObject	Returns the value of this field as an object

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Returns a string representation of this value

Protected Instance Fields

classType	The system type represented by this value
---------------------------	---

dbType	The generic dbtype of this value
isNull	Is this value null
mysqlDbType	The specific MySQL db type
mysqlTypeName	The MySQL-specific typename of this value
objectValue	

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.1. [MySqlValue.numberFormat](#) Field

Syntax: Visual Basic

```
Protected Shared numberFormat As NumberFormatInfo
```

Syntax: C#

```
protected static NumberFormatInfo numberFormat;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.2. [MySqlValue](#) Constructor

Initializes a new instance of the [MySqlValue](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlValue();
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.3. [ValueAsObject](#) Property

Returns the value of this field as an object

Syntax: Visual Basic

```
Public ReadOnly Property ValueAsObject As Object
```

Syntax: C#

```
public object ValueAsObject {get;}
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.4. [MySqlValue.ToString](#) Method

Returns a string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.5. [MySqlValue.classType](#) Field

The system type represented by this value

Syntax: Visual Basic

```
Protected classType As Type
```

Syntax: C#

```
protected Type classType;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.6. [MySqlValue.dbType](#) Field

The generic dbtype of this value

Syntax: Visual Basic

```
Protected dbType As DbType
```

Syntax: C#

```
protected DbType dbType;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.7. [MySqlValue.mysqlDbType](#) Field

The specific MySQL db type

Syntax: Visual Basic

```
Protected mysqlDbType As MySqlDbType
```

Syntax: C#

```
protected MySqlDbType mySqlDbType;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.8. [MySqlValue.mySqlTypeName](#) Field

The MySQL-specific typename of this value

Syntax: Visual Basic

```
Protected mySqlTypeName As String
```

Syntax: C#

```
protected string mySqlTypeName;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.4.1.1.9. [MySqlValue.objectValue](#) Field**Syntax: Visual Basic**

```
Protected objectValue As Object
```

Syntax: C#

```
protected object objectValue;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.5. [IsValidDateTime](#) Property

Indicates if this object contains a value that can be represented as a DateTime

Syntax: Visual Basic

```
Public ReadOnly Property IsValidDateTime As Boolean
```

Syntax: C#

```
public bool IsValidDateTime {get;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.6. [Millisecond](#) Property

Returns the millisecond portion of this datetime

Syntax: Visual Basic

```
Public Property Millisecond As Integer
```

Syntax: C#

```
public int Millisecond {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

22.2.7.2.3.1.7. Minute Property

Returns the minute portion of this datetime

Syntax: Visual Basic

```
Public Property Minute As Integer
```

Syntax: C#

```
public int Minute {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

22.2.7.2.3.1.8. Month Property

Returns the month portion of this datetime

Syntax: Visual Basic

```
Public Property Month As Integer
```

Syntax: C#

```
public int Month {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

22.2.7.2.3.1.9. Second Property

Returns the second portion of this datetime

Syntax: Visual Basic

```
Public Property Second As Integer
```

Syntax: C#

```
public int Second {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

22.2.7.2.3.1.10. Year Property

Returns the year portion of this datetime

Syntax: Visual Basic

```
Public Property Year As Integer
```

Syntax: C#

```
public int Year {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.11. [MySqlDateTime.GetDateTime](#) Method

Returns this value as a DateTime

Syntax: Visual Basic

```
Public Function GetDateTime() As Date
```

Syntax: C#

```
public DateTime GetDateTime();
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

22.2.7.2.3.1.12. [MySqlDateTime.ToString](#) Method

Returns a MySQL-specific string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

22.2.8. Connector/Net Support

The developers of Connector/Net greatly value the input of our users in the software development process. If you find Connector/Net lacking some feature important to you, or if you discover a bug and need to file a bug report, please use the instructions in [Section 1.7, "How to Report Bugs or Problems"](#).

22.2.8.1. Connector/Net Community Support

- Community support for Connector/Net can be found through the forums at <http://forums.mysql.com>.
- Community support for Connector/Net can also be found through the mailing lists at <http://lists.mysql.com>.
- Paid support is available from Oracle. Additional information is available at <http://dev.mysql.com/support/>.

22.2.8.2. How to Report Connector/Net Problems or Bugs

If you encounter difficulties or problems with Connector/Net, contact the Connector/Net community, as explained in [Section 22.2.8.1, "Connector/Net Community Support"](#).

First try to execute the same SQL statements and commands from the `mysql` client program or from `admindemo`. This helps you determine whether the error is in Connector/Net or MySQL.

If reporting a problem, ideally include the following information with the email:

- Operating system and version.
- Connector/Net version.
- MySQL server version.
- Copies of error messages or other unexpected output.
- Simple reproducible sample.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through <http://bugs.mysql.com/>.

22.2.9. Connector/Net FAQ

Questions

- [23.2.9.1: \[2902\]](#) How do I obtain the value of an auto-incremented column?

Questions and Answers

23.2.9.1: How do I obtain the value of an auto-incremented column?

When using `CommandBuilder`, setting `ReturnGeneratedIdentifiers` property to `true` no longer works, as `CommandBuilder` does not add `last_insert_id()` by default.

`CommandBuilder` hooks up to the `DataAdapter.RowUpdating` event handler, which means it will get called for every row. It examines the command object and, if it is the same referenced object, it essentially rebuilds the object, thereby destroying your command text changes.

One approach to solving this problem is to clone the command object so you have a different actual reference:

```
dataAdapter.InsertCommand = cb.GetInsertCommand().Clone()
```

This will work, but since the `CommandBuilder` is still connected to the `DataAdapter`, the `RowUpdating` event will still fire and performance will be hit. To stop that, once all your commands have been added you need to disconnect the `CommandBuilder` from the `DataAdapter`:

```
cb.DataAdapter = null;
```

The last requirement is to make sure the `id` that is returned by `last_insert_id()` has the correct name. For example:

```
SELECT last_insert_id() AS id
```

A complete working example is shown here:

```
using System;  
using System.Collections.Generic;  
using System.Linq;
```

```
using System.Text;

using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace GetAutoIncId
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=TestDB;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                string sql = "SELECT * FROM TestTable";

                MySqlDataAdapter da = new MySqlDataAdapter(sql, conn);
                MySqlCommandBuilder cb = new MySqlCommandBuilder(da);

                MySqlCommand cmd = new MySqlCommand();
                cmd.Connection = conn;
                cmd.CommandText = sql;
                // use Cloned object to avoid .NET rebuilding the object, and
                // thereby throwing away our command text additions.
                MySqlCommand insertCmd = cb.GetInsertCommand().Clone();
                insertCmd.CommandText = insertCmd.CommandText + ";SELECT last_insert_id() AS id";
                insertCmd.UpdatedRowSource = UpdateRowSource.FirstReturnedRecord;
                da.InsertCommand = insertCmd;
                cb.DataAdapter = null; // Unhook RowUpdating event handler

                DataTable dt = new DataTable();
                da.Fill(dt);

                DataRow row = dt.NewRow();
                row["name"] = "Joe Smith";

                dt.Rows.Add(row);
                da.Update(dt);

                System.Console.WriteLine("ID after update: " + row["id"]);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }

            conn.Close();
            Console.WriteLine("Done.");
        }
    }
}
```

22.3. MySQL Connector/J

This section explains how to configure and develop Java applications with MySQL Connector/J, the JDBC driver that is integrated with MySQL.

For release notes detailing the changes in each release of Connector/J, see [MySQL Connector/J Release Notes](#).

22.3.1. Overview of MySQL Connector/J

MySQL provides connectivity for client applications developed in the Java programming language through a JDBC driver, which is called MySQL Connector/J.

MySQL Connector/J is a JDBC Type 4 driver. Different versions are available that are compatible with the JDBC 3.0 and JDBC 4.0 specifications. The Type 4 designation means that the driver is a pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries.

For large-scale programs that use common design patterns of data access, consider using one of the popular persistence frameworks such as [Hibernate](#), [Spring's JDBC templates](#) or [Ibatis SQL Maps](#) to reduce the amount of JDBC code for you to debug, tune, secure, and maintain.

Key Topics

- For help with connection strings, connection options, and setting up your connection through JDBC, see [Section 22.3.5.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”](#).

22.3.2. Connector/J Versions

There are currently four versions of MySQL Connector/J available:

- Connector/J 5.1 is the Type 4 pure Java JDBC driver, which conforms to the JDBC 3.0 and JDBC 4.0 specifications. It provides compatibility with all the functionality of MySQL, including 4.1, 5.0, 5.1, 5.4 and 5.5. Connector/J 5.1 provides ease of development features, including auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for the JDBC-4.0 XML processing, per connection client information, [NCHAR](#), [NVARCHAR](#) and [NCLOB](#) types. This release also includes all bug fixes up to and including Connector/J 5.0.6.
- Connector/J 5.0 provides support for all the functionality offered by Connector/J 3.1 and includes distributed transaction (XA) support.
- Connector/J 3.1 was designed for connectivity to MySQL 4.1 and MySQL 5.0 servers and provides support for all the functionality in MySQL 5.0 except distributed transaction (XA) support.
- Connector/J 3.0 provides core functionality and was designed for connectivity to MySQL 3.x or MySQL 4.1 servers, although it provides basic compatibility with later versions of MySQL. Connector/J 3.0 does not support server-side prepared statements, and does not support any of the features in versions of MySQL later than 4.1.

The following table summarizes the Connector/J versions available, along with the details of JDBC driver type, what version of the JDBC API it supports, what versions of MySQL Server it works with, and whether it is currently supported or not:

Table 22.22. Summary of Connector/J Versions

Connector/J version	Driver Type	JDBC version	MySQL Server version	Status
5.1	4	3.0, 4.0	4.1, 5.0, 5.1, 5.4, 5.5	Recommended version
5.0	4	3.0	4.1, 5.0	Released version
3.1	4	3.0	4.1, 5.0	Obsolete

Connector/J version	Driver Type	JDBC version	MySQL Server version	Status
3.0	4	3.0	3.x, 4.1	Obsolete

The current recommended version for Connector/J is 5.1. This guide covers all four connector versions, with specific notes given where a setting applies to a specific option.

22.3.2.1. Java Versions Supported

The following table summarizes what version of Java RTE is required to use Connector/J with Java applications, and what version of JDK is required to build Connector/J source code:

Table 22.23. Summary of Java Versions Required by Connector/J

Connector/J version	Java RTE required	JDK required (to build source code)
5.1	1.5.x, 1.6.x	1.6.x and 1.5.x
5.0	1.3.x, 1.4.x, 1.5.x, 1.6.x	1.4.2, 1.5.x, 1.6.x
3.1	1.2.x, 1.3.x, 1.4.x, 1.5.x, 1.6.x	1.4.2, 1.5.x, 1.6.x
3.0	1.2.x, 1.3.x, 1.4.x, 1.5.x, 1.6.x	1.4.2, 1.5.x, 1.6.x

If you are building Connector/J from source code using the source distribution (see [Section 22.3.3.4, “Installing from the Development Source Tree”](#)), you must use JDK 1.4.2 or newer to compile the Connector package. For Connector/J 5.1, you must have both JDK-1.6.x and JDK-1.5.x installed to be able to build the source code.

Because of the implementation of `java.sql.Savepoint`, Connector/J 3.1.0 and newer will not run on a Java runtime older than 1.4 unless the class verifier is turned off (by setting the `-Xverify:none` option to the Java runtime). This is because the class verifier will try to load the class definition for `java.sql.Savepoint` even though it is not accessed by the driver unless you actually use `savepoint` functionality.

Caching functionality provided by Connector/J 3.1.0 or newer is also not available on JVMs older than 1.4.x, as it relies on `java.util.LinkedHashMap` which was first available in JDK-1.4.0.

MySQL Connector/J does not support JDK-1.1.x or JDK-1.0.x.

22.3.3. Connector/J Installation

You can install the Connector/J package using either the binary or source distribution. The binary distribution provides the easiest method for installation; the source distribution lets you customize your installation further. With either solution, you manually add the Connector/J location to your Java `CLASSPATH`.

If you are upgrading from a previous version, read the upgrade information in [Section 22.3.3.3, “Upgrading from an Older Version”](#) before continuing.

Connector/J is also available as part of the Maven project. For more information, and to download the Connector/J JAR files, see the [Maven repository](#).

22.3.3.1. Installing Connector/J from a Binary Distribution

For the easiest method of installation, use the binary distribution of the Connector/J package. The binary distribution is available either as a tar/gzip or zip file. Extract it to a suitable location, then optionally

make the information about the package available by changing your `CLASSPATH` (see [Section 22.3.3.2, “Installing the Driver and Configuring the `CLASSPATH`”](#)).

MySQL Connector/J is distributed as a `.zip` or `.tar.gz` archive containing the sources, the class files, and the JAR archive named `mysql-connector-java-version-bin.jar`.

Starting with Connector/J 3.1.9, the `.class` files that constitute the JAR files are only included as part of the driver JAR file.

Starting with Connector/J 3.1.8, the archive also includes a debug build of the driver in a file named `mysql-connector-java-version-bin-g.jar`. Do not use the debug build of the driver unless instructed to do so when reporting a problem or a bug, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

Use the appropriate graphical or command-line utility to extract the distribution (for example, WinZip for the `.zip` archive, and `tar` for the `.tar.gz` archive). Because there are potentially long file names in the distribution, we use the GNU tar archive format. Use GNU tar (or an application that understands the GNU tar archive format) to unpack the `.tar.gz` variant of the distribution.

22.3.3.2. Installing the Driver and Configuring the `CLASSPATH`

Once you have extracted the distribution archive, you can install the driver by placing `mysql-connector-java-version-bin.jar` in your classpath, either by adding the full path to it to your `CLASSPATH` environment variable, or by directly specifying it with the command line switch `-cp` when starting the JVM.

To use the driver with the JDBC `DriverManager`, use `com.mysql.jdbc.Driver` as the class that implements `java.sql.Driver`.

You can set the `CLASSPATH` environment variable under UNIX, Linux or Mac OS X either locally for a user within their `.profile`, `.login` or other login file. You can also set it globally by editing the global `/etc/profile` file.

For example, add the Connector/J driver to your `CLASSPATH` using one of the following forms, depending on your command shell:

```
# Bourne-compatible shell (sh, ksh, bash, zsh):
shell> export CLASSPATH=/path/mysql-connector-java-ver-bin.jar:$CLASSPATH

# C shell (csh, tcsh):
shell> setenv CLASSPATH /path/mysql-connector-java-ver-bin.jar:$CLASSPATH
```

Within Windows 2000, Windows XP, Windows Server 2003 and Windows Vista, you set the environment variable through the System Control Panel.

To use MySQL Connector/J with an application server such as GlassFish, Tomcat or JBoss, read your vendor's documentation for more information on how to configure third-party class libraries, as most application servers ignore the `CLASSPATH` environment variable. For configuration examples for some J2EE application servers, see [Section 22.3.7, “Connection Pooling with Connector/J”](#), [Section 22.3.8, “Load Balancing with Connector/J”](#), and [Section 22.3.9, “Failover with Connector/J”](#). However, the authoritative source for JDBC connection pool configuration information for your particular application server is the documentation for that application server.

If you are developing servlets or JSPs, and your application server is J2EE-compliant, you can put the driver's `.jar` file in the `WEB-INF/lib` subdirectory of your webapp, as this is a standard location for third party class libraries in J2EE web applications.

You can also use the `MysqlDataSource` or `MysqlConnectionPoolDataSource` classes in the `com.mysql.jdbc.jdbc2.optional` package, if your J2EE application server supports or requires them. Starting with Connector/J 5.0.0, the `javax.sql.XADataSource` interface is implemented using the `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class, which supports XA distributed transactions when used in combination with MySQL server version 5.0.

The various `MysqlDataSource` classes support the following parameters (through standard set mutators):

- `user`
- `password`
- `serverName` (see the previous section about fail-over hosts)
- `databaseName`
- `port`

22.3.3.3. Upgrading from an Older Version

This section has information for users who are upgrading from one version of Connector/J to another, or to a new version of the MySQL server that supports a more recent level of JDBC. A newer version of Connector/J might include changes to support new features, improve existing functionality, or comply with new standards.

22.3.3.3.1. Upgrading to MySQL Connector/J 5.1.x

- In Connector/J 5.0.x and earlier, the alias for a table in a `SELECT` statement is returned when accessing the result set metadata using `ResultSetMetaData.getColumnname()`. This behavior however is not JDBC compliant, and in Connector/J 5.1 this behavior was changed so that the original table name, rather than the alias, is returned.

The JDBC-compliant behavior is designed to let API users reconstruct the DML statement based on the metadata within `ResultSet` and `ResultSetMetaData`.

You can get the alias for a column in a result set by calling `ResultSetMetaData.getColumnLabel()`. To use the old noncompliant behavior with `ResultSetMetaData.getColumnname()`, use the `useOldAliasMetadataBehavior` option and set the value to `true`.

In Connector/J 5.0.x, the default value of `useOldAliasMetadataBehavior` was `true`, but in Connector/J 5.1 this was changed to a default value of `false`.

22.3.3.3.2. JDBC-Specific Issues When Upgrading to MySQL Server 4.1 or Newer

- *Using the UTF-8 Character Encoding* - Prior to MySQL server version 4.1, the UTF-8 character encoding was not supported by the server, however the JDBC driver could use it, allowing storage of multiple character sets in `latin1` tables on the server.

Starting with MySQL-4.1, this functionality is deprecated. If you have applications that rely on this functionality, and can not upgrade them to use the official Unicode character support in MySQL server version 4.1 or newer, add the following property to your connection URL:

```
useOldUTF8Behavior=true
```

- *Server-side Prepared Statements* - Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer). If your

application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the following connection property:

```
useServerPrepStmts=false
```

22.3.3.3.3. Upgrading from MySQL Connector/J 3.0 to 3.1

Connector/J 3.1 is designed to be backward-compatible with Connector/J 3.0 as much as possible. Major changes are isolated to new functionality exposed in MySQL-4.1 and newer, which includes Unicode character sets, server-side prepared statements, `SQLState` codes returned in error messages by the server and various performance enhancements that can be enabled or disabled using configuration properties.

- **Unicode Character Sets:** See the next section, as well as [Section 10.1, “Character Set Support”](#), for information on this MySQL feature. If you have something misconfigured, it will usually show up as an error with a message similar to `Illegal mix of collations`.
- **Server-side Prepared Statements:** Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer).

Starting with version 3.1.7, the driver scans SQL you are preparing using all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this feature by passing `emulateUnsupportedPstmts=false` in your JDBC URL.

If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the connection property `useServerPrepStmts=false`.

- **Datetimes** with all-zero components (`0000-00-00 . . .`): These values cannot be represented reliably in Java. Connector/J 3.0.x always converted them to `NULL` when being read from a `ResultSet`.

Connector/J 3.1 throws an exception by default when these values are encountered, as this is the most correct behavior according to the JDBC and SQL standards. This behavior can be modified using the `zeroDateTimeBehavior` configuration property. The permissible values are:

- `exception` (the default), which throws an `SQLException` with an `SQLState` of `S1009`.
- `convertToNull`, which returns `NULL` instead of the date.
- `round`, which rounds the date to the nearest closest value which is `0001-01-01`.

Starting with Connector/J 3.1.7, `ResultSet.getString()` can be decoupled from this behavior using `noDatetimeStringSync=true` (the default value is `false`) so that you can retrieve the unaltered all-zero value as a `String`. Note that this also precludes using any time zone conversions, therefore the driver will not allow you to enable `noDatetimeStringSync` and `useTimezone` at the same time.

- **New SQLState Codes:** Connector/J 3.1 uses SQL:1999 `SQLState` codes returned by the MySQL server (if supported), which are different from the legacy X/Open state codes that Connector/J 3.0 uses. If connected to a MySQL server older than MySQL-4.1.0 (the oldest version to return `SQLStates` as part of the error code), the driver will use a built-in mapping. You can revert to the old mapping by using the configuration property `useSqlStateCodes=false`.
- **`ResultSet.getString()`:** Calling `ResultSet.getString()` on a `BLOB` column will now return the address of the `byte[]` array that represents it, instead of a `String` representation of the `BLOB`. `BLOB`

values have no character set, so they cannot be converted to `java.lang.Strings` without data loss or corruption.

To store strings in MySQL with LOB behavior, use one of the `TEXT` types, which the driver will treat as a `java.sql.Clob`.

- **Debug builds:** Starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-version-bin-g.jar` is shipped alongside the normal binary jar file that is named `mysql-connector-java-version-bin.jar`.

Starting with Connector/J 3.1.9, we do not ship the `.class` files unbundled, they are only available in the JAR archives that ship with the driver.

Do not use the debug build of the driver unless instructed to do so when reporting a problem or bug, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

22.3.3.4. Installing from the Development Source Tree



Caution

Read this section only if you are interested in helping us test our new code. To just get MySQL Connector/J up and running on your system, use a standard binary release distribution.

To install MySQL Connector/J from the development source tree, make sure that you have the following prerequisites:

- A Bazaar client, to check out the sources from our Launchpad repository (available from <http://bazaar-vcs.org/>).
- Apache Ant version 1.7 or newer (available from <http://ant.apache.org/>).
- JDK 1.4.2 or later. Although MySQL Connector/J can be used with older JDKs, compiling it from source requires at least JDK 1.4.2. To build Connector/J 5.1 requires JDK 1.6.x *and* an older JDK such as JDK 1.5.x; point your `JAVA_HOME` environment variable at the older installation.

To check out and compile a specific branch of MySQL Connector/J, follow these steps:

1. Check out the latest code from the branch that you want with one of the following commands.

The source code repository for MySQL Connector/J is located on Launchpad at <https://code.launchpad.net/connectorj>. To check out the latest development branch, use:

```
shell> bzip branch lp:connectorj
```

This creates a `connectorj` subdirectory in the current directory that contains the latest sources for the requested branch.

To check out the latest 5.1 code, use:

```
shell> bzip branch lp:connectorj/5.1
```

This creates a `5.1` subdirectory in the current directory containing the latest 5.1 code.

2. To build Connector/J 5.1, make sure that you have both JDK 1.6.x installed and an older JDK such as JDK 1.5.x. This is because Connector/J supports both JDBC 3.0 (which was prior to JDK 1.6.x) and JDBC 4.0. Set your `JAVA_HOME` environment variable to the path of the older JDK installation.

3. Change your current working directory to either the `connectorj` or `5.1` directory, depending on which branch you intend to build.
4. To build Connector/J 5.1, edit the `build.xml` to reflect the location of your JDK 1.6.x installation. The lines to change are:

```
<property name="com.mysql.jdbc.java6.javac" value="C:\jvms\jdk1.6.0\bin\javac.exe" />
<property name="com.mysql.jdbc.java6.rtjar" value="C:\jvms\jdk1.6.0\jre\lib\rt.jar" />
```

Alternatively, you can set the value of these property names through the Ant `-D` option.

5. Issue the following command to compile the driver and create a `.jar` file suitable for installation:

```
shell> ant dist
```

This creates a `build` directory in the current directory, where all build output will go. A directory is created in the `build` directory that includes the version number of the sources you are building from. This directory contains the sources, compiled `.class` files, and a `.jar` file suitable for deployment. For other possible targets, including ones that will create a fully packaged distribution, issue the following command:

```
shell> ant -projecthelp
```

6. A newly created `.jar` file containing the JDBC driver will be placed in the directory `build/mysql-connector-java-version`.

Install the newly created JDBC driver as you would a binary `.jar` file that you download from MySQL, by following the instructions in [Section 22.3.3.2, "Installing the Driver and Configuring the CLASSPATH"](#).

A package containing both the binary and source code for Connector/J 5.1 can also be found at the following location: [Connector/J 5.1 Download](#)

22.3.4. Connector/J Examples

Examples of using Connector/J are located throughout this document. This section provides a summary and links to these examples.

- [Example 22.1, "Connector/J: Obtaining a connection from the DriverManager"](#)
- [Example 22.2, "Connector/J: Using java.sql.Statement to execute a SELECT query"](#)
- [Example 22.3, "Connector/J: Calling Stored Procedures"](#)
- [Example 22.4, "Connector/J: Using Connection.prepareStatement\(\)"](#)
- [Example 22.5, "Connector/J: Registering output parameters"](#)
- [Example 22.6, "Connector/J: Setting CallableStatement input parameters"](#)
- [Example 22.7, "Connector/J: Retrieving results and output parameter values"](#)
- [Example 22.8, "Connector/J: Retrieving AUTO_INCREMENT column values using Statement.getGeneratedKeys\(\)"](#)
- [Example 22.9, "Connector/J: Retrieving AUTO_INCREMENT column values using SELECT LAST_INSERT_ID\(\)"](#)
- [Example 22.10, "Connector/J: Retrieving AUTO_INCREMENT column values in Updatable ResultSets"](#)

- [Example 22.11, “Connector/J: Using a connection pool with a J2EE application server”](#)
- [Example 22.12, “Connector/J: Example of transaction with retry logic”](#)

22.3.5. Connector/J (JDBC) Reference

This section of the manual contains reference material for MySQL Connector/J.

22.3.5.1. Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J

The name of the class that implements `java.sql.Driver` in MySQL Connector/J is `com.mysql.jdbc.Driver`. The `org.gjt.mm.mysql.Driver` class name is also usable for backward compatibility with MM.MySQL, the predecessor of Connector/J. Use this class name when registering the driver, or when otherwise configuring software to use MySQL Connector/J.

JDBC URL Format

The JDBC URL format for MySQL Connector/J is as follows, with items in square brackets ([,]) being optional:

```
jdbc:mysql://[host][,failoverhost...][:port]/[database] »
[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

If the host name is not specified, it defaults to `127.0.0.1`. If the port is not specified, it defaults to `3306`, the default port number for MySQL servers.

```
jdbc:mysql://[host:port],[host:port].../[database] »
[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

Here is a sample connection URL:

```
jdbc:mysql://localhost:3306/sakila?profileSQL=true
```

IPv6 Connections

For IPv6 connections, use this alternative syntax to specify hosts in the URL, `address=(key=value)`. Supported keys are:

- `(protocol=tcp)`, or `(protocol=pipe)` for named pipes on Windows.
- `(path=path_to_pipe)` for named pipes.
- `(host=hostname)` for TCP connections.
- `(port=port_number)` for TCP connections.

For example:

```
jdbc:mysql://address=(protocol=tcp)(host=localhost)(port=3306)(user=test)/db
```

Any other parameters are treated as host-specific properties that follow the conventions of the JDBC URL properties. This now allows per-host overrides of any configuration property for multi-host connections (that is, when using failover, load balancing, or replication). Limit the overrides to user, password, network timeouts and statement and metadata cache sizes; the results of other per-host overrides are not defined.

Initial Database for Connection

If the database is not specified, the connection is made with no default database. In this case, either call the `setCatalog()` method on the Connection instance, or fully specify table names using the database

name (that is, `SELECT dbname.tablename.colname FROM dbname.tablename...`) in your SQL. Opening a connection without specifying the database to use is generally only useful when building tools that work with multiple databases, such as GUI database managers.

**Note**

Always use the `Connection.setCatalog()` method to specify the desired database in JDBC applications, rather than the `USE database` statement.

Failover Support

MySQL Connector/J has failover support. This enables the driver to fail over to any number of slave hosts and still perform read-only queries. Failover only happens when the connection is in an `autoCommit(true)` state, because failover cannot happen reliably when a `transaction` is in progress. Most application servers and connection pools set `autoCommit` to `true` at the end of every transaction/connection use.

The failover functionality has the following behavior:

- If the URL property `autoReconnect` is `false`: Failover only happens at connection initialization, and fallback occurs when the driver determines that the first host has become available again.
- If the URL property `autoReconnect` is `true`: Failover happens when the driver determines that the connection has failed (checked before every query), and falls back to the first host when it determines that the host has become available again (after `queriesBeforeRetryMaster` queries have been issued).

In either case, whenever you are connected to a “failed-over” server, the connection is set to read-only state, so queries that attempt to modify data will throw exceptions (the query will **never** be processed by the MySQL server).

Setting Configuration Properties

Configuration properties define how Connector/J will make a connection to a MySQL server. Unless otherwise noted, properties can be set for a `DataSource` object or for a `Connection` object.

Configuration properties can be set in one of the following ways:

- Using the `set*()` methods on MySQL implementations of `java.sql.DataSource` (which is the preferred method when using implementations of `java.sql.DataSource`):
 - `com.mysql.jdbc.jdbc2.optional.MysqlDataSource`
 - `com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource`
- As a key/value pair in the `java.util.Properties` instance passed to `DriverManager.getConnection()` or `Driver.connect()`
- As a JDBC URL parameter in the URL given to `java.sql.DriverManager.getConnection()`, `java.sql.Driver.connect()` or the MySQL implementations of the `javax.sql.DataSource.setURL()` method.

**Note**

If the mechanism you use to configure a JDBC URL is XML-based, use the XML character literal `&` to separate configuration parameters, as the ampersand is a reserved character for XML.

The properties are listed in the following tables.

Connection/Authentication.

Property Name	Definition	Default Value	Since Version
<code>user</code>	The user to connect as		all versions
<code>password</code>	The password to use when connecting		all versions
<code>socketFactory</code>	The name of the class that the driver should use for creating socket connections to the server. This class must implement the interface 'com.mysql.jdbc.SocketFactory' and have public no-args constructor.	com.mysql.jdbc.StandardSocketFactory	3.0.3
<code>connectTimeout</code>	Timeout for socket connect (in milliseconds), with 0 being no timeout. Only works on JDK-1.4 or newer. Defaults to '0'.	0	3.0.1
<code>socketTimeout</code>	Timeout on network socket operations (0, the default means no timeout).	0	3.0.1
<code>connectionLifecycleInterceptors</code>	A comma-delimited list of classes that implement "com.mysql.jdbc.ConnectionLifecycleInterceptor" that should notified of connection lifecycle events (creation, destruction, commit, rollback, setCatalog and setAutoCommit) and potentially alter the execution of these commands. ConnectionLifecycleInterceptors are "stackable", more than one interceptor may be specified via the configuration property as a comma-delimited list, with the interceptors executed in order from left to right.		5.1.4
<code>useConfigs</code>	Load the comma-delimited list of configuration properties before parsing the URL or applying user-specified properties. These configurations are explained in the 'Configurations' of the documentation.		3.1.5
<code>authenticationPlugins</code>	Comma-delimited list of classes that implement com.mysql.jdbc.AuthenticationPlugin and which will be used for authentication unless disabled by "disabledAuthenticationPlugins" property.		5.1.19
<code>defaultAuthenticationPlugin</code>	Name of a class implementing com.mysql.jdbc.AuthenticationPlugin which will be used as the default authentication plugin (see below). It is an error to use a class which is not listed in "authenticationPlugins" nor it is one of the built-in plugins. It is an error to set as default a plugin which was disabled with "disabledAuthenticationPlugins" property. It is an error to set this value to null or the empty string (i.e. there must be at least a valid default authentication	com.mysql.jdbc.authentication.DefaultAuthenticationPlugin	5.1.19

	plugin specified for the connection, meeting all constraints listed above).		
<code>disabledAuthenticationPlugins</code>	Comma-delimited list of classes implementing <code>com.mysql.jdbc.AuthenticationPlugin</code> or mechanisms, i.e. "mysql_native_password". The authentication plugins or mechanisms listed will not be used for authentication which will fail if it requires one of them. It is an error to disable the default authentication plugin (either the one named by "defaultAuthenticationPlugin" property or the hard-coded one if "defaultAuthenticationPlugin" property is not set).		5.1.19
<code>interactiveClient</code>	Set the <code>CLIENT_INTERACTIVE</code> flag, which tells MySQL to timeout connections based on <code>INTERACTIVE_TIMEOUT</code> instead of <code>WAIT_TIMEOUT</code>	<code>false</code>	3.1.0
<code>localSocketAddress</code>	Hostname or IP address given to explicitly configure the interface that the driver will bind the client side of the TCP/IP connection to when connecting.		5.0.5
<code>propertiesTransform</code>	An implementation of <code>com.mysql.jdbc.ConnectionPropertiesTransform</code> that the driver will use to modify URL properties passed to the driver before attempting a connection		3.1.4
<code>useCompression</code>	Use zlib compression when communicating with the server (true/false)? Defaults to 'false'.	<code>false</code>	3.0.17

Networking.

Property Name	Definition	Default Value	Since Version
<code>maxAllowedPacket</code>	Maximum allowed packet size to send to server. If not set, the value of system variable 'max_allowed_packet' will be used to initialize this upon connecting. This value will not take effect if set larger than the value of 'max_allowed_packet'.	<code>-1</code>	5.1.8
<code>tcpKeepAlive</code>	If connecting using TCP/IP, should the driver set <code>SO_KEEPALIVE</code> ?	<code>true</code>	5.0.7
<code>tcpNoDelay</code>	If connecting using TCP/IP, should the driver set <code>SO_TCP_NODELAY</code> (disabling the Nagle Algorithm)?	<code>true</code>	5.0.7
<code>tcpRcvBuf</code>	If connecting using TCP/IP, should the driver set <code>SO_RCV_BUF</code> to the given value? The default value of '0', means use the platform default value for this property)	<code>0</code>	5.0.7
<code>tcpSndBuf</code>	If connecting using TCP/IP, should the driver set <code>SO_SND_BUF</code> to the given value? The default value of '0', means use the platform default value for this property)	<code>0</code>	5.0.7
<code>tcpTrafficClass</code>	If connecting using TCP/IP, should the driver set traffic class or type-of-service fields ?See the	<code>0</code>	5.0.7

	documentation for <code>java.net.Socket.setTrafficClass()</code> for more information.		
--	--	--	--

High Availability and Clustering.

Property Name	Definition	Default Value	Since Version
<code>autoReconnect</code>	Should the driver try to re-establish stale and/or dead connections? If enabled the driver will throw an exception for a queries issued on a stale or dead connection, which belong to the current transaction, but will attempt reconnect before the next query issued on the connection in a new transaction. The use of this feature is not recommended, because it has side effects related to session state and data consistency when applications don't handle <code>SQLExceptions</code> properly, and is only designed to be used when you are unable to configure your application to handle <code>SQLExceptions</code> resulting from dead and stale connections properly. Alternatively, as a last option, investigate setting the MySQL server variable "wait_timeout" to a high value, rather than the default of 8 hours.	<code>false</code>	1.1
<code>autoReconnectForPools</code>	Use a reconnection strategy appropriate for connection pools (defaults to 'false')	<code>false</code>	3.1.3
<code>failOverReadOnly</code>	When failing over in <code>autoReconnect</code> mode, should the connection be set to 'read-only'?	<code>true</code>	3.0.12
<code>maxReconnects</code>	Maximum number of reconnects to attempt if <code>autoReconnect</code> is true, default is '3'.	3	1.1
<code>reconnectAtTxEnd</code>	If <code>autoReconnect</code> is set to true, should the driver attempt reconnections at the end of every transaction?	<code>false</code>	3.0.10
<code>retriesAllDown</code>	When using loadbalancing, the number of times the driver should cycle through available hosts, attempting to connect. Between cycles, the driver will pause for 250ms if no servers are available.	120	5.1.6
<code>initialTimeout</code>	If <code>autoReconnect</code> is enabled, the initial time to wait between re-connect attempts (in seconds, defaults to '2').	2	1.1
<code>roundRobinLoadBalance</code>	When <code>autoReconnect</code> is enabled, and <code>failoverReadOnly</code> is false, should we pick hosts to connect to on a round-robin basis?	<code>false</code>	3.1.2
<code>queriesBeforeRetryMaster</code>	Number of queries to issue before falling back to master when failed over (when using multi-host failover). Whichever condition is met first, 'queriesBeforeRetryMaster' or 'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Defaults to 50.	50	3.0.2
<code>secondsBeforeRetryMaster</code>	How long should the driver wait, when failed over, before attempting	30	3.0.2

Connector/J (JDBC) Reference

<code>selfDestructOnPingMaxOpenConnections</code>	If set to a non-zero value, the driver will report close the connection and report failure when <code>Connection.ping()</code> or <code>Connection.isValid(int)</code> is called if the connection's count of commands sent to the server exceeds this value.	0	5.1.6
<code>selfDestructOnPingSecondOpenConnections</code>	If set to a non-zero value, the driver will report close the connection and report failure when <code>Connection.ping()</code> or <code>Connection.isValid(int)</code> is called if the connection's lifetime exceeds this value.	0	5.1.6
<code>resourceId</code>	A globally unique name that identifies the resource that this datasource or connection is connected to, used for <code>XAResource.isSameRM()</code> when the driver can't determine this value based on hostnames used in the URL		5.0.1

Security.

Property Name	Definition	Default Value	Since Version
<code>allowMultiQueries</code>	Allow the use of ';' to delimit multiple queries during one statement (true/false), defaults to 'false', and does not affect the <code>addBatch()</code> and <code>executeBatch()</code> methods, which instead rely on <code>rewriteBatchStatements</code> .	false	3.1.1
<code>useSSL</code>	Use SSL when communicating with the server (true/false), defaults to 'false'	false	3.0.2
<code>requireSSL</code>	Require SSL connection if <code>useSSL=true</code> ? (defaults to 'false').	false	3.1.0
<code>verifyServerCertificate</code>	If "useSSL" is set to "true", should the driver verify the server's certificate? When using this feature, the keystore parameters should be specified by the "clientCertificateKeyStore*" properties, rather than system properties.	true	5.1.6
<code>clientCertificateKeyStoreURL</code>	URL to the client certificate KeyStore (if not specified, use defaults)		5.1.0
<code>clientCertificateKeyStoreType</code>	KeyStore type for client certificates (NULL or empty means use the default, which is "JKS". Standard keystore types supported by the JVM are "JKS" and "PKCS12", your environment may have more available depending on what security products are installed and available to the JVM.	JKS	5.1.0
<code>clientCertificateKeyStorePassword</code>	Password for the client certificates KeyStore		5.1.0
<code>trustCertificateKeyStoreURL</code>	URL to the trusted root certificate KeyStore (if not specified, use defaults)		5.1.0
<code>trustCertificateKeyStoreType</code>	KeyStore type for trusted root certificates (NULL or empty means use the default, which is "JKS". Standard keystore types supported by the JVM are "JKS" and "PKCS12", your environment may have more available depending on what security products are installed and available to the JVM.	JKS	5.1.0

<code>trustCertificateKeyStorePassword</code>	Password for the trusted root certificates KeyStore		5.1.0
<code>allowLoadLocalInfile</code>	Should the driver allow use of 'LOAD DATA LOCAL INFILE...' (defaults to 'true').	<code>true</code>	3.0.3
<code>allowUrlInLocalInfile</code>	Should the driver allow URLs in 'LOAD DATA LOCAL INFILE' statements?	<code>false</code>	3.1.4
<code>paranoid</code>	Take measures to prevent exposure sensitive information in error messages and clear data structures holding sensitive data when possible? (defaults to 'false')	<code>false</code>	3.0.1
<code>passwordCharacterEncoding</code>	What character encoding is used for passwords? Leaving this set to the default value (null), uses the platform character set, which works for ISO8859_1 (i.e. "latin1") passwords. For passwords in other character encodings, the encoding will have to be specified with this property, as it's not possible for the driver to auto-detect this.		5.1.7

Performance Extensions.

Property Name	Definition	Default Value	Since Version
<code>callableStmtCacheSize</code>	If 'cacheCallableStmts' is enabled, how many callable statements should be cached?	100	3.1.2
<code>metadataCacheSize</code>	The number of queries to cache ResultSetMetadata for if cacheResultSetMetaData is set to 'true' (default 50)	50	3.1.1
<code>useLocalSessionState</code>	Should the driver refer to the internal values of autocommit and transaction isolation that are set by <code>Connection.setAutoCommit()</code> and <code>Connection.setTransactionIsolation()</code> and transaction state as maintained by the protocol, rather than querying the database or blindly sending commands to the database for <code>commit()</code> or <code>rollback()</code> method calls?	<code>false</code>	3.1.7
<code>useLocalTransactionState</code>	Should the driver use the in-transaction state provided by the MySQL protocol to determine if a <code>commit()</code> or <code>rollback()</code> should actually be sent to the database?	<code>false</code>	5.1.7
<code>prepStmtCacheSize</code>	If prepared statement caching is enabled, how many prepared statements should be cached?	25	3.0.10
<code>prepStmtCacheSqlLimit</code>	If prepared statement caching is enabled, what's the largest SQL the driver will cache the parsing for?	256	3.0.10
<code>parseInfoCacheFactory</code>	Name of a class implementing <code>com.mysql.jdbc.CacheAdapterFactory</code> , which will be used to create caches for the parsed representation of client-side prepared statements.	<code>com.mysql.jdbc.PerCon</code>	5.1.16
<code>alwaysSendSetIsolation</code>	Should the driver always communicate with the database when <code>Connection.setTransactionIsolation()</code> is	<code>true</code>	3.1.7

	called? If set to false, the driver will only communicate with the database when the requested transaction isolation is different than the whichever is newer, the last value that was set via <code>Connection.setTransactionIsolation()</code> , or the value that was read from the server when the connection was established. Note that <code>useLocalSessionState=true</code> will force the same behavior as <code>alwaysSendSetIsolation=false</code> , regardless of how <code>alwaysSendSetIsolation</code> is set.		
<code>maintainTimeStats</code>	Should the driver maintain various internal timers to enable idle time calculations as well as more verbose error messages when the connection to the server fails? Setting this property to false removes at least two calls to <code>System.currentTimeMillis()</code> per query.	<code>true</code>	3.1.9
<code>useCursorFetch</code>	If connected to MySQL > 5.0.2, and <code>setFetchSize() > 0</code> on a statement, should that statement use cursor-based fetching to retrieve rows?	<code>false</code>	5.0.0
<code>blobSendChunkSize</code>	Chunk to use when sending BLOB/CLOBs via <code>ServerPreparedStatement</code> s	<code>1048576</code>	3.1.9
<code>cacheCallableStmts</code>	Should the driver cache the parsing stage of <code>CallableStatements</code>	<code>false</code>	3.1.2
<code>cachePrepStmts</code>	Should the driver cache the parsing stage of <code>PreparedStatement</code> s of client-side prepared statements, the "check" for suitability of server-side prepared and server-side prepared statements themselves?	<code>false</code>	3.0.10
<code>cacheResultSetMetadata</code>	Should the driver cache <code>ResultSetMetaData</code> for <code>Statements</code> and <code>PreparedStatement</code> s? (Req. JDK-1.4+, true/false, default 'false')	<code>false</code>	3.1.1
<code>cacheServerConfiguration</code>	Should the driver cache the results of 'SHOW VARIABLES' and 'SHOW COLLATION' on a per-URL basis?	<code>false</code>	3.1.5
<code>defaultFetchSize</code>	The driver will call <code>setFetchSize(n)</code> with this value on all newly-created <code>Statements</code>	<code>0</code>	3.1.9
<code>dontTrackOpenResources</code>	The JDBC specification requires the driver to automatically track and close resources, however if your application doesn't do a good job of explicitly calling <code>close()</code> on statements or result sets, this can cause memory leakage. Setting this property to true relaxes this constraint, and can be more memory efficient for some applications.	<code>false</code>	3.1.7
<code>dynamicCalendars</code>	Should the driver retrieve the default calendar when required, or cache it per connection/session?	<code>false</code>	3.1.5
<code>elideSetAutoCommits</code>	If using MySQL-4.1 or newer, should the driver only issue 'set autocommit=n' queries when the server's state doesn't match the requested state by <code>Connection.setAutoCommit(boolean)</code> ?	<code>false</code>	3.1.3

<code>enableQueryTimeouts</code>	When enabled, query timeouts set via <code>Statement.setQueryTimeout()</code> use a shared <code>java.util.Timer</code> instance for scheduling. Even if the timeout doesn't expire before the query is processed, there will be memory used by the <code>TimerTask</code> for the given timeout which won't be reclaimed until the time the timeout would have expired if it hadn't been cancelled by the driver. High-load environments might want to consider disabling this functionality.	<code>true</code>	5.0.6
<code>holdResultsOpenOverStatementClose</code>	Should the driver close result sets on <code>Statement.close()</code> as required by the JDBC specification?	<code>false</code>	3.1.7
<code>largeRowSizeThreshold</code>	What size result set row should the JDBC driver consider "large", and thus use a more memory-efficient way of representing the row internally?	2048	5.1.1
<code>loadBalanceStrategy</code>	If using a load-balanced connection to connect to SQL nodes in a MySQL Cluster/ NDB configuration (by using the URL prefix <code>"jdbc:mysql:loadbalance://"</code>), which load balancing algorithm should the driver use: (1) "random" - the driver will pick a random host for each request. This tends to work better than round-robin, as the randomness will somewhat account for spreading loads where requests vary in response time, while round-robin can sometimes lead to overloaded nodes if there are variations in response times across the workload. (2) "bestResponseTime" - the driver will route the request to the host that had the best response time for the previous transaction.	<code>random</code>	5.0.6
<code>locatorFetchBufferSize</code>	If 'emulateLocators' is configured to 'true', what size buffer should be used when fetching BLOB data for <code>getBinaryInputStream()</code> ?	1048576	3.2.1
<code>rewriteBatchedStatements</code>	Should the driver use multiqueries (irregardless of the setting of "allowMultiQueries") as well as rewriting of prepared statements for INSERT into multi-value inserts when <code>executeBatch()</code> is called? Notice that this has the potential for SQL injection if using plain <code>java.sql.Statements</code> and your code doesn't sanitize input correctly. Notice that for prepared statements, server-side prepared statements can not currently take advantage of this rewrite option, and that if you don't specify stream lengths when using <code>PreparedStatement.set*Stream()</code> , the driver won't be able to determine the optimum number of parameters per batch and you might receive an error from the driver that the resultant packet is too large. <code>Statement.getGeneratedKeys()</code> for these rewritten statements only works when the entire batch includes INSERT statements.	<code>false</code>	3.1.13

<code>useDirectRowUnpack</code>	Use newer result set row unpacking code that skips a copy from network buffers to a MySQL packet instance and instead reads directly into the result set row data buffers.	<code>true</code>	5.1.1
<code>useDynamicCharsetInfo</code>	Should the driver use a per-connection cache of character set information queried from the server when necessary, or use a built-in static mapping that is more efficient, but isn't aware of custom character sets or character sets implemented after the release of the JDBC driver?	<code>true</code>	5.0.6
<code>useFastDateParsing</code>	Use internal String->Date/Time/Timestamp conversion routines to avoid excessive object creation?	<code>true</code>	5.0.5
<code>useFastIntParsing</code>	Use internal String->Integer conversion routines to avoid excessive object creation?	<code>true</code>	3.1.4
<code>useJvmCharsetConverters</code>	Always use the character encoding routines built into the JVM, rather than using lookup tables for single-byte character sets?	<code>false</code>	5.0.1
<code>useReadAheadInput</code>	Use newer, optimized non-blocking, buffered input stream when reading from the server?	<code>true</code>	3.1.5

Debugging/Profiling.

Property Name	Definition	Default Value	Since Version
<code>logger</code>	The name of a class that implements "com.mysql.jdbc.log.Log" that will be used to log messages to. (default is "com.mysql.jdbc.log.StandardLogger", which logs to STDERR)	<code>com.mysql.jdbc.log.StandardLogger</code>	3.1.1
<code>gatherPerfMetrics</code>	Should the driver gather performance metrics, and report them via the configured logger every 'reportMetricsIntervalMillis' milliseconds?	<code>false</code>	3.1.2
<code>profileSQL</code>	Trace queries and their execution/fetch times to the configured logger (true/false) defaults to 'false'	<code>false</code>	3.1.0
<code>profileSql</code>	Deprecated, use 'profileSQL' instead. Trace queries and their execution/fetch times on STDERR (true/false) defaults to 'false'		2.0.14
<code>reportMetricsIntervalMillis</code>	If 'gatherPerfMetrics' is enabled, how often should they be logged (in ms)?	<code>30000</code>	3.1.2
<code>maxQuerySizeToLog</code>	Controls the maximum length/size of a query that will get logged when profiling or tracing	<code>2048</code>	3.1.3
<code>packetDebugBufferSize</code>	The maximum number of packets to retain when 'enablePacketDebug' is true	<code>20</code>	3.1.3
<code>slowQueryThresholdMillis</code>	If 'logSlowQueries' is enabled, how long should a query (in ms) before it is logged as 'slow'?	<code>2000</code>	3.1.2
<code>slowQueryThresholdNanos</code>	If 'useNanosForElapsedTime' is set to true, and this property is set to a non-zero value, the driver	<code>0</code>	5.0.7

Connector/J (JDBC) Reference

	will use this threshold (in nanosecond units) to determine if a query was slow.		
<code>useUsageAdvisor</code>	Should the driver issue 'usage' warnings advising proper and efficient usage of JDBC and MySQL Connector/J to the log (true/false, defaults to 'false')?	<code>false</code>	3.1.1
<code>autoGenerateTestcaseScript</code>	Should the driver dump the SQL it is executing, including server-side prepared statements to STDERR?	<code>false</code>	3.1.9
<code>autoSlowLog</code>	Instead of using <code>slowQueryThreshold*</code> to determine if a query is slow enough to be logged, maintain statistics that allow the driver to determine queries that are outside the 99th percentile?	<code>true</code>	5.1.4
<code>clientInfoProvider</code>	The name of a class that implements the <code>com.mysql.jdbc.JDBC4ClientInfoProvider</code> interface in order to support JDBC-4.0's <code>Connection.getClientInfo()</code> methods	<code>com.mysql.jdbc.JDBC4ClientInfoProvider</code>	5.1.10
<code>dumpMetadataOnColumnNotFoundException</code>	Should the driver dump the field-level metadata of a result set into the exception message when <code>ResultSet.findColumn()</code> fails?	<code>false</code>	3.1.13
<code>dumpQueriesOnException</code>	Should the driver dump the contents of the query sent to the server in the message for <code>SQLExceptions</code> ?	<code>false</code>	3.1.3
<code>enablePacketDebug</code>	When enabled, a ring-buffer of <code>'packetDebugBufferSize'</code> packets will be kept, and dumped when exceptions are thrown in key areas in the driver's code	<code>false</code>	3.1.3
<code>explainSlowQueries</code>	If <code>'logSlowQueries'</code> is enabled, should the driver automatically issue an 'EXPLAIN' on the server and send the results to the configured log at a WARN level?	<code>false</code>	3.1.2
<code>includeInnodbStatusInDeadlockExceptions</code>	Include the output of "SHOW ENGINE INNODB STATUS" in exception messages when deadlock exceptions are detected?	<code>false</code>	5.0.7
<code>includeThreadDumpInDeadlockExceptions</code>	Include a current Java thread dump in exception messages when deadlock exceptions are detected?	<code>false</code>	5.1.15
<code>includeThreadNamesAsStatusInDeadlockExceptions</code>	Include the name of the current thread as a comment visible in "SHOW PROCESSLIST", or in Innodb deadlock dumps, useful in correlation with <code>"includeInnodbStatusInDeadlockExceptions=true"</code> and <code>"includeThreadDumpInDeadlockExceptions=true"</code> .	<code>false</code>	5.1.15
<code>logSlowQueries</code>	Should queries that take longer than <code>'slowQueryThresholdMillis'</code> be logged?	<code>false</code>	3.1.2
<code>logXaCommands</code>	Should the driver log XA commands sent by <code>MysqlXaConnection</code> to the server, at the DEBUG level of logging?	<code>false</code>	5.0.5

<code>profilerEventHandler</code>	Name of a class that implements the interface <code>com.mysql.jdbc.profiler.ProfilerEventHandler</code> that will be used to handle profiling/tracing events.	<code>com.mysql.jdbc.profiler.I</code>	5.1.6
<code>resultSetSizeThreshold</code>	If the usage advisor is enabled, how many rows should a result set contain before the driver warns that it is suspiciously large?	100	5.0.5
<code>traceProtocol</code>	Should trace-level network protocol be logged?	<code>false</code>	3.1.2
<code>useNanosForElapsedTime</code>	For profiling/debugging functionality that measures elapsed time, should the driver try to use nanoseconds resolution if available (JDK >= 1.5)?	<code>false</code>	5.0.7

Miscellaneous.

Property Name	Definition	Default Value	Since Version
<code>useUnicode</code>	Should the driver use Unicode character encodings when handling strings? Should only be used when the driver can't determine the character set mapping, or you are trying to 'force' the driver to use a character set that MySQL either doesn't natively support (such as UTF-8), true/false, defaults to 'true'	<code>true</code>	1.1g
<code>characterEncoding</code>	If 'useUnicode' is set to true, what character encoding should the driver use when dealing with strings? (defaults is to 'autodetect')		1.1g
<code>characterSetResults</code>	Character set to tell the server to return results as.		3.0.13
<code>connectionCollation</code>	If set, tells the server to use this collation via 'set collation_connection'		3.0.13
<code>useBlobToStoreUTF8OutsideBmp</code>	Tells the driver to treat [MEDIUM/LONG]BLOB columns as [LONG]VARCHAR columns holding text encoded in UTF-8 that has characters outside the BMP (4-byte encodings), which MySQL server can't handle natively.	<code>false</code>	5.1.3
<code>utf8OutsideBmpExcludedColumnNamePattern</code>	When "useBlobToStoreUTF8OutsideBmp" is set to "true", column names matching the given regex will still be treated as BLOBs unless they match the regex specified for "utf8OutsideBmpIncludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package.		5.1.3
<code>utf8OutsideBmpIncludedColumnNamePattern</code>	Used to specify exclusion rules to "utf8OutsideBmpExcludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package.		5.1.3
<code>loadBalanceEnableJMX</code>	Enables JMX-based management of load-balanced connection groups, including live addition/removal of hosts from load-balancing pool.	<code>false</code>	5.1.13
<code>sessionVariables</code>	A comma-separated list of name/value pairs to be sent as SET SESSION ... to the server when the driver connects.		3.1.8

<code>useColumnNamesInFindColumns</code>	Prior to JDBC-4.0, the JDBC specification had a bug related to what could be given as a "column name" to <code>ResultSet</code> methods like <code>findColumn()</code> , or getters that took a <code>String</code> property. JDBC-4.0 clarified "column name" to mean the label, as given in an "AS" clause and returned by <code>ResultSetMetaData.getColumnLabel()</code> , and if no AS clause, the column name. Setting this property to "true" will give behavior that is congruent to JDBC-3.0 and earlier versions of the JDBC specification, but which because of the specification bug could give unexpected results. This property is preferred over "useOldAliasMetadataBehavior" unless you need the specific behavior that it provides with respect to <code>ResultSetMetadata</code> .	<code>false</code>	5.1.7
<code>allowNanAndInf</code>	Should the driver allow NaN or +/- INF values in <code>PreparedStatement.setDouble()</code> ?	<code>false</code>	3.1.5
<code>autoClosePstmtStreams</code>	Should the driver automatically call <code>.close()</code> on streams/readers passed as arguments via <code>set*()</code> methods?	<code>false</code>	3.1.12
<code>autoDeserialize</code>	Should the driver automatically detect and de-serialize objects stored in BLOB fields?	<code>false</code>	3.1.5
<code>blobsAreStrings</code>	Should the driver always treat BLOBs as Strings - specifically to work around dubious metadata returned by the server for GROUP BY clauses?	<code>false</code>	5.0.8
<code>capitalizeTypeNames</code>	Capitalize type names in <code>DatabaseMetaData</code> ? (usually only useful when using <code>WebObjects</code> , true/false, defaults to 'false')	<code>true</code>	2.0.7
<code>clobCharacterEncoding</code>	The character encoding to use for sending and retrieving TEXT, MEDIUMTEXT and LONGTEXT values instead of the configured connection characterEncoding		5.0.0
<code>clobberStreamingResults</code>	This will cause a 'streaming' <code>ResultSet</code> to be automatically closed, and any outstanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server.	<code>false</code>	3.0.9
<code>compensateOnDuplicateKeys</code>	Should the driver compensate for the update counts of "ON DUPLICATE KEY" INSERT statements (2 = 1, 0 = 1) when using prepared statements?	<code>false</code>	5.1.7
<code>continueBatchOnError</code>	Should the driver continue processing batch commands if one statement fails. The JDBC spec allows either way (defaults to 'true').	<code>true</code>	3.0.3
<code>createDatabaseIfNotExist</code>	Creates the database given in the URL if it doesn't yet exist. Assumes the configured user has permissions to create databases.	<code>false</code>	3.1.9
<code>emptyStringsConvertToZero</code>	Should the driver allow conversions from empty string fields to numeric values of '0'?	<code>true</code>	3.1.8

Connector/J (JDBC) Reference

<code>emulateLocators</code>	Should the driver emulate <code>java.sql.Blobs</code> with locators? With this feature enabled, the driver will delay loading the actual Blob data until the one of the retrieval methods (<code>getInputStream()</code> , <code>getBytes()</code> , and so forth) on the blob data stream has been accessed. For this to work, you must use a column alias with the value of the column to the actual name of the Blob. The feature also has the following restrictions: The SELECT that created the result set must reference only one table, the table must have a primary key; the SELECT must alias the original blob column name, specified as a string, to an alternate name; the SELECT must cover all columns that make up the primary key.	<code>false</code>	3.1.0
<code>emulateUnsupportedPstmts</code>	Should the driver detect prepared statements that are not supported by the server, and replace them with client-side emulated versions?	<code>true</code>	3.1.7
<code>exceptionInterceptors</code>	Comma-delimited list of classes that implement <code>com.mysql.jdbc.ExceptionInterceptor</code> . These classes will be instantiated one per Connection instance, and all <code>SQLExceptions</code> thrown by the driver will be allowed to be intercepted by these interceptors, in a chained fashion, with the first class listed as the head of the chain.		5.1.8
<code>functionsNeverReturnBlobs</code>	Should the driver always treat data from functions returning BLOBs as Strings - specifically to work around dubious metadata returned by the server for GROUP BY clauses?	<code>false</code>	5.0.8
<code>generateSimpleParameterMetadata</code>	Should the driver generate simplified parameter metadata for PreparedStatements when no metadata is available either because the server couldn't support preparing the statement, or server-side prepared statements are disabled?	<code>false</code>	5.0.5
<code>ignoreNonTxTables</code>	Ignore non-transactional table warning for rollback? (defaults to 'false').	<code>false</code>	3.0.9
<code>jdbcCompliantTruncation</code>	Should the driver throw <code>java.sql.DataTruncation</code> exceptions when data is truncated as is required by the JDBC specification when connected to a server that supports warnings (MySQL 4.1.0 and newer)? This property has no effect if the server sql-mode includes <code>STRICT_TRANS_TABLES</code> .	<code>true</code>	3.1.2
<code>loadBalanceAutoCommitStatements</code>	When load-balancing is enabled for auto-commit statements (via <code>loadBalanceAutoCommitStatementThreshold</code>), the statement counter will only increment when the SQL matches the regular expression. By default, every statement issued matches.		5.1.15
<code>loadBalanceAutoCommitStatementsThreshold</code>	When auto-commit is enabled, the number of statements which should be executed before triggering load-balancing to rebalance. Default	<code>0</code>	5.1.15

Connector/J (JDBC) Reference

	value of 0 causes load-balanced connections to only rebalance when exceptions are encountered, or auto-commit is disabled and transactions are explicitly committed or rolled back.		
<code>loadBalanceBlacklistTime</code>	Time in milliseconds between checks of servers which are unavailable, by controlling how long a server lives in the global blacklist.	0	5.1.0
<code>loadBalanceConnectionGroup</code>	Logical group of load-balanced connections within a classloader, used to manage different groups independently. If not specified, live management of load-balanced connections is disabled.		5.1.13
<code>loadBalanceExceptionChecker</code>	Fully-qualified class name of custom exception checker. The class must implement <code>com.mysql.jdbc.LoadBalanceExceptionChecker</code> interface, and is used to inspect <code>SQLException</code> s and determine whether they should trigger fail-over to another host in a load-balanced deployment.	<code>com.mysql.jdbc.</code>	5.1.13
<code>loadBalancePingTimeout</code>	Time in milliseconds to wait for ping response from each of load-balanced physical connections when using load-balanced Connection.	0	5.1.13
<code>loadBalanceSQLExceptions</code>	Comma-delimited list of classes/interfaces used by default load-balanced exception checker to determine whether a given <code>SQLException</code> should trigger failover. The comparison is done using <code>Class.isInstance(SQLException)</code> using the thrown <code>SQLException</code> .		5.1.13
<code>loadBalanceSQLStateFailover</code>	Comma-delimited list of <code>SQLState</code> codes used by default load-balanced exception checker to determine whether a given <code>SQLException</code> should trigger failover. The <code>SQLState</code> of a given <code>SQLException</code> is evaluated to determine whether it begins with any value in the comma-delimited list.		5.1.13
<code>loadBalanceValidateConnection</code>	Should the load-balanced Connection explicitly check whether the connection is live when swapping to a new physical connection at commit/rollback?	<code>false</code>	5.1.13
<code>maxRows</code>	The maximum number of rows to return (0, the default means return all rows).	-1	all versions
<code>netTimeoutForStreamingResults</code>	What value should the driver automatically set the server setting 'net_write_timeout' to when the streaming result sets feature is in use? (value has unit of seconds, the value '0' means the driver will not try and adjust this value)	600	5.1.0
<code>noAccessToProcedureBodies</code>	When determining procedure parameter types for <code>CallableStatements</code> , and the connected user can't access procedure bodies through "SHOW CREATE PROCEDURE" or select on <code>mysql.proc</code> should the driver instead create basic metadata (all parameters reported as IN VARCHARs, but allowing <code>registerOutParameter()</code> to be called on them anyway) instead of throwing an exception?	<code>false</code>	5.0.3

Connector/J (JDBC) Reference

<code>noDatetimeStringSync</code>	Don't ensure that <code>ResultSet.getDatetimeType().toString().equals(ResultSet.getString())</code>	<code>false</code>	3.1.7
<code>noTimezoneConversionForTime</code>	Don't convert TIME values using the server timezone if 'useTimezone'='true'	<code>false</code>	5.0.0
<code>nullCatalogMeansCurrent</code>	When <code>DatabaseMetadataMethods</code> ask for a 'catalog' parameter, does the value null mean use the current catalog? (this is not JDBC-compliant, but follows legacy behavior from earlier versions of the driver)	<code>true</code>	3.1.8
<code>nullNamePatternMatchesAll</code>	Should <code>DatabaseMetaData</code> methods that accept *pattern parameters treat null the same as '%' (this is not JDBC-compliant, however older versions of the driver accepted this departure from the specification)	<code>true</code>	3.1.8
<code>overrideSupportsIntegrityEnhancementFacility</code>	Should the driver return "true" for <code>DatabaseMetaData.supportsIntegrityEnhancementFacility()</code> even if the database doesn't support it to workaround applications that require this method to return "true" to signal support of foreign keys, even though the SQL specification states that this facility contains much more than just foreign key support (one such application being OpenOffice)?	<code>false</code>	3.1.12
<code>padCharsWithSpace</code>	If a result set column has the CHAR type and the value does not fill the amount of characters specified in the DDL for the column, should the driver pad the remaining characters with space (for ANSI compliance)?	<code>false</code>	5.0.6
<code>pedantic</code>	Follow the JDBC spec to the letter.	<code>false</code>	3.0.0
<code>pinGlobalTxToPhysicalConnections</code>	When using XAConnections, should the driver ensure that operations on a given XID are always routed to the same physical connection? This allows the XAConnection to support "XA START ... JOIN" after "XA END" has been called	<code>false</code>	5.0.1
<code>populateInsertRowWithDefaults</code>	When using ResultSets that are CONCUR_UPDATABLE, should the driver pre-populate the "insert" row with default values from the DDL for the table used in the query so those values are immediately available for ResultSet accessors? This functionality requires a call to the database for metadata each time a result set of this type is created. If disabled (the default), the default values will be populated by the an internal call to <code>refreshRow()</code> which pulls back default values and/or values changed by triggers.	<code>false</code>	5.0.5
<code>processEscapeCodesForPrepares</code>	Should the driver process escape codes in queries that are prepared?	<code>true</code>	3.1.12
<code>queryTimeoutKillsConnection</code>	If the timeout given in <code>Statement.setQueryTimeout()</code> expires, should the driver forcibly abort the Connection instead of attempting to abort the query?	<code>false</code>	5.1.9

Connector/J (JDBC) Reference

<code>relaxAutoCommit</code>	If the version of MySQL the driver connects to does not support transactions, still allow calls to <code>commit()</code> , <code>rollback()</code> and <code>setAutoCommit()</code> (true/false, defaults to 'false')?	<code>false</code>	2.0.13
<code>retainStatementAfterResultSetClose</code>	Should the driver retain the Statement reference in a ResultSet after <code>ResultSet.close()</code> has been called. This is not JDBC-compliant after JDBC-4.0.	<code>false</code>	3.1.11
<code>rollbackOnPooledClose</code>	Should the driver issue a <code>rollback()</code> when the logical connection in a pool is closed?	<code>true</code>	3.0.15
<code>runningCTS13</code>	Enables workarounds for bugs in Sun's JDBC compliance testsuite version 1.3	<code>false</code>	3.1.7
<code>serverTimezone</code>	Override detection/mapping of timezone. Used when timezone from server doesn't map to Java timezone		3.0.2
<code>statementInterceptors</code>	A comma-delimited list of classes that implement "com.mysql.jdbc.StatementInterceptor" that should be placed "in between" query execution to influence the results. StatementInterceptors are "chainable", the results returned by the "current" interceptor will be passed on to the next in in the chain, from left-to-right order, as specified in this property.		5.1.1
<code>strictFloatingPoint</code>	Used only in older versions of compliance test	<code>false</code>	3.0.0
<code>strictUpdates</code>	Should the driver do strict checking (all primary keys selected) of updatable result sets (true, false, defaults to 'true')?	<code>true</code>	3.0.4
<code>tinyIntIsBit</code>	Should the driver treat the datatype TINYINT(1) as the BIT type (because the server silently converts BIT -> TINYINT(1) when creating tables)?	<code>true</code>	3.0.16
<code>transformedBitIsBoolean</code>	If the driver converts TINYINT(1) to a different type, should it use BOOLEAN instead of BIT for future compatibility with MySQL-5.0, as MySQL-5.0 has a BIT type?	<code>false</code>	3.1.9
<code>treatUtilDateAsTimestamp</code>	Should the driver treat <code>java.util.Date</code> as a <code>TIMESTAMP</code> for the purposes of <code>PreparedStatement.setObject()</code> ?	<code>true</code>	5.0.5
<code>ultraDevHack</code>	Create PreparedStatements for <code>prepareCall()</code> when required, because UltraDev is broken and issues a <code>prepareCall()</code> for <code>_all_</code> statements? (true/false, defaults to 'false')	<code>false</code>	2.0.3
<code>useAffectedRows</code>	Don't set the <code>CLIENT_FOUND_ROWS</code> flag when connecting to the server (not JDBC-compliant, will break most applications that rely on "found" rows vs. "affected rows" for DML statements), but does cause "correct" update counts from "INSERT ... ON DUPLICATE KEY UPDATE" statements to be returned by the server.	<code>false</code>	5.1.7
<code>useGmtMillisForDatetimes</code>	Convert between session timezone and GMT before creating Date and Timestamp instances (value	<code>false</code>	3.1.12

Connector/J (JDBC) Reference

	of "false" is legacy behavior, "true" leads to more JDBC-compliant behavior.		
<code>useHostsInPrivileges</code>	Add '@hostname' to users in DatabaseMetaData.getColumn/TablePrivileges() (true/false), defaults to 'true'.	<code>true</code>	3.0.2
<code>useInformationSchema</code>	When connected to MySQL-5.0.7 or newer, should the driver use the INFORMATION_SCHEMA to derive information used by DatabaseMetaData?	<code>false</code>	5.0.0
<code>useJDBCCompliantTimezoneShift</code>	Should the driver use JDBC-compliant rules when converting TIME/TIMESTAMP/DATETIME values' timezone information for those JDBC arguments which take a java.util.Calendar argument? (Notice that this option is exclusive of the "useTimezone=true" configuration option.)	<code>false</code>	5.0.0
<code>useLegacyDatetimeCode</code>	Use code for DATE/TIME/DATETIME/TIMESTAMP handling in result sets and statements that consistently handles timezone conversions from client to server and back again, or use the legacy code for these datatypes that has been in the driver for backwards-compatibility?	<code>true</code>	5.1.6
<code>useOldAliasMetadataBehavior</code>	Should the driver use the legacy behavior for "AS" clauses on columns and tables, and only return aliases (if any) for ResultSetMetaData.getColumn() or ResultSetMetaData.getTable() rather than the original column/table name? In 5.0.x, the default value was true.	<code>false</code>	5.0.4
<code>useOldUTF8Behavior</code>	Use the UTF-8 behavior the driver did when communicating with 4.0 and older servers	<code>false</code>	3.1.6
<code>useOnlyServerErrorMessage</code>	Don't prepend 'standard' SQLState error messages to error messages returned by the server.	<code>true</code>	3.0.15
<code>useSSPSCompatibleTimezoneShift</code>	If migrating from an environment that was using server-side prepared statements, and the configuration property "useJDBCCompliantTimezoneShift" set to "true", use compatible behavior when not using server-side prepared statements when sending TIMESTAMP values to the MySQL server.	<code>false</code>	5.0.5
<code>useServerPrepStmts</code>	Use server-side prepared statements if the server supports them?	<code>false</code>	3.1.0
<code>useSqlStateCodes</code>	Use SQL Standard state codes instead of 'legacy' X/Open/SQL state codes (true/false), default is 'true'	<code>true</code>	3.1.3
<code>useStreamLengthsInPrepStmts</code>	Honor stream length parameter in PreparedStatement/ResultSet.setXXXStream() method calls (true/false, defaults to 'true')?	<code>true</code>	3.0.2
<code>useTimezone</code>	Convert time/date types between client and server timezones (true/false, defaults to 'false')?	<code>false</code>	3.0.2

<code>useUnbufferedInput</code>	Don't use <code>BufferedInputStream</code> for reading data from the server	<code>true</code>	3.0.11
<code>yearIsDateType</code>	Should the JDBC driver treat the MySQL type "YEAR" as a <code>java.sql.Date</code> , or as a SHORT?	<code>true</code>	3.1.9
<code>zeroDateTimeBehavior</code>	What should happen when the driver encounters DATETIME values that are composed entirely of zeros (used by MySQL to represent invalid dates)? Valid values are "exception", "round" and "convertToNull".	<code>exception</code>	3.1.4

Connector/J also supports access to MySQL using named pipes on Windows NT, Windows 2000, or Windows XP using the `NamedPipeSocketFactory` as a plugin-socket factory using the `socketFactory` property. If you do not use a `namedPipePath` property, the default of `'\\.\pipe\MySQL'` is used. If you use the `NamedPipeSocketFactory`, the host name and port number values in the JDBC url are ignored. To enable this feature, use:

```
socketFactory=com.mysql.jdbc.NamedPipeSocketFactory
```

Named pipes only work when connecting to a MySQL server on the same physical machine where the JDBC driver is running. In simple performance tests, named pipe access is between 30%-50% faster than the standard TCP/IP access. However, this varies per system, and named pipes are slower than TCP/IP in many Windows configurations.

To create your own socket factories, follow the example code in

`com.mysql.jdbc.NamedPipeSocketFactory`, or `com.mysql.jdbc.StandardSocketFactory`.

22.3.5.1.1. Properties Files for the `useConfigs` Option

The `useConfigs` connection option is a convenient shorthand for specifying combinations of options for particular scenarios. The argument values you can use with this option correspond to the names of `.properties` files within the Connector/J `mysql-connector-java-version-bin.jar` JAR file. For example, the Connector/J 5.1.9 driver includes the following configuration properties files:

```
$ unzip mysql-connector-java-5.1.19-bin.jar '*/configs/*'
Archive:  mysql-connector-java-5.1.19-bin.jar
  creating:  com/mysql/jdbc/configs/
  inflating:  com/mysql/jdbc/configs/3-0-Compat.properties
  inflating:  com/mysql/jdbc/configs/5-0-Compat.properties
  inflating:  com/mysql/jdbc/configs/clusterBase.properties
  inflating:  com/mysql/jdbc/configs/coldFusion.properties
  inflating:  com/mysql/jdbc/configs/fullDebug.properties
  inflating:  com/mysql/jdbc/configs/maxPerformance.properties
  inflating:  com/mysql/jdbc/configs/solarisMaxPerformance.properties
```

To specify one of these combinations of options, specify `useConfigs=3-0-Compat`, `useConfigs=maxPerformance`, and so on. The following sections show the options that are part of each `useConfigs` setting. For the details of why each one is included, see the comments in the `.properties` files.

3-0-Compat

```
emptyStringsConvertToZero=true
jdbcCompliantTruncation=false
noDatetimeStringSync=true
nullCatalogMeansCurrent=true
nullNamePatternMatchesAll=true
transformedBitIsBoolean=false
dontTrackOpenResources=true
zeroDateTimeBehavior=convertToNull
useServerPrepStmts=false
```

```

autoClosePstmtStreams=true
processEscapeCodesForPrepStmts=false
useFastDateParsing=false
populateInsertRowWithDefaultValues=false
useDirectRowUnpack=false

```

5-0-Compat

```
useDirectRowUnpack=false
```

clusterBase

```

autoReconnect=true
failOverReadOnly=false
roundRobinLoadBalance=true

```

coldFusion

```

useDynamicCharsetInfo=false
alwaysSendSetIsolation=false
useLocalSessionState=true
autoReconnect=true

```

fullDebug

```

profileSQL=true
gatherPerMetrics=true
useUsageAdvisor=true
logSlowQueries=true
explainSlowQueries=true

```

maxPerformance

```

cachePrepStmts=true
cacheCallableStmts=true
cacheServerConfiguration=true
useLocalSessionState=true
elideSetAutoCommits=true
alwaysSendSetIsolation=false
enableQueryTimeouts=false

```

solarisMaxPerformance

```

useUnbufferedInput=false
useReadAheadInput=false
maintainTimeStats=false

```

22.3.5.2. JDBC API Implementation Notes

MySQL Connector/J passes all of the tests in the publicly available version of Sun's JDBC compliance test suite. This section gives details on a interface-by-interface level about implementation decisions that might affect how you code applications with MySQL Connector/J. The JDBC specification is vague about how certain functionality should be implemented, or the specification enables leeway in implementation.

- **BLOB**

Starting with Connector/J version 3.1.0, you can emulate BLOBs with locators by adding the property `emulateLocators=true` to your JDBC URL. Using this method, the driver will delay loading the actual BLOB data until you retrieve the other data and then use retrieval methods (`getInputStream()`, `getBytes()`, and so forth) on the BLOB data stream.

You must use a column alias with the value of the column to the actual name of the BLOB, for example:

```
SELECT id, 'data' as blob_data from blobtable
```

You must also follow these rules:

- The `SELECT` must reference only one table. The table must have a [primary key](#).
- The `SELECT` must alias the original BLOB column name, specified as a string, to an alternate name.
- The `SELECT` must cover all columns that make up the primary key.

The BLOB implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, use the corresponding `PreparedStatement.setBlob()` or `ResultSet.updateBlob()` (in the case of updatable result sets) methods to save changes back to the database.

- **CallableStatement**

Starting with Connector/J 3.1.1, stored procedures are supported when connecting to MySQL version 5.0 or newer using the `CallableStatement` interface. Currently, the `getParameterMetaData()` method of `CallableStatement` is not supported.

- **CLOB**

The CLOB implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, use the `PreparedStatement.setClob()` method to save changes back to the database. The JDBC API does not have a `ResultSet.updateClob()` method.

- **Connection**

Unlike the pre-Connector/J JDBC driver (`MM.MySQL`), the `isClosed()` method does not ping the server to determine if it is available. In accordance with the JDBC specification, it only returns true if `closed()` has been called on the connection. If you need to determine if the connection is still valid, issue a simple query, such as `SELECT 1`. The driver will throw an exception if the connection is no longer valid.

- **DatabaseMetaData**

[Foreign key](#) information (`getImportedKeys()/getExportedKeys()` and `getCrossReference()`) is only available from `InnoDB` tables. The driver uses `SHOW CREATE TABLE` to retrieve this information, so if any other storage engines add support for foreign keys, the driver would transparently support them as well.

- **PreparedStatement**

PreparedStatements are implemented by the driver, as MySQL does not have a prepared statement feature. Because of this, the driver does not implement `getParameterMetaData()` or `getMetaData()` as it would require the driver to have a complete SQL parser in the client.

Starting with version 3.1.0 MySQL Connector/J, server-side prepared statements and binary-encoded result sets are used when the server supports them.

Take care when using a server-side prepared statement with **large** parameters that are set using `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()`, or `setClob()`. To re-execute the statement with any large parameter changed to a nonlarge parameter, call `clearParameters()` and set all parameters again. The reason for this is as follows:

- During both server-side prepared statements and client-side emulation, large data is exchanged only when `PreparedStatement.execute()` is called.

- Once that has been done, the stream used to read the data on the client side is closed (as per the JDBC spec), and cannot be read from again.
- If a parameter changes from large to nonlarge, the driver must reset the server-side state of the prepared statement to allow the parameter that is being changed to take the place of the prior large value. This removes all of the large data that has already been sent to the server, thus requiring the data to be re-sent, using the `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()` or `setClob()` method.

Consequently, to change the type of a parameter to a nonlarge one, you must call `clearParameters()` and set all parameters of the prepared statement again before it can be re-executed.

- **ResultSet**

By default, ResultSets are completely retrieved and stored in memory. In most cases this is the most efficient way to operate, and due to the design of the MySQL network protocol is easier to implement. If you are working with ResultSets that have a large number of rows or large values, and cannot allocate heap space in your JVM for the memory required, you can tell the driver to stream the results back one row at a time.

To enable this functionality, create a `Statement` instance in the following manner:

```
stmt = conn.createStatement( java.sql.ResultSet.TYPE_FORWARD_ONLY,  
                             java.sql.ResultSet.CONCUR_READ_ONLY );  
stmt.setFetchSize( Integer.MIN_VALUE );
```

The combination of a forward-only, read-only result set, with a fetch size of `Integer.MIN_VALUE` serves as a signal to the driver to stream result sets row-by-row. After this, any result sets created with the statement will be retrieved row-by-row.

There are some caveats with this approach. You must read all of the rows in the result set (or close it) before you can issue any other queries on the connection, or an exception will be thrown.

The earliest the locks these statements hold can be released (whether they be `MyISAM` table-level locks or row-level locks in some other storage engine such as `InnoDB`) is when the statement completes.

If the statement is within scope of a transaction, then locks are released when the transaction completes (which implies that the statement needs to complete first). As with most other databases, statements are not complete until all the results pending on the statement are read or the active result set for the statement is closed.

Therefore, if using streaming results, process them as quickly as possible if you want to maintain concurrent access to the tables referenced by the statement producing the result set.

- **ResultSetMetaData**

The `isAutoIncrement()` method only works when using MySQL servers 4.0 and newer.

- **Statement**

When using versions of the JDBC driver earlier than 3.2.1, and connected to server versions earlier than 5.0.3, the `setFetchSize()` method has no effect, other than to toggle result set streaming as described above.

Connector/J 5.0.0 and later include support for both `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL 5.0.0 or newer server, and require a separate connection to issue the `KILL QUERY` statement. In the case of `setQueryTimeout()`, the implementation creates an additional thread to handle the timeout functionality.

**Note**

Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeException` rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

**Note**

The MySQL statement `KILL QUERY` (which is what the driver uses to implement `Statement.cancel()`) is non-deterministic; thus, avoid the use of `Statement.cancel()` if possible. If no query is in process, the next query issued will be killed by the server. This race condition is guarded against as of Connector/J 5.1.18.

MySQL does not support SQL cursors, and the JDBC driver doesn't emulate them, so `setCursorName()` has no effect.

Connector/J 5.1.3 and later include two additional methods:

- `setLocalInfileInputStream()` sets an `InputStream` instance that will be used to send data to the MySQL server for a `LOAD DATA LOCAL INFILE` statement rather than a `FileInputStream` or `URLInputStream` that represents the path given as an argument to the statement.

This stream will be read to completion upon execution of a `LOAD DATA LOCAL INFILE` statement, and will automatically be closed by the driver, so it needs to be reset before each call to `execute*()` that would cause the MySQL server to request data to fulfill the request for `LOAD DATA LOCAL INFILE`.

If this value is set to `NULL`, the driver will revert to using a `FileInputStream` or `URLInputStream` as required.

- `getLocalInfileInputStream()` returns the `InputStream` instance that will be used to send data in response to a `LOAD DATA LOCAL INFILE` statement.

This method returns `NULL` if no such stream has been set using `setLocalInfileInputStream()`.

22.3.5.3. Java, JDBC and MySQL Types

MySQL Connector/J is flexible in the way it handles conversions between MySQL data types and Java data types.

In general, any MySQL data type can be converted to a `java.lang.String`, and any numeric type can be converted to any of the Java numeric types, although round-off, overflow, or loss of precision may occur.

**Note**

All `TEXT` types return `Types.LONGVARCHAR` with different `getPrecision()` values (65535, 255, 16777215, and 2147483647 respectively) with `getColumnType()` returning `-1`. This behavior is intentional even though

`TINYTEXT` does not fall, regarding to its size, within the `LONGVARCHAR` category. This is to avoid different handling inside the same base type. And `getColumnType()` returns `-1` because the internal server handling is of type `TEXT`, which is similar to `BLOB`.

Also note that `getColumnTypeName()` will return `VARCHAR` even though `getColumnType()` returns `Types.LONGVARCHAR`, because `VARCHAR` is the designated column database-specific name for this type.

Starting with Connector/J 3.1.0, the JDBC driver issues warnings or throws `DataTruncation` exceptions as is required by the JDBC specification unless the connection was configured not to do so by using the property `jdbcCompliantTruncation` and setting it to `false`.

The conversions that are always guaranteed to work are listed in the following table. The first column lists one or more MySQL data types, and the second column lists one or more Java types to which the MySQL types can be converted.

Table 22.24. Connection Properties - Miscellaneous

These MySQL Data Types	Can always be converted to these Java types
<code>CHAR</code> , <code>VARCHAR</code> , <code>BLOB</code> , <code>TEXT</code> , <code>ENUM</code> , and <code>SET</code>	<code>java.lang.String</code> , <code>java.io.InputStream</code> , <code>java.io.Reader</code> , <code>java.sql.Blob</code> , <code>java.sql.Clob</code>
<code>FLOAT</code> , <code>REAL</code> , <code>DOUBLE PRECISION</code> , <code>NUMERIC</code> , <code>DECIMAL</code> , <code>TINYINT</code> , <code>SMALLINT</code> , <code>MEDIUMINT</code> , <code>INTEGER</code> , <code>BIGINT</code>	<code>java.lang.String</code> , <code>java.lang.Short</code> , <code>java.lang.Integer</code> , <code>java.lang.Long</code> , <code>java.lang.Double</code> , <code>java.math.BigDecimal</code>
<code>DATE</code> , <code>TIME</code> , <code>DATETIME</code> , <code>TIMESTAMP</code>	<code>java.lang.String</code> , <code>java.sql.Date</code> , <code>java.sql.Timestamp</code>



Note

Round-off, overflow or loss of precision may occur if you choose a Java numeric data type that has less precision or capacity than the MySQL data type you are converting to/from.

The `ResultSet.getObject()` method uses the type conversions between MySQL and Java types, following the JDBC specification where appropriate. The value returned by `ResultSetMetaData.getColumnClassName()` is also shown below. For more information on the `java.sql.Types` classes see [Java 2 Platform Types](#).

Table 22.25. MySQL Types to Java Types for `ResultSet.getObject()`

MySQL Type Name	Return value of <code>getColumnClassName</code>	Returned as Java Class
<code>BIT(1)</code> (new in MySQL-5.0)	<code>BIT</code>	<code>java.lang.Boolean</code>
<code>BIT(> 1)</code> (new in MySQL-5.0)	<code>BIT</code>	<code>byte[]</code>
<code>TINYINT</code>	<code>TINYINT</code>	<code>java.lang.Boolean</code> if the configuration property <code>tinyIntIsBit</code> is set to <code>true</code> (the default) and the storage size is 1, or <code>java.lang.Integer</code> if not.
<code>BOOL</code> , <code>BOOLEAN</code>	<code>TINYINT</code>	See <code>TINYINT</code> , above as these are aliases for <code>TINYINT(1)</code> , currently.

MySQL Type Name	Return value of <code>getColumnClassName</code>	Returned as Java Class
<code>SMALLINT[(M)] [UNSIGNED]</code>	<code>SMALLINT [UNSIGNED]</code>	<code>java.lang.Integer</code> (regardless if <code>UNSIGNED</code> or not)
<code>MEDIUMINT[(M)] [UNSIGNED]</code>	<code>MEDIUMINT [UNSIGNED]</code>	<code>java.lang.Integer</code> , if <code>UNSIGNED</code> <code>java.lang.Long</code> (C/J 3.1 and earlier), or <code>java.lang.Integer</code> for C/J 5.0 and later
<code>INT, INTEGER[(M)] [UNSIGNED]</code>	<code>INTEGER [UNSIGNED]</code>	<code>java.lang.Integer</code> , if <code>UNSIGNED</code> <code>java.lang.Long</code>
<code>BIGINT[(M)] [UNSIGNED]</code>	<code>BIGINT [UNSIGNED]</code>	<code>java.lang.Long</code> , if <code>UNSIGNED</code> <code>java.math.BigInteger</code>
<code>FLOAT[(M,D)]</code>	<code>FLOAT</code>	<code>java.lang.Float</code>
<code>DOUBLE[(M,B)]</code>	<code>DOUBLE</code>	<code>java.lang.Double</code>
<code>DECIMAL[(M[,D])]</code>	<code>DECIMAL</code>	<code>java.math.BigDecimal</code>
<code>DATE</code>	<code>DATE</code>	<code>java.sql.Date</code>
<code>DATETIME</code>	<code>DATETIME</code>	<code>java.sql.Timestamp</code>
<code>TIMESTAMP[(M)]</code>	<code>TIMESTAMP</code>	<code>java.sql.Timestamp</code>
<code>TIME</code>	<code>TIME</code>	<code>java.sql.Time</code>
<code>YEAR[(2 4)]</code>	<code>YEAR</code>	If <code>yearIsDateType</code> configuration property is set to <code>false</code> , then the returned object type is <code>java.sql.Short</code> . If set to <code>true</code> (the default), then the returned object is of type <code>java.sql.Date</code> with the date set to January 1st, at midnight.
<code>CHAR(M)</code>	<code>CHAR</code>	<code>java.lang.String</code> (unless the character set for the column is <code>BINARY</code> , then <code>byte[]</code> is returned.
<code>VARCHAR(M) [BINARY]</code>	<code>VARCHAR</code>	<code>java.lang.String</code> (unless the character set for the column is <code>BINARY</code> , then <code>byte[]</code> is returned.
<code>BINARY(M)</code>	<code>BINARY</code>	<code>byte[]</code>
<code>VARBINARY(M)</code>	<code>VARBINARY</code>	<code>byte[]</code>
<code>TINYBLOB</code>	<code>TINYBLOB</code>	<code>byte[]</code>
<code>TINYTEXT</code>	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>BLOB</code>	<code>BLOB</code>	<code>byte[]</code>
<code>TEXT</code>	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>MEDIUMBLOB</code>	<code>MEDIUMBLOB</code>	<code>byte[]</code>
<code>MEDIUMTEXT</code>	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>LOBLOB</code>	<code>LOBLOB</code>	<code>byte[]</code>
<code>LONGTEXT</code>	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>ENUM('value1', 'value2', ...)</code>	<code>CHAR..)</code>	<code>java.lang.String</code>
<code>SET('value1', 'value2', ...)</code>	<code>CHAR..)</code>	<code>java.lang.String</code>

22.3.5.4. Using Character Sets and Unicode

All strings sent from the JDBC driver to the server are converted automatically from native Java Unicode form to the client character encoding, including all queries sent using `Statement.execute()`,

`Statement.executeUpdate()`, `Statement.executeQuery()` as well as all `PreparedStatement` and `CallableStatement` parameters with the exclusion of parameters set using `setBytes()`, `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()` and `setBlob()`.

Number of Encodings Per Connection

In MySQL Server 4.1 and higher, Connector/J supports a single character encoding between client and server, and any number of character encodings for data returned by the server to the client in `ResultSets`.

Prior to MySQL Server 4.1, Connector/J supported a single character encoding per connection, which could either be automatically detected from the server configuration, or could be configured by the user through the `useUnicode` and `characterEncoding` properties.

Setting the Character Encoding

The character encoding between client and server is automatically detected upon connection. You specify the encoding on the server using the `character_set_server` [488] for server versions 4.1.0 and newer, and `character_set` system variable for server versions older than 4.1.0. The driver automatically uses the encoding specified by the server. For more information, see [Section 10.1.3.1, “Server Character Set and Collation”](#).

For example, to use 4-byte UTF-8 character sets with Connector/J, configure the MySQL server with `character_set_server=utf8mb4` [488], and leave `characterEncoding` out of the Connector/J connection string. Connector/J will then autodetect the UTF-8 setting.

To override the automatically detected encoding on the client side, use the `characterEncoding` property in the URL used to connect to the server.

To allow multiple character sets to be sent from the client, use the UTF-8 encoding, either by configuring `utf8` as the default server character set, or by configuring the JDBC driver to use UTF-8 through the `characterEncoding` property.

When specifying character encodings on the client side, use Java-style names. The following table lists MySQL character set names and the corresponding Java-style names:

Table 22.26. MySQL to Java Encoding Name Translations

MySQL Character Set Name	Java-Style Character Encoding Name
<code>ascii</code>	US-ASCII
<code>big5</code>	Big5
<code>gbk</code>	GBK
<code>sjis</code>	SJIS (or Cp932 or MS932 for MySQL Server < 4.1.11)
<code>cp932</code>	Cp932 or MS932 (MySQL Server > 4.1.11)
<code>gb2312</code>	EUC_CN
<code>ujis</code>	EUC_JP
<code>euackr</code>	EUC_KR
<code>latin1</code>	Cp1252
<code>latin2</code>	ISO8859_2
<code>greek</code>	ISO8859_7
<code>hebrew</code>	ISO8859_8

MySQL Character Set Name	Java-Style Character Encoding Name
cp866	Cp866
tis620	TIS620
cp1250	Cp1250
cp1251	Cp1251
cp1257	Cp1257
macroman	MacRoman
macce	MacCentralEurope
utf8	UTF-8
ucs2	UnicodeBig



Warning

Do not issue the query `set names` with Connector/J, as the driver will not detect that the character set has changed, and will continue to use the character set detected during the initial connection setup.

22.3.5.5. Connecting Securely Using SSL

SSL in MySQL Connector/J encrypts all data (other than the initial handshake) between the JDBC driver and the server. The performance penalty for enabling SSL is an increase in query processing time between 35% and 50%, depending on the size of the query, and the amount of data it returns.

For SSL support to work, you must have the following:

- A JDK that includes JSSE (Java Secure Sockets Extension), like JDK-1.4.1 or newer. SSL does not currently work with a JDK that you can add JSSE to, like JDK-1.2.x or JDK-1.3.x due to the following JSSE bug: <http://developer.java.sun.com/developer/bugParade/bugs/4273544.html>
- A MySQL server that supports SSL and has been compiled and configured to do so, which is MySQL 4.0.4 or later. For more information, see [Section 6.3.8.2, “Configuring MySQL for SSL”](#).
- A client certificate (covered later in this section)

The system works through two Java truststore files, one file contains the certificate information for the server (`truststore` in the examples below). The other file contains the certificate for the client (`keystore` in the examples below). All Java truststore files are password protected by supplying a suitable password to the `keytool` when you create the files. You need the file names and associated passwords to create an SSL connection.

You will first need to import the MySQL server CA Certificate into a Java truststore. A sample MySQL server CA Certificate is located in the `SSL` subdirectory of the MySQL source distribution. This is what SSL will use to determine if you are communicating with a secure MySQL server. Alternatively, use the CA Certificate that you have generated or been provided with by your SSL provider.

To use Java's `keytool` to create a truststore in the current directory, and import the server's CA certificate (`cacert.pem`), you can do the following (assuming that `keytool` is in your path. The `keytool` is typically located in the `bin` subdirectory of your JDK or JRE):

```
shell> keytool -import -alias mysqlServerCACert \
  -file cacert.pem -keystore truststore
```

Enter the password when prompted for the keystore file. Interaction with `keytool` looks like this:

```
Enter keystore password: *****
```

```

Owner: EMAILADDRESS=walrus@example.com, CN=Walrus,
      O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=Walrus,
       O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Serial number: 0
Valid from:
  Fri Aug 02 16:55:53 CDT 2002 until: Sat Aug 02 16:55:53 CDT 2003
Certificate fingerprints:
  MD5:  61:91:A0:F2:03:07:61:7A:81:38:66:DA:19:C4:8D:AB
  SHA1: 25:77:41:05:D5:AD:99:8C:14:8C:CA:68:9C:2F:B8:89:C3:34:4D:6C
Trust this certificate? [no]: yes
Certificate was added to keystore

```

You then have two options: either import the client certificate that matches the CA certificate you just imported, or create a new client certificate.

Importing an existing certificate requires the certificate to be in DER format. You can use [openssl](#) to convert an existing certificate into the new format. For example:

```
shell> openssl x509 -outform DER -in client-cert.pem -out client.cert
```

Now import the converted certificate into your keystore using [keytool](#):

```
shell> keytool -import -file client.cert -keystore keystore -alias mysqlClientCertificate
```

To generate your own client certificate, use [keytool](#) to create a suitable certificate and add it to the [keystore](#) file:

```
shell> keytool -genkey -keyalg rsa \
  -alias mysqlClientCertificate -keystore keystore
```

Keytool will prompt you for the following information, and create a keystore named [keystore](#) in the current directory.

Respond with information that is appropriate for your situation:

```

Enter keystore password: *****
What is your first and last name?
  [Unknown]: Matthews
What is the name of your organizational unit?
  [Unknown]: Software Development
What is the name of your organization?
  [Unknown]: MySQL AB
What is the name of your City or Locality?
  [Unknown]: Flossmoor
What is the name of your State or Province?
  [Unknown]: IL
What is the two-letter country code for this unit?
  [Unknown]: US
Is <CN=Matthews, OU=Software Development, O=MySQL AB,
  L=Flossmoor, ST=IL, C=US> correct?
  [no]: y

Enter key password for <mysqlClientCertificate>
  (RETURN if same as keystore password):

```

Finally, to get JSSE to use the keystore and truststore that you have generated, you need to set the following system properties when you start your JVM, replacing [path_to_keystore_file](#) with the full path to the keystore file you created, [path_to_truststore_file](#) with the path to the truststore file you created, and using the appropriate password values for each property. You can do this either on the command line:

```

-Djavax.net.ssl.keyStore=path\_to\_keystore\_file
-Djavax.net.ssl.keyStorePassword=password
-Djavax.net.ssl.trustStore=path\_to\_truststore\_file

```

```
-Djavax.net.ssl.trustStorePassword=password
```

Or you can set the values directly within the application:

```
System.setProperty("javax.net.ssl.keyStore", "path_to_keystore_file");
System.setProperty("javax.net.ssl.keyStorePassword", "password");
System.setProperty("javax.net.ssl.trustStore", "path_to_truststore_file");
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

You will also need to set `useSSL` to `true` in your connection parameters for MySQL Connector/J, either by adding `useSSL=true` to your URL, or by setting the property `useSSL` to `true` in the `java.util.Properties` instance you pass to `DriverManager.getConnection()`.

You can test that SSL is working by turning on JSSE debugging (as detailed below), and look for the following key events:

```
...
*** ClientHello, v3.1
RandomCookie: GMT: 1018531834 bytes = { 199, 148, 180, 215, 74, 12, »
    54, 244, 0, 168, 55, 103, 215, 64, 16, 138, 225, 190, 132, 153, 2, »
    217, 219, 239, 202, 19, 121, 78 }
Session ID: {}
Cipher Suites: { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 17 }
Compression Methods: { 0 }
***
[write] MD5 and SHA1 hashes: len = 59
0000: 01 00 00 37 03 01 3D B6 90 FA C7 94 B4 D7 4A 0C ...7..=.....J.
0010: 36 F4 00 A8 37 67 D7 40 10 8A E1 BE 84 99 02 D9 6...7g.@.....
0020: DB EF CA 13 79 4E 00 00 10 00 05 00 04 00 09 00 ...yN.....
0030: 0A 00 12 00 13 00 03 00 11 01 00 .....
main, WRITE: SSL v3.1 Handshake, length = 59
main, READ: SSL v3.1 Handshake, length = 74
*** ServerHello, v3.1
RandomCookie: GMT: 1018577560 bytes = { 116, 50, 4, 103, 25, 100, 58, »
    202, 79, 185, 178, 100, 215, 66, 254, 21, 83, 187, 190, 42, 170, 3, »
    132, 110, 82, 148, 160, 92 }
Session ID: {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63, »
    182, 158, 30, 11, 150, 79, 170, 76, 255, 92, 15, 226, 24, 17, 177, »
    219, 158, 177, 187, 143}
Cipher Suite: { 0, 5 }
Compression Method: 0
***
%% Created: [Session-1, SSL_RSA_WITH_RC4_128_SHA]
** SSL_RSA_WITH_RC4_128_SHA
[read] MD5 and SHA1 hashes: len = 74
0000: 02 00 00 46 03 01 3D B6 43 98 74 32 04 67 19 64 ...F..=.C.t2.g.d
0010: 3A CA 4F B9 B2 64 D7 42 FE 15 53 BB BE 2A AA 03 :.O..d.B..S.*..
0020: 84 6E 52 94 A0 5C 20 A3 E3 54 35 51 7F FC FE B2 .nR...\..T5Q....
0030: B3 44 3F B6 9E 1E 0B 96 4F AA 4C FF 5C 0F E2 18 .D?.....O.L.\...
0040: 11 B1 DB 9E B1 BB 8F 00 05 00 .....
main, READ: SSL v3.1 Handshake, length = 1712
...

```

JSSE provides debugging (to `stdout`) when you set the following system property: `-Djavax.net.debug=all`. This will tell you what keystores and truststores are being used, as well as what is going on during the SSL handshake and certificate exchange. It will be helpful when trying to determine what is not working when trying to get an SSL connection to happen.

22.3.5.6. Connecting Using PAM Authentication

Java applications using Connector/J 5.1.21 and higher can connect to MySQL servers that use the pluggable authentication module (PAM) authentication scheme.

For PAM authentication to work, you must have the following:

- A MySQL server that supports PAM authentication: a commercial distribution of MySQL 5.5.16 or higher. See [Section 6.3.6.2, “The PAM Authentication Plugin”](#) for more information. Connector/J implements the same cleartext authentication method as in [Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”](#).
- SSL capability, as explained in [Section 22.3.5.5, “Connecting Securely Using SSL”](#). Because the PAM authentication scheme sends the original password to the server, the connection to the server must be encrypted.

PAM authentication support is enabled by default in Connector/J 5.1.21 and up, so no extra configuration is needed.

To disable the PAM authentication feature, specify `mysql_clear_password` (the method) or `com.mysql.jdbc.authentication.MysqlClearPasswordPlugin` (the class name) in the comma-separated list of arguments for the `disabledAuthenticationPlugins` connection option. See [Section 22.3.5.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”](#) for details about that connection option.

22.3.5.7. Using Master/Slave Replication with ReplicationConnection

Connector/J 3.1.7 and higher includes a variant of the driver that will automatically send queries to a read/write master, or a failover or round-robin loadbalanced set of slaves based on the state of `Connection.getReadOnly()`.

An application signals that it wants a transaction to be read-only by calling `Connection.setReadOnly(true)`, this replication-aware connection will use one of the slave connections, which are load-balanced per-vm using a round-robin scheme (a given connection is sticky to a slave unless that slave is removed from service). If you have a write transaction, or if you have a read that is time-sensitive (remember, replication in MySQL is asynchronous), set the connection to be not read-only, by calling `Connection.setReadOnly(false)` and the driver will ensure that further calls are sent to the master MySQL server. The driver takes care of propagating the current state of autocommit, isolation level, and catalog between all of the connections that it uses to accomplish this load balancing functionality.

To enable this functionality, use the `com.mysql.jdbc.ReplicationDriver` class when configuring your application server's connection pool or when creating an instance of a JDBC driver for your standalone application. Because it accepts the same URL format as the standard MySQL JDBC driver, `ReplicationDriver` does not currently work with `java.sql.DriverManager`-based connection creation unless it is the only MySQL JDBC driver registered with the `DriverManager`.

Here is a short example of how `ReplicationDriver` might be used in a standalone application:

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.util.Properties;

import com.mysql.jdbc.ReplicationDriver;

public class ReplicationDriverDemo {

    public static void main(String[] args) throws Exception {
        ReplicationDriver driver = new ReplicationDriver();

        Properties props = new Properties();

        // We want this for failover on the slaves
        props.put("autoReconnect", "true");

        // We want to load balance between the slaves
```

```

props.put("roundRobinLoadBalance", "true");

props.put("user", "foo");
props.put("password", "bar");

//
// Looks like a normal MySQL JDBC url, with a
// comma-separated list of hosts, the first
// being the 'master', the rest being any number
// of slaves that the driver will load balance against
//

Connection conn =
    driver.connect("jdbc:mysql:replication://master,slave1,slave2,slave3/test",
        props);

//
// Perform read/write work on the master
// by setting the read-only flag to "false"
//

conn.setReadOnly(false);
conn.setAutoCommit(false);
conn.createStatement().executeUpdate("UPDATE some_table ...");
conn.commit();

//
// Now, do a query from a slave, the driver automatically picks one
// from the list
//

conn.setReadOnly(true);

ResultSet rs =
    conn.createStatement().executeQuery("SELECT a,b FROM alt_table");

    .....
}
}

```

Consider investigating the Load Balancing JDBC Pool ([lbpool](#)) tool, which provides a wrapper around the standard JDBC driver and enables you to use DB connection pools that includes checks for system failures and uneven load distribution. For more information, see [Load Balancing JDBC Pool \(lbpool\)](#).

22.3.5.8. Mapping MySQL Error Numbers to JDBC SQLState Codes

The table below provides a mapping of the MySQL error numbers to JDBC [SQLState](#) values.

Table 22.27. Mapping of MySQL Error Numbers to SQLStates

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1022	ER_DUP_KEY	S1000	23000
1037	ER_OUTOFMEMORY	S1001	HY001
1038	ER_OUT_OF_SORTMEMORY	S1001	HY001
1040	ER_CON_COUNT_ERROR	08004	08004
1042	ER_BAD_HOST_ERROR	08004	08S01
1043	ER_HANDSHAKE_ERROR	08004	08S01
1044	ER_DBACCESS_DENIED_ERROR	S1000	42000
1045	ER_ACCESS_DENIED_ERROR	28000	28000

Connector/J (JDBC) Reference

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1047	ER_UNKNOWN_COM_ERROR	08S01	HY000
1050	ER_TABLE_EXISTS_ERROR	S1000	42S01
1051	ER_BAD_TABLE_ERROR	42S02	42S02
1052	ER_NON_UNIQ_ERROR	S1000	23000
1053	ER_SERVER_SHUTDOWN	S1000	08S01
1054	ER_BAD_FIELD_ERROR	S0022	42S22
1055	ER_WRONG_FIELD_WITH_GROUP	S1009	42000
1056	ER_WRONG_GROUP_FIELD	S1009	42000
1057	ER_WRONG_SUM_SELECT	S1009	42000
1058	ER_WRONG_VALUE_COUNT	21S01	21S01
1059	ER_TOO_LONG_IDENT	S1009	42000
1060	ER_DUP_FIELDNAME	S1009	42S21
1061	ER_DUP_KEYNAME	S1009	42000
1062	ER_DUP_ENTRY	S1009	23000
1063	ER_WRONG_FIELD_SPEC	S1009	42000
1064	ER_PARSE_ERROR	42000	42000
1065	ER_EMPTY_QUERY	42000	42000
1066	ER_NONUNIQ_TABLE	S1009	42000
1067	ER_INVALID_DEFAULT	S1009	42000
1068	ER_MULTIPLE_PRI_KEY	S1009	42000
1069	ER_TOO_MANY_KEYS	S1009	42000
1070	ER_TOO_MANY_KEY_PARTS	S1009	42000
1071	ER_TOO_LONG_KEY	S1009	42000
1072	ER_KEY_COLUMN_DOES_NOT_EXISTS	S1009	42000
1073	ER_BLOB_USED_AS_KEY	S1009	42000
1074	ER_TOO_BIG_FIELDLENGTH	S1009	42000
1075	ER_WRONG_AUTO_KEY	S1009	42000
1080	ER_FORCING_CLOSE	S1000	08S01
1081	ER_IPSOCK_ERROR	08S01	08S01
1082	ER_NO_SUCH_INDEX	S1009	42S12
1083	ER_WRONG_FIELD_TERMINATORS	S1009	42000
1084	ER_BLOBS_AND_NO_TERMINATED	S1009	42000
1090	ER_CANT_REMOVE_ALL_FIELDS	S1000	42000
1091	ER_CANT_DROP_FIELD_OR_KEY	S1000	42000
1101	ER_BLOB_CANT_HAVE_DEFAULT	S1000	42000
1102	ER_WRONG_DB_NAME	S1000	42000
1103	ER_WRONG_TABLE_NAME	S1000	42000

Connector/J (JDBC) Reference

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1104	ER_TOO_BIG_SELECT	S1000	42000
1106	ER_UNKNOWN_PROCEDURE	S1000	42000
1107	ER_WRONG_PARAMCOUNT_TO_PROCEDURE	S1000	42000
1109	ER_UNKNOWN_TABLE	S1000	42S02
1110	ER_FIELD_SPECIFIED_TWICE	S1000	42000
1112	ER_UNSUPPORTED_EXTENSION	S1000	42000
1113	ER_TABLE_MUST_HAVE_COLUMNS	S1000	42000
1115	ER_UNKNOWN_CHARACTER_SET	S1000	42000
1118	ER_TOO_BIG_ROWSIZE	S1000	42000
1120	ER_WRONG_OUTER_JOIN	S1000	42000
1121	ER_NULL_COLUMN_IN_INDEX	S1000	42000
1129	ER_HOST_IS_BLOCKED	08004	HY000
1130	ER_HOST_NOT_PRIVILEGED	08004	HY000
1131	ER_PASSWORD_ANONYMOUS_USER	S1000	42000
1132	ER_PASSWORD_NOT_ALLOWED	S1000	42000
1133	ER_PASSWORD_NO_MATCH	S1000	42000
1136	ER_WRONG_VALUE_COUNT_ON_ROW	S1000	21S01
1138	ER_INVALID_USE_OF_NULL	S1000	42000
1139	ER_REGEXP_ERROR	S1000	42000
1140	ER_MIX_OF_GROUP_FUNC_AND_FIELDS	S1000	42000
1141	ER_NONEXISTING_GRANT	S1000	42000
1142	ER_TABLEACCESS_DENIED_ERROR	S1000	42000
1143	ER_COLUMNACCESS_DENIED_ERROR	S1000	42000
1144	ER_ILLEGAL_GRANT_FOR_TABLE	S1000	42000
1145	ER_GRANT_WRONG_HOST_OR_USER	S1000	42000
1146	ER_NO_SUCH_TABLE	S1000	42S02
1147	ER_NONEXISTING_TABLE_GRANT	S1000	42000
1148	ER_NOT_ALLOWED_COMMAND	S1000	42000
1149	ER_SYNTAX_ERROR	S1000	42000
1152	ER_ABORTING_CONNECTION	S1000	08S01
1153	ER_NET_PACKET_TOO_LARGE	S1000	08S01
1154	ER_NET_READ_ERROR_FROM_PIPE	S1000	08S01
1155	ER_NET_FCNTL_ERROR	S1000	08S01
1156	ER_NET_PACKETS_OUT_OF_ORDER	S1000	08S01
1157	ER_NET_UNCOMPRESS_ERROR	S1000	08S01
1158	ER_NET_READ_ERROR	S1000	08S01
1159	ER_NET_READ_INTERRUPTED	S1000	08S01

Connector/J (JDBC) Reference

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1160	ER_NET_ERROR_ON_WRITE	S1000	08S01
1161	ER_NET_WRITE_INTERRUPTED	S1000	08S01
1162	ER_TOO_LONG_STRING	S1000	42000
1163	ER_TABLE_CANT_HANDLE_BLOB	S1000	42000
1164	ER_TABLE_CANT_HANDLE_AUTO_INCREMENT	S1000	42000
1166	ER_WRONG_COLUMN_NAME	S1000	42000
1167	ER_WRONG_KEY_COLUMN	S1000	42000
1169	ER_DUP_UNIQUE	S1000	23000
1170	ER_BLOB_KEY_WITHOUT_LENGTH	S1000	42000
1171	ER_PRIMARY_CANT_HAVE_NULL	S1000	42000
1172	ER_TOO_MANY_ROWS	S1000	42000
1173	ER_REQUIRES_PRIMARY_KEY	S1000	42000
1177	ER_CHECK_NO_SUCH_TABLE	S1000	42000
1178	ER_CHECK_NOT_IMPLEMENTED	S1000	42000
1179	ER_CANT_DO_THIS_DURING_AN_TRANSACTION	S1000	25000
1184	ER_NEW_ABORTING_CONNECTION	S1000	08S01
1189	ER_MASTER_NET_READ	S1000	08S01
1190	ER_MASTER_NET_WRITE	S1000	08S01
1203	ER_TOO_MANY_USER_CONNECTIONS	S1000	42000
1205	ER_LOCK_WAIT_TIMEOUT	41000	41000
1207	ER_READ_ONLY_TRANSACTION	S1000	25000
1211	ER_NO_PERMISSION_TO_CREATE_USER	S1000	42000
1213	ER_LOCK_DEADLOCK	41000	40001
1216	ER_NO_REFERENCED_ROW	S1000	23000
1217	ER_ROW_IS_REFERENCED	S1000	23000
1218	ER_CONNECT_TO_MASTER	S1000	08S01
1222	ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT	S1000	21000
1226	ER_USER_LIMIT_REACHED	S1000	42000
1230	ER_NO_DEFAULT	S1000	42000
1231	ER_WRONG_VALUE_FOR_VAR	S1000	42000
1232	ER_WRONG_TYPE_FOR_VAR	S1000	42000
1234	ER_CANT_USE_OPTION_HERE	S1000	42000
1235	ER_NOT_SUPPORTED_YET	S1000	42000
1239	ER_WRONG_FK_DEF	S1000	42000
1241	ER_OPERAND_COLUMNS	S1000	21000
1242	ER_SUBQUERY_NO_1_ROW	S1000	21000
1247	ER_ILLEGAL_REFERENCE	S1000	42S22

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1248	ER_DERIVED_MUST_HAVE_ALIAS	S1000	42000
1249	ER_SELECT_REDUCED	S1000	01000
1250	ER_TABLENAME_NOT_ALLOWED_HERE	S1000	42000
1251	ER_NOT_SUPPORTED_AUTH_MODE	S1000	08004
1252	ER_SPATIAL_CANT_HAVE_NULL	S1000	42000
1253	ER_COLLATION_CHARSET_MISMATCH	S1000	42000
1261	ER_WARN_TOO_FEW_RECORDS	S1000	01000
1262	ER_WARN_TOO_MANY_RECORDS	S1000	01000
1263	ER_WARN_NULL_TO_NOTNULL	S1000	01000
1264	ER_WARN_DATA_OUT_OF_RANGE	S1000	01000
1265	ER_WARN_DATA_TRUNCATED	S1000	01000
1280	ER_WRONG_NAME_FOR_INDEX	S1000	42000
1281	ER_WRONG_NAME_FOR_CATALOG	S1000	42000
1286	ER_UNKNOWN_STORAGE_ENGINE	S1000	42000

22.3.6. JDBC Concepts

This section provides some general JDBC background.

22.3.6.1. Connecting to MySQL Using the JDBC `DriverManager` Interface

When you are using JDBC outside of an application server, the `DriverManager` class manages the establishment of Connections.

Specify to the `DriverManager` which JDBC drivers to try to make Connections with. The easiest way to do this is to use `Class.forName()` on the class that implements the `java.sql.Driver` interface. With MySQL Connector/J, the name of this class is `com.mysql.jdbc.Driver`. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

The following section of Java code shows how you might register MySQL Connector/J from the `main()` method of your application. If testing this code, first read the installation section at [Section 22.3.3, “Connector/J Installation”](#), to make sure you have connector installed correctly and the `CLASSPATH` set up. Also, ensure that MySQL is configured to accept external TCP/IP connections.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

// Notice, do not import com.mysql.jdbc.*
// or you will have problems!

public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations

            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
```

```

        // handle the error
    }
}

```

After the driver has been registered with the `DriverManager`, you can obtain a `Connection` instance that is connected to a particular database by calling `DriverManager.getConnection()`:

Example 22.1. Connector/J: Obtaining a connection from the `DriverManager`

If you have not already done so, please review the section [Section 22.3.6.1, “Connecting to MySQL Using the JDBC `DriverManager` Interface”](#) before working with these examples.

This example shows how you can obtain a `Connection` instance from the `DriverManager`. There are a few different signatures for the `getConnection()` method. Consult the API documentation that comes with your JDK for more specific information on how to use them.

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

Connection conn = null;
...
try {
    conn =
        DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                                   "user=monty&password=greatsqlldb");

    // Do something with the Connection

    ...
} catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}

```

Once a `Connection` is established, it can be used to create `Statement` and `PreparedStatement` objects, as well as retrieve metadata about the database. This is explained in the following sections.

22.3.6.2. Using JDBC `Statement` Objects to Execute SQL

`Statement` objects allow you to execute basic SQL queries and retrieve the results through the `ResultSet` class, which is described later.

To create a `Statement` instance, you call the `createStatement()` method on the `Connection` object you have retrieved using one of the `DriverManager.getConnection()` or `DataSource.getConnection()` methods described earlier.

Once you have a `Statement` instance, you can execute a `SELECT` query by calling the `executeQuery(String)` method with the SQL you want to use.

To update data in the database, use the `executeUpdate(String SQL)` method. This method returns the number of rows matched by the update statement, not the number of rows that were modified.

If you do not know ahead of time whether the SQL statement will be a `SELECT` or an `UPDATE/INSERT`, then you can use the `execute(String SQL)` method. This method will return true if the SQL query was a `SELECT`, or false if it was an `UPDATE`, `INSERT`, or `DELETE` statement. If the statement was a `SELECT` query, you can retrieve the results by calling the `getResultSet()` method. If the statement was an `UPDATE`, `INSERT`, or `DELETE` statement, you can retrieve the affected rows count by calling `getUpdateCount()` on the `Statement` instance.

Example 22.2. Connector/J: Using `java.sql.Statement` to execute a `SELECT` query

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

// assume that conn is an already created JDBC connection (see previous examples)

Statement stmt = null;
ResultSet rs = null;

try {
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");

    // or alternatively, if you don't know ahead of time that
    // the query will be a SELECT...

    if (stmt.execute("SELECT foo FROM bar")) {
        rs = stmt.getResultSet();
    }

    // Now do something with the ResultSet ....
}
catch (SQLException ex){
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
finally {
    // it is a good idea to release
    // resources in a finally{} block
    // in reverse-order of their creation
    // if they are no-longer needed

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { } // ignore

        rs = null;
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { } // ignore

        stmt = null;
    }
}
```

22.3.6.3. Using JDBC `CallableStatements` to Execute Stored Procedures

Starting with MySQL server version 5.0 when used with Connector/J 3.1.1 or newer, the `java.sql.CallableStatement` interface is fully implemented with the exception of the `getParameterMetaData()` method.

For more information on MySQL stored procedures, please refer to <http://dev.mysql.com/doc/mysql/en/stored-routines.html>.

Connector/J exposes stored procedure functionality through JDBC's `CallableStatement` interface.

**Note**

Current versions of MySQL server do not return enough information for the JDBC driver to provide result set metadata for callable statements. This means that when using `CallableStatement`, `ResultSetMetaData` may return `NULL`.

The following example shows a stored procedure that returns the value of `inOutParam` incremented by 1, and the string passed in using `inputParam` as a `ResultSet`:

Example 22.3. Connector/J: Calling Stored Procedures

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), \
                       INOUT inOutParam INT)
BEGIN
  DECLARE z INT;
  SET z = inOutParam + 1;
  SET inOutParam = z;

  SELECT inputParam;

  SELECT CONCAT('zyxw', inputParam);
END
```

To use the `demoSp` procedure with Connector/J, follow these steps:

1. Prepare the callable statement by using `Connection.prepareCall()`.

Notice that you have to use JDBC escape syntax, and that the parentheses surrounding the parameter placeholders are not optional:

Example 22.4. Connector/J: Using `Connection.prepareCall()`

```
import java.sql.CallableStatement;

...

//
// Prepare a call to the stored procedure 'demoSp'
// with two parameters
//
// Notice the use of JDBC-escape syntax ({call ...})
//

CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");

cStmt.setString(1, "abcdefg");
```

**Note**

`Connection.prepareCall()` is an expensive method, due to the metadata retrieval that the driver performs to support output parameters. For performance reasons, minimize unnecessary calls to `Connection.prepareCall()` by reusing `CallableStatement` instances in your code.

2. Register the output parameters (if any exist)

To retrieve the values of output parameters (parameters specified as `OUT` or `INOUT` when you created the stored procedure), JDBC requires that they be specified before statement execution using the various `registerOutputParameter()` methods in the `CallableStatement` interface:

Example 22.5. Connector/J: Registering output parameters

```

import java.sql.Types;
...
//
// Connector/J supports both named and indexed
// output parameters. You can register output
// parameters using either method, as well
// as retrieve output parameters using either
// method, regardless of what method was
// used to register them.
//
// The following examples show how to use
// the various methods of registering
// output parameters (you should of course
// use only one registration per parameter).
//
//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//
cStmt.registerOutParameter(2, Types.INTEGER);
//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//
cStmt.registerOutParameter("inOutParam", Types.INTEGER);
...

```

3. Set the input parameters (if any exist)

Input and in/out parameters are set as for [PreparedStatement](#) objects. However, [CallableStatement](#) also supports setting parameters by name:

Example 22.6. Connector/J: Setting [CallableStatement](#) input parameters

```

...
//
// Set a parameter by index
//
cStmt.setString(1, "abcdefg");
//
// Alternatively, set a parameter using
// the parameter name
//
cStmt.setString("inputParameter", "abcdefg");
//
// Set the 'in/out' parameter using an index
//
cStmt.setInt(2, 1);
//
// Alternatively, set the 'in/out' parameter

```

```

// by name
//
cStmt.setInt("inOutParam", 1);
...

```

4. Execute the `CallableStatement`, and retrieve any result sets or output parameters.

Although `CallableStatement` supports calling any of the `Statement` execute methods (`executeUpdate()`, `executeQuery()` or `execute()`), the most flexible method to call is `execute()`, as you do not need to know ahead of time if the stored procedure returns result sets:

Example 22.7. Connector/J: Retrieving results and output parameter values

```

...

boolean hadResults = cStmt.execute();

//
// Process all returned result sets
//

while (hadResults) {
    ResultSet rs = cStmt.getResultSet();

    // process result set
    ...

    hadResults = cStmt.getMoreResults();
}

//
// Retrieve output parameters
//
// Connector/J supports both index-based and
// name-based retrieval
//

int outputValue = cStmt.getInt(2); // index-based

outputValue = cStmt.getInt("inOutParam"); // name-based

...

```

22.3.6.4. Retrieving `AUTO_INCREMENT` Column Values through JDBC

Before version 3.0 of the JDBC API, there was no standard way of retrieving key values from databases that supported auto increment or identity columns. With older JDBC drivers for MySQL, you could always use a MySQL-specific method on the `Statement` interface, or issue the query `SELECT LAST_INSERT_ID()` after issuing an `INSERT` to a table that had an `AUTO_INCREMENT` key. Using the MySQL-specific method call isn't portable, and issuing a `SELECT` to get the `AUTO_INCREMENT` key's value requires another round-trip to the database, which isn't as efficient as possible. The following code snippets demonstrate the three different ways to retrieve `AUTO_INCREMENT` values. First, we demonstrate the use of the new JDBC 3.0 method `getGeneratedKeys()` which is now the preferred method to use if you need to retrieve `AUTO_INCREMENT` keys and have access to JDBC 3.0. The second example shows how you can retrieve the same value using a standard `SELECT LAST_INSERT_ID()` query. The final example shows how updatable result sets can retrieve the `AUTO_INCREMENT` value when using the `insertRow()` method.

Example 22.8. Connector/J: Retrieving `AUTO_INCREMENT` column values using `Statement.getGeneratedKeys()`

```
Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets assuming you have a
    // Connection 'conn' to a MySQL database already
    // available

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                                java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ( "
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //

    stmt.executeUpdate(
        "INSERT INTO autoIncTutorial (dataField) "
        + "values ('Can I Get the Auto Increment Field?')",
        Statement.RETURN_GENERATED_KEYS);

    //
    // Example of using Statement.getGeneratedKeys()
    // to retrieve the value of an auto-increment
    // value
    //

    int autoIncKeyFromApi = -1;

    rs = stmt.getGeneratedKeys();

    if (rs.next()) {
        autoIncKeyFromApi = rs.getInt(1);
    } else {

        // throw an exception from here
    }

    rs.close();

    rs = null;

    System.out.println("Key returned from getGeneratedKeys():"
        + autoIncKeyFromApi);
} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

```

    }
}

if (stmt != null) {
    try {
        stmt.close();
    } catch (SQLException ex) {
        // ignore
    }
}
}
}

```

Example 22.9. Connector/J: Retrieving `AUTO_INCREMENT` column values using `SELECT LAST_INSERT_ID()`

```

Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets.

    stmt = conn.createStatement();

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //

    stmt.executeUpdate(
        "INSERT INTO autoIncTutorial (dataField) "
        + "values ('Can I Get the Auto Increment Field?')");

    //
    // Use the MySQL LAST_INSERT_ID()
    // function to do the same thing as getGeneratedKeys()
    //

    int autoIncKeyFromFunc = -1;
    rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");

    if (rs.next()) {
        autoIncKeyFromFunc = rs.getInt(1);
    } else {
        // throw an exception from here
    }

    rs.close();

    System.out.println("Key returned from " +
        "'SELECT LAST_INSERT_ID()': " +
        autoIncKeyFromFunc);

} finally {

```

```

if (rs != null) {
    try {
        rs.close();
    } catch (SQLException ex) {
        // ignore
    }
}

if (stmt != null) {
    try {
        stmt.close();
    } catch (SQLException ex) {
        // ignore
    }
}
}

```

Example 22.10. Connector/J: Retrieving `AUTO_INCREMENT` column values in `Updatable ResultSets`

```

Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets as well as an 'updatable'
    // one, assuming you have a Connection 'conn' to
    // a MySQL database already available
    //

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                                java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Example of retrieving an AUTO INCREMENT key
    // from an updatable result set
    //

    rs = stmt.executeQuery("SELECT priKey, dataField "
        + "FROM autoIncTutorial");

    rs.moveToInsertRow();

    rs.updateString("dataField", "AUTO INCREMENT here?");
    rs.insertRow();

    //
    // the driver adds rows at the end
    //

    rs.last();

    //
    // We should now be on the row we just inserted
    //

```

```
int autoIncKeyFromRS = rs.getInt("priKey");

rs.close();

rs = null;

System.out.println("Key returned for inserted row: "
    + autoIncKeyFromRS);
} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}
```

Running the preceding example code should produce the following output:

```
Key returned from getGeneratedKeys(): 1
Key returned from SELECT LAST_INSERT_ID(): 1
Key returned for inserted row: 2
```

At times, it can be tricky to use the `SELECT LAST_INSERT_ID()` query, as that function's value is scoped to a connection. So, if some other query happens on the same connection, the value is overwritten. On the other hand, the `getGeneratedKeys()` method is scoped by the `Statement` instance, so it can be used even if other queries happen on the same connection, but not on the same `Statement` instance.

22.3.7. Connection Pooling with Connector/J

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any [thread](#) that needs them. Connection pooling can greatly increase the performance of your Java application, while reducing overall resource usage.

How Connection Pooling Works

Most applications only need a thread to have access to a JDBC connection when they are actively processing a [transaction](#), which often takes only milliseconds to complete. When not processing a transaction, the connection sits idle. Connection pooling enables the idle connection to be used by some other thread to do useful work.

In practice, when a thread needs to do work against a MySQL or other database with JDBC, it requests a connection from the pool. When the thread is finished using the connection, it returns it to the pool, so that it can be used by any other threads.

When the connection is loaned out from the pool, it is used exclusively by the thread that requested it. From a programming point of view, it is the same as if your thread called `DriverManager.getConnection()` every time it needed a JDBC connection. With connection pooling, your thread may end up using either a new connection or an already-existing connection.

Benefits of Connection Pooling

The main benefits to connection pooling are:

- Reduced connection creation time.

Although this is not usually an issue with the quick connection setup that MySQL offers compared to other databases, creating new JDBC connections still incurs networking and JDBC driver overhead that will be avoided if connections are recycled.

- Simplified programming model.

When using connection pooling, each individual thread can act as though it has created its own JDBC connection, allowing you to use straightforward JDBC programming techniques.

- Controlled resource usage.

If you create a new connection every time a thread needs one, rather than using connection pooling, your application's resource usage can be wasteful and lead to unpredictable behavior under load.

Using Connection Pooling with Connector/J

Sun has standardized the concept of connection pooling in JDBC through the JDBC 2.0 Optional interfaces, and all major application servers have implementations of these APIs that work with MySQL Connector/J.

Generally, you configure a connection pool in your application server configuration files, and access it through the Java Naming and Directory Interface (JNDI). The following code shows how you might use a connection pool from an application deployed in a J2EE application server:

Example 22.11. Connector/J: Using a connection pool with a J2EE application server

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.InitialContext;
import javax.sql.DataSource;

public class MyServletJspOrEjb {

    public void doSomething() throws Exception {
        /*
         * Create a JNDI Initial context to be able to
         * lookup the DataSource
         *
         * In production-level code, this should be cached as
         * an instance or static variable, as it can
         * be quite expensive to create a JNDI context.
         *
         * Note: This code only works when you are using servlets
         * or EJBs in a J2EE application server. If you are
         * using connection pooling in standalone Java code, you
         * will have to create/configure datasources using whatever
         * mechanisms your particular connection pooling library
         * provides.
         */

        InitialContext ctx = new InitialContext();

        /*
         * Lookup the DataSource, which will be backed by a pool
         */
    }
}
```

```
* that the application server provides. DataSource instances
* are also a good candidate for caching as an instance
* variable, as JNDI lookups can be expensive as well.
*/

DataSource ds =
    (DataSource)ctx.lookup("java:comp/env/jdbc/MySQLDB");

/*
 * The following code is what would actually be in your
 * Servlet, JSP or EJB 'service' method...where you need
 * to work with a JDBC connection.
 */

Connection conn = null;
Statement stmt = null;

try {
    conn = ds.getConnection();

    /*
     * Now, use normal JDBC programming to work with
     * MySQL, making sure to close each resource when you're
     * finished with it, which permits the connection pool
     * resources to be recovered as quickly as possible
     */

    stmt = conn.createStatement();
    stmt.execute("SOME SQL QUERY");

    stmt.close();
    stmt = null;

    conn.close();
    conn = null;
} finally {
    /*
     * close any jdbc instances here that weren't
     * explicitly closed during normal code path, so
     * that we don't 'leak' resources...
     */

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlex) {
            // ignore, as we can't do anything about it here
        }

        stmt = null;
    }

    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException sqlex) {
            // ignore, as we can't do anything about it here
        }

        conn = null;
    }
}
}
```

As shown in the example above, after obtaining the JNDI [InitialContext](#), and looking up the [DataSource](#), the rest of the code follows familiar JDBC conventions.

When using connection pooling, always make sure that connections, and anything created by them (such as statements or result sets) are closed. This rule applies no matter what happens in your code (exceptions, flow-of-control, and so forth). When these objects are closed, they can be re-used; otherwise, they will be stranded, which means that the MySQL server resources they represent (such as buffers, locks, or sockets) are tied up for some time, or in the worst case can be tied up forever.

Sizing the Connection Pool

Each connection to MySQL has overhead (memory, CPU, context switches, and so forth) on both the client and server side. Every connection limits how many resources there are available to your application as well as the MySQL server. Many of these resources will be used whether or not the connection is actually doing any useful work! Connection pools can be tuned to maximize performance, while keeping resource utilization below the point where your application will start to fail rather than just run slower.

The optimal size for the connection pool depends on anticipated load and average database transaction time. In practice, the optimal connection pool size can be smaller than you might expect. If you take Sun's Java Petstore blueprint application for example, a connection pool of 15-20 connections can serve a relatively moderate load (600 concurrent users) using MySQL and Tomcat with acceptable response times.

To correctly size a connection pool for your application, create load test scripts with tools such as Apache JMeter or The Grinder, and load test your application.

An easy way to determine a starting point is to configure your connection pool's maximum number of connections to be unbounded, run a load test, and measure the largest amount of concurrently used connections. You can then work backward from there to determine what values of minimum and maximum pooled connections give the best performance for your particular application.

Validating Connections

MySQL Connector/J can validate the connection by executing a lightweight ping against a server. In the case of load-balanced connections, this is performed against all active pooled internal connections that are retained. This is beneficial to Java applications using connection pools, as the pool can use this feature to validate connections. Depending on your connection pool and configuration, this validation can be carried out at different times:

1. Before the pool returns a connection to the application.
2. When the application returns a connection to the pool.
3. During periodic checks of idle connections.

To use this feature, specify a validation query in your connection pool that starts with `/* ping */`. Note that the syntax must be exactly as specified. This will cause the driver send a ping to the server and return a dummy lightweight result set. When using a [ReplicationConnection](#) or [LoadBalancedConnection](#), the ping will be sent across all active connections.

It is critical that the syntax be specified correctly. The syntax needs to be exact for reasons of efficiency, as this test is done for every statement that is executed:

```
protected static final String PING_MARKER = "/* ping */";
...
if (sql.charAt(0) == '/') {
if (sql.startsWith(PING_MARKER)) {
doPingInstead();
...
}
```

None of the following snippets will work, because the ping syntax is sensitive to whitespace, capitalization, and placement:

```
sql = "/* PING */ SELECT 1";
sql = "SELECT 1 /* ping*/";
sql = "/*ping*/ SELECT 1";
sql = " /* ping */ SELECT 1";
sql = "/*to ping or not to ping*/ SELECT 1";
```

All of the previous statements will issue a normal `SELECT` statement and will **not** be transformed into the lightweight ping. Further, for load-balanced connections, the statement will be executed against one connection in the internal pool, rather than validating each underlying physical connection. This results in the non-active physical connections assuming a stale state, and they may die. If Connector/J then re-balances, it might select a dead connection, resulting in an exception being passed to the application. To help prevent this, you can use `loadBalanceValidateConnectionOnSwapServer` to validate the connection before use.

If your Connector/J deployment uses a connection pool that allows you to specify a validation query, take advantage of it, but ensure that the query starts *exactly* with `/* ping */`. This is particularly important if you are using the load-balancing or replication-aware features of Connector/J, as it will help keep alive connections which otherwise will go stale and die, causing problems later.

22.3.8. Load Balancing with Connector/J

Connector/J has long provided an effective means to distribute read/write load across multiple MySQL server instances for Cluster or master-master replication deployments. Starting with Connector/J 5.1.3, you can now dynamically configure load-balanced connections, with no service outage. In-process transactions are not lost, and no application exceptions are generated if any application is trying to use that particular server instance.

There are two connection string options associated with this functionality:

- `loadBalanceConnectionGroup` – This provides the ability to group connections from different sources. This allows you to manage these JDBC sources within a single class loader in any combination you choose. If they use the same configuration, and you want to manage them as a logical single group, give them the same name. This is the key property for management: if you do not define a name (string) for `loadBalanceConnectionGroup`, you cannot manage the connections. All load-balanced connections sharing the same `loadBalanceConnectionGroup` value, regardless of how the application creates them, will be managed together.
- `loadBalanceEnableJMX` – The ability to manage the connections is exposed when you define a `loadBalanceConnectionGroup`, but if you want to manage this externally, enable JMX by setting this property to `true`. This enables a JMX implementation, which exposes the management and monitoring operations of a connection group. Further, start your application with the `-Dcom.sun.management.jmxremote` JVM flag. You can then perform connect and perform operations using a JMX client such as `jconsole`.

Once a connection has been made using the correct connection string options, a number of monitoring properties are available:

- Current active host count.
- Current active physical connection count.
- Current active logical connection count.
- Total logical connections created.
- Total transaction count.

The following management operations can also be performed:

- Add host.
- Remove host.

The JMX interface, `com.mysql.jdbc.jmx.LoadBalanceConnectionGroupManagerMBean`, has the following methods:

- `int getActiveHostCount(String group);`
- `int getTotalHostCount(String group);`
- `long getTotalLogicalConnectionCount(String group);`
- `long getActiveLogicalConnectionCount(String group);`
- `long getActivePhysicalConnectionCount(String group);`
- `long getTotalPhysicalConnectionCount(String group);`
- `long getTotalTransactionCount(String group);`
- `void removeHost(String group, String host) throws SQLException;`
- `void stopNewConnectionsToHost(String group, String host) throws SQLException;`
- `void addHost(String group, String host, boolean forExisting);`
- `String getActiveHostsList(String group);`
- `String getRegisteredConnectionGroups();`

The `getRegisteredConnectionGroups()` method returns the names of all connection groups defined in that class loader.

You can test this setup with the following code:

```
public class Test {

    private static String URL = "jdbc:mysql:loadbalance://" +
        "localhost:3306,localhost:3310/test?" +
        "loadBalanceConnectionGroup=first&loadBalanceEnableJMX=true";

    public static void main(String[] args) throws Exception {
        new Thread(new Repeater()).start();
        new Thread(new Repeater()).start();
        new Thread(new Repeater()).start();
    }

    static Connection getNewConnection() throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.jdbc.Driver");
        return DriverManager.getConnection(URL, "root", "");
    }

    static void executeSimpleTransaction(Connection c, int conn, int trans){
        try {
            c.setAutoCommit(false);
            Statement s = c.createStatement();
            s.executeQuery("SELECT SLEEP(1) /* Connection: " + conn + ", transaction: " + trans + " */");
            c.commit();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```

public static class Repeater implements Runnable {
    public void run() {
        for(int i=0; i < 100; i++){
            try {
                Connection c = getNewConnection();
                for(int j=0; j < 10; j++){
                    executeSimpleTransaction(c, i, j);
                    Thread.sleep(Math.round(100 * Math.random()));
                }
                c.close();
                Thread.sleep(100);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

After compiling, the application can be started with the `-Dcom.sun.management.jmxremote` flag, to enable remote management. `jconsole` can then be started. The `Test` main class will be listed by `jconsole`. Select this and click **Connect**. You can then navigate to the `com.mysql.jdbc.jmx.LoadBalanceConnectionGroupManager` bean. At this point, you can click on various operations and examine the returned result.

If you now had an additional instance of MySQL running on port 3309, you could ensure that Connector/J starts using it by using the `addHost()`, which is exposed in `jconsole`. Note that these operations can be performed dynamically without having to stop the application running.

For further information on the combination of load balancing and failover, see [Section 22.3.9, "Failover with Connector/J"](#).

22.3.9. Failover with Connector/J

Connector/J provides a useful load-balancing implementation for Cluster or multi-master deployments, as explained in [Section 22.3.8, "Load Balancing with Connector/J"](#). As of Connector/J 5.1.12, this same implementation is used for balancing load between read-only slaves with `ReplicationDriver`. When trying to balance workload between multiple servers, the driver has to determine when it is safe to swap servers, doing so in the middle of a transaction, for example, could cause problems. It is important not to lose state information. For this reason, Connector/J will only try to pick a new server when one of the following happens:

1. At transaction boundaries (transactions are explicitly committed or rolled back).
2. A communication exception (SQL State starting with "08") is encountered.
3. When a `SQLException` matches conditions defined by user, using the extension points defined by the `loadBalanceSQLStateFailover`, `loadBalanceSQLExceptionSubclassFailover` or `loadBalanceExceptionChecker` properties.

The third condition revolves around three new properties introduced with Connector/J 5.1.13. It allows you to control which `SQLExceptions` trigger failover.

- `loadBalanceExceptionChecker` - The `loadBalanceExceptionChecker` property is really the key. This takes a fully-qualified class name which implements the new `com.mysql.jdbc.LoadBalanceExceptionChecker` interface. This interface is very simple, and you only need to implement the following method:

```
public boolean shouldExceptionTriggerFailover(SQLException ex)
```

A `SQLException` is passed in, and a boolean returned. A value of `true` triggers a failover, `false` does not.

You can use this to implement your own custom logic. An example where this might be useful is when dealing with transient errors when using MySQL Cluster, where certain buffers may become overloaded. The following code snippet illustrates this:

```
public class NdbLoadBalanceExceptionChecker
    extends StandardLoadBalanceExceptionChecker {

    public boolean shouldExceptionTriggerFailover(SQLException ex) {
        return super.shouldExceptionTriggerFailover(ex)
            || checkNdbException(ex);
    }

    private boolean checkNdbException(SQLException ex){
        // Have to parse the message since most NDB errors
        // are mapped to the same DEMC.
        return (ex.getMessage().startsWith("Lock wait timeout exceeded") ||
            (ex.getMessage().startsWith("Got temporary error")
            && ex.getMessage().endsWith("from NDB")));
    }
}
```

The code above extends `com.mysql.jdbc.StandardLoadBalanceExceptionChecker`, which is the default implementation. There are a few convenient shortcuts built into this, for those who want to have some level of control using properties, without writing Java code. This default implementation uses the two remaining properties: `loadBalanceSQLStateFailover` and `loadBalanceSQLExceptionSubclassFailover`.

- `loadBalanceSQLStateFailover` - allows you to define a comma-delimited list of `SQLState` code prefixes, against which a `SQLException` is compared. If the prefix matches, failover is triggered. So, for example, the following would trigger a failover if a given `SQLException` starts with "00", or is "12345":

```
loadBalanceSQLStateFailover=00,12345
```

- `loadBalanceSQLExceptionSubclassFailover` - can be used in conjunction with `loadBalanceSQLStateFailover` or on its own. If you want certain subclasses of `SQLException` to trigger failover, simply provide a comma-delimited list of fully-qualified class or interface names to check against. For example, if you want all `SQLTransientConnectionExceptions` to trigger failover, you would specify:

```
loadBalanceSQLExceptionSubclassFailover=java.sql.SQLTransientConnectionException
```

While the three fail-over conditions enumerated earlier suit most situations, if `auto-commit` is enabled, Connector/J never re-balances, and continues using the same physical connection. This can be problematic, particularly when load-balancing is being used to distribute read-only load across multiple slaves. However, Connector/J can be configured to re-balance after a certain number of statements are executed, when `auto-commit` is enabled. This functionality is dependent upon the following properties:

- `loadBalanceAutoCommitStatementThreshold` – defines the number of matching statements which will trigger the driver to potentially swap physical server connections. The default value, 0, retains the behavior that connections with `auto-commit` enabled are never balanced.
- `loadBalanceAutoCommitStatementRegex` – the regular expression against which statements must match. The default value, blank, matches all statements. So, for example, using the following properties will cause Connector/J to re-balance after every third statement that contains the string "test":

```
loadBalanceAutoCommitStatementThreshold=3
loadBalanceAutoCommitStatementRegex=.*test.*
```

`loadBalanceAutoCommitStatementRegex` can prove useful in a number of situations. Your application may use temporary tables, server-side session state variables, or connection state, where letting the driver arbitrarily swap physical connections before processing is complete could cause data loss or other problems. This allows you to identify a trigger statement that is only executed when it is safe to swap physical connections.

22.3.10. Using the Connector/J Interceptor Classes

An interceptor is a software design pattern that provides a transparent way to extend or modify some aspect of a program, similar to a user exit. No recompiling is required. With Connector/J, the interceptors are enabled and disabled by updating the connection string to refer to different sets of interceptor classes that you instantiate.

The connection properties that control the interceptors are explained in [Section 22.3.5.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”](#):

- `connectionLifecycleInterceptors`, where you specify the fully qualified names of classes that implement the `com.mysql.jdbc.ConnectionLifecycleInterceptor` interface. In these kinds of interceptor classes, you might log events such as rollbacks, measure the time between transaction start and end, or count events such as calls to `setAutoCommit()`.
- `exceptionInterceptors`, where you specify the fully qualified names of classes that implement the `com.mysql.jdbc.ExceptionInterceptor` interface. In these kinds of interceptor classes, you might add extra diagnostic information to exceptions that can have multiple causes or indicate a problem with server settings. Because `exceptionInterceptors` classes are only called when handling a `SQLException` thrown from Connector/J code, they can be used even in production deployments without substantial performance overhead.
- `statementInterceptors`, where you specify the fully qualified names of classes that implement the `com.mysql.jdbc.StatementInterceptorV2` interface. In these kinds of interceptor classes, you might change or augment the processing done by certain kinds of statements, such as automatically checking for queried data in a `memcached` server, rewriting slow queries, logging information about statement execution, or route requests to remote servers.

22.3.11. Using Connector/J with Tomcat

The following instructions are based on the instructions for Tomcat-5.x, available at <http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html> which is current at the time this document was written.

First, install the `.jar` file that comes with Connector/J in `$(CATALINA_HOME)/common/lib` so that it is available to all applications installed in the container.

Next, configure the JNDI DataSource by adding a declaration resource to `$(CATALINA_HOME)/conf/server.xml` in the context that defines your web application:

```
<Context ....>
...
<Resource name="jdbc/MySQLDB"
    auth="Container"
    type="javax.sql.DataSource"/>
```

```
<ResourceParams name="jdbc/MySQLDB">
  <parameter>
    <name>factory</name>
    <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
  </parameter>

  <parameter>
    <name>maxActive</name>
    <value>10</value>
  </parameter>

  <parameter>
    <name>maxIdle</name>
    <value>5</value>
  </parameter>

  <parameter>
    <name>validationQuery</name>
    <value>SELECT 1</value>
  </parameter>

  <parameter>
    <name>testOnBorrow</name>
    <value>true</value>
  </parameter>

  <parameter>
    <name>testWhileIdle</name>
    <value>true</value>
  </parameter>

  <parameter>
    <name>timeBetweenEvictionRunsMillis</name>
    <value>10000</value>
  </parameter>

  <parameter>
    <name>minEvictableIdleTimeMillis</name>
    <value>60000</value>
  </parameter>

  <parameter>
    <name>username</name>
    <value>someuser</value>
  </parameter>

  <parameter>
    <name>password</name>
    <value>somepass</value>
  </parameter>

  <parameter>
    <name>driverClassName</name>
    <value>com.mysql.jdbc.Driver</value>
  </parameter>

  <parameter>
    <name>url</name>
    <value>jdbc:mysql://localhost:3306/test</value>
  </parameter>

</ResourceParams>
</Context>
```

Note that Connector/J 5.1.3 introduced a facility whereby, rather than use a `validationQuery` value of `SELECT 1`, it is possible to use `validationQuery` with a value set to `/* ping */`. This sends a ping to

the server which then returns a fake result set. This is a lighter weight solution. It also has the advantage that if using [ReplicationConnection](#) or [LoadBalancedConnection](#) type connections, the ping will be sent across all active connections. The following XML snippet illustrates how to select this option:

```
<parameter>
  <name>validationQuery</name>
  <value>/* ping */</value>
</parameter>
```

Note that `/* ping */` has to be specified exactly.

In general, follow the installation instructions that come with your version of Tomcat, as the way you configure datasources in Tomcat changes from time to time, and if you use the wrong syntax in your XML file, you will most likely end up with an exception similar to the following:

```
Error: java.sql.SQLException: Cannot load JDBC driver class 'null ' SQL
state: null
```

Note that the auto-loading of drivers having the `META-INF/service/java.sql.Driver` class in JDBC 4.0 causes an improper undeployment of the Connector/J driver in Tomcat on Windows. Namely, the Connector/J jar remains locked. This is an initialization problem that is not related to the driver. The possible workarounds, if viable, are as follows: use `antiResourceLocking=true` as a Tomcat Context attribute, or remove the `META-INF/` directory.

22.3.12. Using Connector/J with JBoss

These instructions cover JBoss-4.x. To make the JDBC driver classes available to the application server, copy the `.jar` file that comes with Connector/J to the `lib` directory for your server configuration (which is usually called `default`). Then, in the same configuration directory, in the subdirectory named `deploy`, create a datasource configuration file that ends with `-ds.xml`, which tells JBoss to deploy this file as a JDBC Datasource. The file should have the following contents:

```
<datasources>
  <local-tx-datasource>

    <jndi-name>MySQLDB</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/dbname</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>user</user-name>
    <password>pass</password>

    <min-pool-size>5</min-pool-size>

    <max-pool-size>20</max-pool-size>

    <idle-timeout-minutes>5</idle-timeout-minutes>

    <exception-sorter-class-name>
com.mysql.jdbc.integration.jboss.ExtendedMySQLExceptionSorter
    </exception-sorter-class-name>
    <valid-connection-checker-class-name>
com.mysql.jdbc.integration.jboss.MySqlValidConnectionChecker
    </valid-connection-checker-class-name>

  </local-tx-datasource>
</datasources>
```

22.3.13. Using Connector/J with Spring

The Spring Framework is a Java-based application framework designed for assisting in application design by providing a way to configure components. The technique used by Spring is a well known design pattern

called Dependency Injection (see [Inversion of Control Containers and the Dependency Injection pattern](#)). This article will focus on Java-oriented access to MySQL databases with Spring 2.0. For those wondering, there is a .NET port of Spring appropriately named Spring.NET.

Spring is not only a system for configuring components, but also includes support for aspect oriented programming (AOP). This is one of the main benefits and the foundation for Spring's resource and transaction management. Spring also provides utilities for integrating resource management with JDBC and Hibernate.

For the examples in this section the MySQL world sample database will be used. The first task is to set up a MySQL data source through Spring. Components within Spring use the "bean" terminology. For example, to configure a connection to a MySQL server supporting the world sample database, you might use:

```
<util:map id="dbProps">
  <entry key="db.driver" value="com.mysql.jdbc.Driver"/>
  <entry key="db.jdbcurl" value="jdbc:mysql://localhost/world"/>
  <entry key="db.username" value="myuser"/>
  <entry key="db.password" value="mypass"/>
</util:map>
```

In the above example, we are assigning values to properties that will be used in the configuration. For the datasource configuration:

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="${db.driver}"/>
  <property name="url" value="${db.jdbcurl}"/>
  <property name="username" value="${db.username}"/>
  <property name="password" value="${db.password}"/>
</bean>
```

The placeholders are used to provide values for properties of this bean. This means that you can specify all the properties of the configuration in one place instead of entering the values for each property on each bean. We do, however, need one more bean to pull this all together. The last bean is responsible for actually replacing the placeholders with the property values.

```
<bean
  class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="properties" ref="dbProps"/>
</bean>
```

Now that we have our MySQL data source configured and ready to go, we write some Java code to access it. The example below will retrieve three random cities and their corresponding country using the data source we configured with Spring.

```
// Create a new application context. this processes the Spring config
ApplicationContext ctx =
  new ClassPathXmlApplicationContext("exlappContext.xml");
// Retrieve the data source from the application context
DataSource ds = (DataSource) ctx.getBean("dataSource");
// Open a database connection using Spring's DataSourceUtils
Connection c = DataSourceUtils.getConnection(ds);
try {
  // retrieve a list of three random cities
  PreparedStatement ps = c.prepareStatement(
    "select City.Name as 'City', Country.Name as 'Country' " +
    "from City inner join Country on City.CountryCode = Country.Code " +
    "order by rand() limit 3");
  ResultSet rs = ps.executeQuery();
  while(rs.next()) {
```

```

        String city = rs.getString("City");
        String country = rs.getString("Country");
        System.out.printf("The city %s is in %s%n", city, country);
    }
} catch (SQLException ex) {
    // something has failed and we print a stack trace to analyse the error
    ex.printStackTrace();
    // ignore failure closing connection
    try { c.close(); } catch (SQLException e) { }
} finally {
    // properly release our connection
    DataSourceUtils.releaseConnection(c, ds);
}

```

This is very similar to normal JDBC access to MySQL with the main difference being that we are using `DataSourceUtils` instead of the `DriverManager` to create the connection.

While it may seem like a small difference, the implications are somewhat far reaching. Spring manages this resource in a way similar to a container managed data source in a J2EE application server. When a connection is opened, it can be subsequently accessed in other parts of the code if it is synchronized with a transaction. This makes it possible to treat different parts of your application as transactional instead of passing around a database connection.

22.3.13.1. Using `JdbcTemplate`

Spring makes extensive use of the Template method design pattern (see [Template Method Pattern](#)). Our immediate focus will be on the `JdbcTemplate` and related classes, specifically `NamedParameterJdbcTemplate`. The template classes handle obtaining and releasing a connection for data access when one is needed.

The next example shows how to use `NamedParameterJdbcTemplate` inside of a DAO (Data Access Object) class to retrieve a random city given a country code.

```

public class Ex2JdbcDao {
    /**
     * Data source reference which will be provided by Spring.
     */
    private DataSource dataSource;

    /**
     * Our query to find a random city given a country code. Notice
     * the ":country" parameter toward the end. This is called a
     * named parameter.
     */
    private String queryString = "select Name from City " +
        "where CountryCode = :country order by rand() limit 1";

    /**
     * Retrieve a random city using Spring JDBC access classes.
     */
    public String getRandomCityByCountryCode(String cntryCode) {
        // A template that permits using queries with named parameters
        NamedParameterJdbcTemplate template =
            new NamedParameterJdbcTemplate(dataSource);
        // A java.util.Map is used to provide values for the parameters
        Map params = new HashMap();
        params.put("country", cntryCode);
        // We query for an Object and specify what class we are expecting
        return (String)template.queryForObject(queryString, params, String.class);
    }

    /**
     * A JavaBean setter-style method to allow Spring to inject the data source.
     * @param dataSource
     */
}

```

```

*/
public void setDataSource(DataSource dataSource) {
    this.dataSource = dataSource;
}
}

```

The focus in the above code is on the `getRandomCityByCountryCode()` method. We pass a country code and use the `NamedParameterJdbcTemplate` to query for a city. The country code is placed in a Map with the key "country", which is the parameter is named in the SQL query.

To access this code, you need to configure it with Spring by providing a reference to the data source.

```

<bean id="dao" class="code.Ex2JdbcDao">
    <property name="dataSource" ref="dataSource"/>
</bean>

```

At this point, we can just grab a reference to the DAO from Spring and call `getRandomCityByCountryCode()`.

```

// Create the application context
ApplicationContext ctx =
new ClassPathXmlApplicationContext("ex2appContext.xml");
// Obtain a reference to our DAO
Ex2JdbcDao dao = (Ex2JdbcDao) ctx.getBean("dao");

String countryCode = "USA";

// Find a few random cities in the US
for(int i = 0; i < 4; ++i)
    System.out.printf("A random city in %s is %s\n", countryCode,
        dao.getRandomCityByCountryCode(countryCode));

```

This example shows how to use Spring's JDBC classes to completely abstract away the use of traditional JDBC classes including `Connection` and `PreparedStatement`.

22.3.13.2. Transactional JDBC Access

You might be wondering how we can add transactions into our code if we do not deal directly with the JDBC classes. Spring provides a transaction management package that not only replaces JDBC transaction management, but also enables declarative transaction management (configuration instead of code).

To use transactional database access, we will need to change the storage engine of the tables in the world database. The downloaded script explicitly creates MyISAM tables which do not support transactional semantics. The InnoDB storage engine does support transactions and this is what we will be using. We can change the storage engine with the following statements.

```

ALTER TABLE City ENGINE=InnoDB;
ALTER TABLE Country ENGINE=InnoDB;
ALTER TABLE CountryLanguage ENGINE=InnoDB;

```

A good programming practice emphasized by Spring is separating interfaces and implementations. What this means is that we can create a Java interface and only use the operations on this interface without any internal knowledge of what the actual implementation is. We will let Spring manage the implementation and with this it will manage the transactions for our implementation.

First you create a simple interface:

```

public interface Ex3Dao {
    Integer createCity(String name, String countryCode,
        String district, Integer population);
}

```

This interface contains one method that will create a new city record in the database and return the id of the new record. Next you need to create an implementation of this interface.

```
public class Ex3DaoImpl implements Ex3Dao {
    protected DataSource dataSource;
    protected SqlUpdate updateQuery;
    protected SqlFunction idQuery;

    public Integer createCity(String name, String countryCode,
        String district, Integer population) {
        updateQuery.update(new Object[] { name, countryCode,
            district, population });
        return getLastId();
    }

    protected Integer getLastId() {
        return idQuery.run();
    }
}
```

You can see that we only operate on abstract query objects here and do not deal directly with the JDBC API. Also, this is the complete implementation. All of our transaction management will be dealt with in the configuration. To get the configuration started, we need to create the DAO.

```
<bean id="dao" class="code.Ex3DaoImpl">
    <property name="dataSource" ref="dataSource"/>
    <property name="updateQuery">...</property>
    <property name="idQuery">...</property>
</bean>
```

Now you need to set up the transaction configuration. The first thing you must do is create transaction manager to manage the data source and a specification of what transaction properties are required for the [dao](#) methods.

```
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*" />
    </tx:attributes>
</tx:advice>
```

The preceding code creates a transaction manager that handles transactions for the data source provided to it. The [txAdvice](#) uses this transaction manager and the attributes specify to create a transaction for all methods. Finally you need to apply this advice with an AOP pointcut.

```
<aop:config>
    <aop:pointcut id="daoMethods"
        expression="execution(* code.Ex3Dao.*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="daoMethods"/>
</aop:config>
```

This basically says that all methods called on the [Ex3Dao](#) interface will be wrapped in a transaction. To make use of this, you only have to retrieve the [dao](#) from the application context and call a method on the [dao](#) instance.

```
Ex3Dao dao = (Ex3Dao) ctx.getBean("dao");
Integer id = dao.createCity(name, countryCode, district, pop);
```

We can verify from this that there is no transaction management happening in our Java code and it is all configured with Spring. This is a very powerful notion and regarded as one of the most beneficial features of Spring.

22.3.13.3. Connection Pooling with Spring

In many situations, such as web applications, there will be a large number of small database transactions. When this is the case, it usually makes sense to create a pool of database connections available for web requests as needed. Although MySQL does not spawn an extra process when a connection is made, there is still a small amount of overhead to create and set up the connection. Pooling of connections also alleviates problems such as collecting large amounts of sockets in the `TIME_WAIT` state.

Setting up pooling of MySQL connections with Spring is as simple as changing the data source configuration in the application context. There are a number of configurations that we can use. The first example is based on the [Jakarta Commons DBCP library](#). The example below replaces the source configuration that was based on `DriverManagerDataSource` with DBCP's `BasicDataSource`.

```
<bean id="dataSource" destroy-method="close"
  class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="${db.driver}"/>
  <property name="url" value="${db.jdbcurl}"/>
  <property name="username" value="${db.username}"/>
  <property name="password" value="${db.password}"/>
  <property name="initialSize" value="3"/>
</bean>
```

The configuration of the two solutions is very similar. The difference is that DBCP will pool connections to the database instead of creating a new connection every time one is requested. We have also set a parameter here called `initialSize`. This tells DBCP that we want three connections in the pool when it is created.

Another way to configure connection pooling is to configure a data source in our J2EE application server. Using JBoss as an example, you can set up the MySQL connection pool by creating a file called `mysql-local-ds.xml` and placing it in the `server/default/deploy` directory in JBoss. Once we have this setup, we can use JNDI to look it up. With Spring, this lookup is very simple. The data source configuration looks like this.

```
<jee:jndi-lookup id="dataSource" jndi-name="java:MySQL_DS"/>
```

22.3.14. Using Connector/J with GlassFish

This section explains how to use MySQL Connector/J with Glassfish™ Server Open Source Edition 3.0.1. Glassfish can be downloaded from the [Glassfish website](#).

Once Glassfish is installed you will need to make sure it can access MySQL Connector/J. To do this copy the MySQL Connector/J JAR file to the directory `GLASSFISH_INSTALL/glassfish/lib`. For example, copy `mysql-connector-java-5.1.12-bin.jar` to `C:\glassfishv3\glassfish\lib`. Restart the Glassfish Application Server.

You are now ready to create JDBC Connection Pools and JDBC Resources.

Creating a Connection Pool

1. In the Glassfish Administration Console, using the navigation tree navigate to **Resources, JDBC, Connection Pools**.

2. In the **JDBC Connection Pools** frame click **New**. You will enter a two step wizard.
3. In the **Name** field under **General Settings** enter the name for the connection pool, for example enter [MySQLConnPool](#).
4. In the **Resource Type** field, select [javax.sql.DataSource](#) from the drop-down listbox.
5. In the **Database Vendor** field, select [MySQL](#) from the drop-down listbox. Click **Next** to go to the next page of the wizard.
6. You can accept the default settings for General Settings, Pool Settings and Transactions for this example. Scroll down to Additional Properties.
7. In Additional Properties you will need to ensure the following properties are set:
 - **ServerName** - The server to connect to. For local testing this will be [localhost](#).
 - **User** - The user name with which to connect to MySQL.
 - **Password** - The corresponding password for the user.
 - **DatabaseName** - The database to connect to, for example the sample MySQL database [World](#).
8. Click **Finish** to exit the wizard. You will be taken to the **JDBC Connection Pools** page where all current connection pools, including the one you just created, will be displayed.
9. In the **JDBC Connection Pools** frame click on the connection pool you just created. Here you can review and edit information about the connection pool.
10. To test your connection pool click the **Ping** button at the top of the frame. A message will be displayed confirming correct operation or otherwise. If an error message is received recheck the previous steps, and ensure that MySQL Connector/J has been correctly copied into the previously specified location.

Now that you have created a connection pool you will also need to create a JDBC Resource (data source) for use by your application.

Creating a JDBC Resource

Your Java application will usually reference a data source object to establish a connection with the database. This needs to be created first using the following procedure.

- Using the navigation tree in the Glassfish Administration Console, navigate to **Resources, JDBC, JDBC Resources**. A list of resources will be displayed in the **JDBC Resources** frame.
- Click **New**. The **New JDBC Resource** frame will be displayed.
- In the **JNDI Name** field, enter the JNDI name that will be used to access this resource, for example enter [jdbc/MySQLDataSource](#).
- In the **Pool Name** field, select a connection pool you want this resource to use from the drop-down listbox.
- Optionally, you can enter a description into the **Description** field.
- Additional properties can be added if required.
- Click **OK** to create the new JDBC resource. The **JDBC Resources** frame will list all available JDBC Resources.

22.3.14.1. A Simple JSP Application with Glassfish, Connector/J and MySQL

This section shows how to deploy a simple JSP application on Glassfish, that connects to a MySQL database.

This example assumes you have already set up a suitable Connection Pool and JDBC Resource, as explained in the preceding sections. It is also assumed you have a sample database installed, such as [world](#).

The main application code, [index.jsp](#) is presented here:

```
<%@ page import="java.sql.*, javax.sql.*, java.io.*, javax.naming.*" %>
<html>
<head><title>Hello world from JSP</title></head>
<body>
<%
    InitialContext ctx;
    DataSource ds;
    Connection conn;
    Statement stmt;
    ResultSet rs;

    try {
        ctx = new InitialContext();
        ds = (DataSource) ctx.lookup("java:comp/env/jdbc/MySQLDataSource");
        //ds = (DataSource) ctx.lookup("jdbc/MySQLDataSource");
        conn = ds.getConnection();
        stmt = conn.createStatement();
        rs = stmt.executeQuery("SELECT * FROM Country");

        while(rs.next()) {
%>
            <h3>Name: <%= rs.getString("Name") %></h3>
            <h3>Population: <%= rs.getString("Population") %></h3>
<%
        }
    }
    catch (SQLException se) {
%>
        <%= se.getMessage() %>
<%
    }
    catch (NamingException ne) {
%>
        <%= ne.getMessage() %>
<%
    }
%>
</body>
</html>
```

In addition two XML files are required: [web.xml](#), and [sun-web.xml](#). There may be other files present, such as classes and images. These files are organized into the directory structure as follows:

```
index.jsp
WEB-INF
|
- web.xml
- sun-web.xml
```

The code for [web.xml](#) is:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <display-name>HelloWebApp</display-name>
  <distributable/>
  <resource-ref>
    <res-ref-name>jdbc/MySQLDataSource</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
</web-app>
```

The code for `sun-web.xml` is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server 8.1 Servlet 2.4//EN" "http://www.sun.com/xml/ns/sun-web-app"
<sun-web-app>
  <context-root>HelloWebApp</context-root>
  <resource-ref>
    <res-ref-name>jdbc/MySQLDataSource</res-ref-name>
    <jndi-name>jdbc/MySQLDataSource</jndi-name>
  </resource-ref>
</sun-web-app>
```

These XML files illustrate a very important aspect of running JDBC applications on Glassfish. On Glassfish it is important to map the string specified for a JDBC resource to its JNDI name, as set up in the Glassfish administration console. In this example, the JNDI name for the JDBC resource, as specified in the Glassfish Administration console when creating the JDBC Resource, was `jdbc/MySQLDataSource`. This must be mapped to the name given in the application. In this example the name specified in the application, `jdbc/MySQLDataSource`, and the JNDI name, happen to be the same, but this does not necessarily have to be the case. Note that the XML element `<res-ref-name>` is used to specify the name as used in the application source code, and this is mapped to the JNDI name specified using the `<jndi-name>` element, in the file `sun-web.xml`. The resource also has to be created in the `web.xml` file, although the mapping of the resource to a JNDI name takes place in the `sun-web.xml` file.

If you do not have this mapping set up correctly in the XML files you will not be able to lookup the data source using a JNDI lookup string such as:

```
ds = (DataSource) ctx.lookup("java:comp/env/jdbc/MySQLDataSource");
```

You will still be able to access the data source directly using:

```
ds = (DataSource) ctx.lookup("jdbc/MySQLDataSource");
```

With the source files in place, in the correct directory structure, you are ready to deploy the application:

1. In the navigation tree, navigate to **Applications** - the **Applications** frame will be displayed. Click **Deploy**.
2. You can now deploy an application packaged into a single WAR file from a remote client, or you can choose a packaged file or directory that is locally accessible to the server. If you are simply testing an application locally you can simply point Glassfish at the directory that contains your application, without needing to package the application into a WAR file.
3. Now select the application type from the **Type** drop-down listbox, which in this example is **Web application**.
4. Click OK.

Now, when you navigate to the **Applications** frame, you will have the option to **Launch**, **Redeploy**, or **Restart** your application. You can test your application by clicking **Launch**. The application will connect to the MySQL database and display the Name and Population of countries in the **Country** table.

22.3.14.2. A Simple Servlet with Glassfish, Connector/J and MySQL

This section describes a simple servlet that can be used in the Glassfish environment to access a MySQL database. As with the previous section, this example assumes the sample database `world` is installed.

The project is set up with the following directory structure:

```
index.html
WEB-INF
|
- web.xml
- sun-web.xml
- classes
  |
  - HelloWebServlet.java
  - HelloWebServlet.class
```

The code for the servlet, located in `HelloWebServlet.java`, is as follows:

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class HelloWebServlet extends HttpServlet {

    InitialContext ctx = null;
    DataSource ds = null;
    Connection conn = null;
    PreparedStatement ps = null;
    ResultSet rs = null;

    String sql = "SELECT Name, Population FROM Country WHERE Name=?";

    public void init () throws ServletException {
        try {
            ctx = new InitialContext();
            ds = (DataSource) ctx.lookup("java:comp/env/jdbc/MySQLDataSource");
            conn = ds.getConnection();
            ps = conn.prepareStatement(sql);
        }
        catch (SQLException se) {
            System.out.println("SQLException: "+se.getMessage());
        }
        catch (NamingException ne){
            System.out.println("NamingException: "+ne.getMessage());
        }
    }

    public void destroy () {
        try {
            if (rs != null)
                rs.close();
            if (ps != null)
                ps.close();
            if (conn != null)
                conn.close();
            if (ctx != null)
                ctx.close();
        }
        catch (SQLException se) {
            System.out.println("SQLException: "+se.getMessage());
        }
    }
}
```

```

catch (NamingException ne) {
    System.out.println("NamingException: "+ne.getMessage());
}
}

public void doPost(HttpServletRequest req, HttpServletResponse resp){
    try {
        String country_name = req.getParameter("country_name");
        resp.setContentType("text/html");
        PrintWriter writer = resp.getWriter();
        writer.println("<html><body>");
        writer.println("<p>Country: "+country_name+"</p>");
        ps.setString(1, country_name);
        rs = ps.executeQuery();
        if (!rs.next()){
            writer.println("<p>Country does not exist!</p>");
        }
        else {
            rs.beforeFirst();
            while(rs.next()) {
                writer.println("<p>Name: "+rs.getString("Name")+ "</p>");
                writer.println("<p>Population: "+rs.getString("Population")+ "</p>");
            }
        }
        writer.println("</body></html>");
        writer.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

public void doGet(HttpServletRequest req, HttpServletResponse resp){
    try {
        resp.setContentType("text/html");
        PrintWriter writer = resp.getWriter();
        writer.println("<html><body>");
        writer.println("<p>Hello from servlet doGet()</p>");
        writer.println("</body></html>");
        writer.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

In the preceding code a basic `doGet()` method is implemented, but is not used in the example. The code to establish the connection with the database is as shown in the previous example, [Section 22.3.14.1, "A Simple JSP Application with Glassfish, Connector/J and MySQL"](#), and is most conveniently located in the servlet `init()` method. The corresponding freeing of resources is located in the destroy method. The main functionality of the servlet is located in the `doPost()` method. If the user enters into the input form a country name that can be located in the database, the population of the country is returned. The code is invoked using a POST action associated with the input form. The form is defined in the file [index.html](#):

```

<html>
  <head><title>HelloWebServlet</title></head>

  <body>
    <h1>HelloWebServlet</h1>

    <p>Please enter country name:</p>

    <form action="HelloWebServlet" method="POST">
      <input type="text" name="country_name" length="50" />

```

```
<input type="submit" value="Submit" />
</form>

</body>
</html>
```

The XML files `web.xml` and `sun-web.xml` are as for the example in the preceding section, [Section 22.3.14.1, “A Simple JSP Application with Glassfish, Connector/J and MySQL”](#), no additional changes are required.

When compiling the Java source code, you will need to specify the path to the file `javaee.jar`. On Windows, this can be done as follows:

```
shell> javac -classpath c:\glassfishv3\glassfish\lib\javaee.jar HelloWebServlet.java
```

Once the code is correctly located within its directory structure, and compiled, the application can be deployed in Glassfish. This is done in exactly the same way as described in the preceding section, [Section 22.3.14.1, “A Simple JSP Application with Glassfish, Connector/J and MySQL”](#).

Once deployed the application can be launched from within the Glassfish Administration Console. Enter a country name such as “England”, and the application will return “Country does not exist!”. Enter “France”, and the application will return a population of 59225700.

22.3.15. Troubleshooting Connector/J Applications

This section explains the symptoms and resolutions for the most commonly encountered issues with applications using MySQL Connector/J.

Questions

- [23.3.15.1: \[2976\]](#) When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What is going on? I can connect just fine with the MySQL command-line client.

- [23.3.15.2: \[2977\]](#) My application throws an SQLException 'No Suitable Driver'. Why is this happening?
- [23.3.15.3: \[2977\]](#) I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

- [23.3.15.4: \[2977\]](#) I have a servlet/application that works fine for a day, and then stops working overnight
- [23.3.15.5: \[2980\]](#) I'm trying to use JDBC 2.0 updatable result sets, and I get an exception saying my result set is not updatable.
- [23.3.15.6: \[2980\]](#) I cannot connect to the MySQL server using Connector/J, and I'm sure the connection parameters are correct.

- [23.3.15.7: \[2980\]](#) I am trying to connect to my MySQL server within my application, but I get the following error and stack trace:

```
java.net.SocketException
MESSAGE: Software caused connection abort: recv failed

STACKTRACE:

java.net.SocketException: Software caused connection abort: recv failed
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(Unknown Source)
at com.mysql.jdbc.MysqlIO.readFully(MysqlIO.java:1392)
at com.mysql.jdbc.MysqlIO.readPacket(MysqlIO.java:1414)
at com.mysql.jdbc.MysqlIO.doHandshake(MysqlIO.java:625)
at com.mysql.jdbc.Connection.createNewIO(Connection.java:1926)
at com.mysql.jdbc.Connection.<init>(Connection.java:452)
at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:411)
```

- [23.3.15.8: \[2980\]](#) My application is deployed through JBoss and I am using transactions to handle the statements on the MySQL database. Under heavy loads I am getting a error and stack trace, but these only occur after a fixed period of heavy activity.
- [23.3.15.9: \[2981\]](#) When using `gcj` an `java.io.CharConversionException` is raised when working with certain character sequences.
- [23.3.15.10: \[2981\]](#) Updating a table that contains a primary key that is either `FLOAT` or compound primary key that uses `FLOAT` fails to update the table and raises an exception.
- [23.3.15.11: \[2981\]](#) You get an `ER_NET_PACKET_TOO_LARGE [3989]` exception, even though the binary blob size you want to insert using JDBC is safely below the `max_allowed_packet [519]` size.
- [23.3.15.12: \[2981\]](#) What should you do if you receive error messages similar to the following: “Communications link failure – Last packet sent to the server was X ms ago”?
- [23.3.15.13: \[2982\]](#) Why does Connector/J not reconnect to MySQL and re-issue the statement after a communication failure, instead of throwing an Exception, even though I use the `autoReconnect` connection string option?
- [23.3.15.14: \[2983\]](#) How can I use 3-byte UTF8 with Connector/J?
- [23.3.15.15: \[2983\]](#) How can I use 4-byte UTF8, `utf8mb4` with Connector/J?
- [23.3.15.16: \[2983\]](#) Using `useServerPrepStmts=false` and certain character encodings can lead to corruption when inserting BLOBs. How can this be avoided?

Questions and Answers

23.3.15.1: When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What is going on? I can connect just fine with the MySQL command-line client.

MySQL Connector/J must use TCP/IP sockets to connect to MySQL, as Java does not support Unix Domain Sockets. Therefore, when MySQL Connector/J connects to MySQL, the security manager in MySQL server will use its grant tables to determine whether the connection is permitted.

You must add the necessary security credentials to the MySQL server for this to happen, using the `GRANT` statement to your MySQL Server. See [Section 13.7.1.3, “GRANT Syntax”](#), for more information.



Note

Testing your connectivity with the `mysql` command-line client will not work unless you add the "host" flag, and use something other than `localhost` for the host. The `mysql` command-line client will use Unix domain sockets if you use the special host name `localhost`. If you are testing connectivity to `localhost`, use `127.0.0.1` as the host name instead.



Warning

Changing privileges and permissions improperly in MySQL can potentially cause your server installation to not have optimal security properties.

23.3.15.2: My application throws an SQLException 'No Suitable Driver'. Why is this happening?

There are three possible causes for this error:

- The Connector/J driver is not in your `CLASSPATH`, see [Section 22.3.3, "Connector/J Installation"](#).
- The format of your connection URL is incorrect, or you are referencing the wrong JDBC driver.
- When using `DriverManager`, the `jdbc.drivers` system property has not been populated with the location of the Connector/J driver.

23.3.15.3: I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

Either you're running an Applet, your MySQL server has been installed with the "skip-networking" option set, or your MySQL server has a firewall sitting in front of it.

Applets can only make network connections back to the machine that runs the web server that served the .class files for the applet. This means that MySQL must run on the same machine (or you must have some sort of port re-direction) for this to work. This also means that you will not be able to test applets from your local file system, you must always deploy them to a web server.

MySQL Connector/J can only communicate with MySQL using TCP/IP, as Java does not support Unix domain sockets. TCP/IP communication with MySQL might be affected if MySQL was started with the "skip-networking" flag, or if it is firewalled.

If MySQL has been started with the "skip-networking" option set (the Debian Linux package of MySQL server does this for example), you need to comment it out in the file `/etc/mysql/my.cnf` or `/etc/my.cnf`. Of course your `my.cnf` file might also exist in the `data` directory of your MySQL server, or anywhere else (depending on how MySQL was compiled for your system). Binaries created by us always look in `/etc/my.cnf` and `[datadir]/my.cnf`. If your MySQL server has been firewalled, you will need to have the firewall configured to allow TCP/IP connections from the host where your Java code is running to the MySQL server on the port that MySQL is listening to (by default, 3306).

23.3.15.4: I have a servlet/application that works fine for a day, and then stops working overnight

MySQL closes connections after 8 hours of inactivity. You either need to use a connection pool that handles stale connections or use the "autoReconnect" parameter (see [Section 22.3.5.1, "Driver/ Datasource Class Names, URL Syntax and Configuration Properties for Connector/J"](#)).

Also, catch `SQLExceptions` in your application and deal with them, rather than propagating them all the way until your application exits. This is just good programming practice. MySQL Connector/J will set the `SQLState` (see `java.sql.SQLException.getSQLState()` in your API docs) to "08S01" when it encounters network-connectivity issues during the processing of a query. Attempt to reconnect to MySQL at this point.

The following (simplistic) example shows what code that can handle these exceptions might look like:

Example 22.12. Connector/J: Example of transaction with retry logic

```
public void doBusinessOp() throws SQLException {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    //
    // How many times do you want to retry the transaction
    // (or at least _getting_ a connection)?
    //
    int retryCount = 5;

    boolean transactionCompleted = false;

    do {
        try {
            conn = getConnection(); // assume getting this from a
                                   // javax.sql.DataSource, or the
                                   // java.sql.DriverManager

            conn.setAutoCommit(false);

            //
            // Okay, at this point, the 'retry-ability' of the
            // transaction really depends on your application logic,
            // whether or not you're using autocommit (in this case
            // not), and whether you're using transactional storage
            // engines
            //
            // For this example, we'll assume that it's _not_ safe
            // to retry the entire transaction, so we set retry
            // count to 0 at this point
            //
            // If you were using exclusively transaction-safe tables,
            // or your application could recover from a connection going
            // bad in the middle of an operation, then you would not
            // touch 'retryCount' here, and just let the loop repeat
            // until retryCount == 0.
            //
            retryCount = 0;

            stmt = conn.createStatement();

            String query = "SELECT foo FROM bar ORDER BY baz";

            rs = stmt.executeQuery(query);

            while (rs.next()) {
            }

            rs.close();
            rs = null;

            stmt.close();
            stmt = null;

            conn.commit();
        }
    }
}
```

```

        conn.close();
        conn = null;

        transactionCompleted = true;
    } catch (SQLException sqlEx) {

        //
        // The two SQL states that are 'retry-able' are 08S01
        // for a communications error, and 40001 for deadlock.
        //
        // Only retry if the error was due to a stale connection,
        // communications problem or deadlock
        //
        String sqlState = sqlEx.getSQLState();

        if ("08S01".equals(sqlState) || "40001".equals(sqlState)) {
            retryCount -= 1;
        } else {
            retryCount = 0;
        }
    } finally {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException sqlEx) {
                // You'd probably want to log this...
            }
        }

        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException sqlEx) {
                // You'd probably want to log this as well...
            }
        }

        if (conn != null) {
            try {
                //
                // If we got here, and conn is not null, the
                // transaction should be rolled back, as not
                // all work has been done

                try {
                    conn.rollback();
                } finally {
                    conn.close();
                }
            } catch (SQLException sqlEx) {
                //
                // If we got an exception here, something
                // pretty serious is going on, so we better
                // pass it up the stack, rather than just
                // logging it...

                throw sqlEx;
            }
        }
    }
} while (!transactionCompleted && (retryCount > 0));
}

```



Note

Use of the `autoReconnect` option is not recommended because there is no safe method of reconnecting to the MySQL server without risking some corruption of the connection state or database state information. Instead, use a connection pool, which will enable your application to connect to the MySQL server using an available connection from the pool. The `autoReconnect` facility is deprecated, and may be removed in a future release.

23.3.15.5: I'm trying to use JDBC 2.0 updatable result sets, and I get an exception saying my result set is not updatable.

Because MySQL does not have row identifiers, MySQL Connector/J can only update result sets that have come from queries on tables that have at least one primary key, the query must select every primary key and the query can only span one table (that is, no joins). This is outlined in the JDBC specification.

Note that this issue only occurs when using updatable result sets, and is caused because Connector/J is unable to guarantee that it can identify the correct rows within the result set to be updated without having a unique reference to each row. There is no requirement to have a unique field on a table if you are using `UPDATE` or `DELETE` statements on a table where you can individually specify the criteria to be matched using a `WHERE` clause.

23.3.15.6: I cannot connect to the MySQL server using Connector/J, and I'm sure the connection parameters are correct.

Make sure that the `skip-networking` [461] option has not been enabled on your server. Connector/J must be able to communicate with your server over TCP/IP, named sockets are not supported. Also ensure that you are not filtering connections through a Firewall or other network security system. For more information, see [Section C.5.2.2, "Can't connect to \[local\] MySQL server"](#).

23.3.15.7: I am trying to connect to my MySQL server within my application, but I get the following error and stack trace:

```
java.net.SocketException
MESSAGE: Software caused connection abort: recv failed

STACKTRACE:

java.net.SocketException: Software caused connection abort: recv failed
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(Unknown Source)
at com.mysql.jdbc.MysqlIO.readFully(MysqlIO.java:1392)
at com.mysql.jdbc.MysqlIO.readPacket(MysqlIO.java:1414)
at com.mysql.jdbc.MysqlIO.doHandshake(MysqlIO.java:625)
at com.mysql.jdbc.Connection.createNewIO(Connection.java:1926)
at com.mysql.jdbc.Connection.<init>(Connection.java:452)
at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:411)
```

The error probably indicates that you are using a older version of the Connector/J JDBC driver (2.0.14 or 3.0.x) and you are trying to connect to a MySQL server with version 4.1x or newer. The older drivers are not compatible with 4.1 or newer of MySQL as they do not support the newer authentication mechanisms.

It is likely that the older version of the Connector/J driver exists within your application directory or your `CLASSPATH` includes the older Connector/J package.

23.3.15.8: My application is deployed through JBoss and I am using transactions to handle the statements on the MySQL database. Under heavy loads I am getting a error and stack trace, but these only occur after a fixed period of heavy activity.

This is a JBoss, not Connector/J, issue and is connected to the use of transactions. Under heavy loads the time taken for transactions to complete can increase, and the error is caused because you have exceeded the predefined timeout.

You can increase the timeout value by setting the `TransactionTimeout` attribute to the `TransactionManagerService` within the `/conf/jboss-service.xml` file (pre-4.0.3) or `/deploy/jta-service.xml` for JBoss 4.0.3 or later. See `TransactionTimeout` within the JBoss wiki for more information.

23.3.15.9: When using `gcj` an `java.io.CharConversionException` is raised when working with certain character sequences.

This is a known issue with `gcj` which raises an exception when it reaches an unknown character or one it cannot convert. Add `useJvmCharsetConverters=true` to your connection string to force character conversion outside of the `gcj` libraries, or try a different JDK.

23.3.15.10: Updating a table that contains a primary key that is either `FLOAT` or compound primary key that uses `FLOAT` fails to update the table and raises an exception.

Connector/J adds conditions to the `WHERE` clause during an `UPDATE` to check the old values of the primary key. If there is no match then Connector/J considers this a failure condition and raises an exception.

The problem is that rounding differences between supplied values and the values stored in the database may mean that the values never match, and hence the update fails. The issue will affect all queries, not just those from Connector/J.

To prevent this issue, use a primary key that does not use `FLOAT`. If you have to use a floating point column in your primary key use `DOUBLE` or `DECIMAL` types in place of `FLOAT`.

23.3.15.11: You get an `ER_NET_PACKET_TOO_LARGE [3989]` exception, even though the binary blob size you want to insert using JDBC is safely below the `max_allowed_packet [519]` size.

This is because the `hexEscapeBlock()` method in `com.mysql.jdbc.PreparedStatement.streamToBytes()` may almost double the size of your data.

23.3.15.12: What should you do if you receive error messages similar to the following: “Communications link failure – Last packet sent to the server was X ms ago”?

Generally speaking, this error suggests that the network connection has been closed. There can be several root causes:

- Firewalls or routers may clamp down on idle connections (the MySQL client/server protocol does not ping).
- The MySQL Server may be closing idle connections which exceed the `wait_timeout` or `interactive_timeout` threshold.

To help troubleshoot these issues, the following tips can be used. If a recent (5.1.13+) version of Connector/J is used, you will see an improved level of information compared to earlier versions. Older versions simply display the last time a packet was sent to the server, which is frequently 0 ms ago. This is of limited use, as it may be that a packet was just sent, while a packet from the server has not been received for several hours. Knowing the period of time since Connector/J last received a packet from the server is useful information, so if this is not displayed in your exception message, it is recommended that you update Connector/J.

Further, if the time a packet was last sent/received exceeds the `wait_timeout` or `interactive_timeout` threshold, this is noted in the exception message.

Although network connections can be volatile, the following can be helpful in avoiding problems:

- Ensure connections are valid when used from the connection pool. Use a query that starts with `/* ping */` to execute a lightweight ping instead of full query. Note, the syntax of the ping needs to be exactly as specified here.
- Minimize the duration a connection object is left idle while other application logic is executed.
- Explicitly validate the connection before using it if the connection has been left idle for an extended period of time.
- Ensure that `wait_timeout` and `interactive_timeout` are set sufficiently high.
- Ensure that `tcpKeepalive` is enabled.
- Ensure that any configurable firewall or router timeout settings allow for the maximum expected connection idle time.



Note

Do not expect to be able to reuse a connection without problems, if it has been lying idle for a period. If a connection is to be reused after being idle for any length of time, ensure that you explicitly test it before reusing it.

23.3.15.13: Why does Connector/J not reconnect to MySQL and re-issue the statement after a communication failure, instead of throwing an Exception, even though I use the `autoReconnect` connection string option?

There are several reasons for this. The first is transactional integrity. The MySQL Reference Manual states that “there is no safe method of reconnecting to the MySQL server without risking some corruption of the connection state or database state information”. Consider the following series of statements for example:

```
conn.createStatement().execute(
    "UPDATE checking_account SET balance = balance - 1000.00 WHERE customer='Smith'");
conn.createStatement().execute(
    "UPDATE savings_account SET balance = balance + 1000.00 WHERE customer='Smith'");
conn.commit();
```

Consider the case where the connection to the server fails after the `UPDATE` to `checking_account`. If no exception is thrown, and the application never learns about the problem, it will continue executing. However, the server did not commit the first transaction in this case, so that will get rolled back. But execution continues with the next transaction, and increases the `savings_account` balance by 1000. The application did not receive an exception, so it continued regardless, eventually committing the second transaction, as the commit only applies to the changes made in the new connection. Rather than a transfer taking place, a deposit was made in this example.

Note that running with `auto-commit` enabled does not solve this problem. When Connector/J encounters a communication problem, there is no means to determine whether the server processed the currently executing statement or not. The following theoretical states are equally possible:

- The server never received the statement, and therefore no related processing occurred on the server.
- The server received the statement, executed it in full, but the response was not received by the client.

If you are running with `auto-commit` enabled, it is not possible to guarantee the state of data on the server when a communication exception is encountered. The statement may have reached the server, or it may not. All you know is that communication failed at some point, before the client received confirmation

(or data) from the server. This does not only affect `auto-commit` statements though. If the communication problem occurred during `Connection.commit()`, the question arises of whether the transaction was committed on the server before the communication failed, or whether the server received the commit request at all.

The second reason for the generation of exceptions is that transaction-scoped contextual data may be vulnerable, for example:

- Temporary tables.
- User-defined variables.
- Server-side prepared statements.

These items are lost when a connection fails, and if the connection silently reconnects without generating an exception, this could be detrimental to the correct execution of your application.

In summary, communication errors generate conditions that may well be unsafe for Connector/J to simply ignore by silently reconnecting. It is necessary for the application to be notified. It is then for the application developer to decide how to proceed in the event of connection errors and failures.

23.3.15.14: How can I use 3-byte UTF8 with Connector/J?

To use 3-byte UTF8 with Connector/J set `characterEncoding=utf8` and set `useUnicode=true` in the connection string.

23.3.15.15: How can I use 4-byte UTF8, `utf8mb4` with Connector/J?

To use 4-byte UTF8 with Connector/J configure the MySQL server with `character_set_server=utf8mb4`. Connector/J will then use that setting as long as `characterEncoding` has not been set in the connection string. This is equivalent to autodetection of the character set.

23.3.15.16: Using `useServerPrepStmts=false` and certain character encodings can lead to corruption when inserting BLOBs. How can this be avoided?

When using certain character encodings, such as SJIS, CP932, and BIG5, it is possible that BLOB data contains characters that can be interpreted as control characters, for example, backslash, `'\'`. This can lead to corrupted data when inserting BLOBs into the database. There are two things that need to be done to avoid this:

1. Set the connection string option `useServerPrepStmts` to `true`.
2. Set `SQL_MODE` to `NO_BACKSLASH_ESCAPES`.

22.3.16. Connector/J Support

22.3.16.1. Connector/J Community Support

Oracle provides assistance to the user community by means of its mailing lists. For Connector/J related issues, you can get help from experienced users by using the MySQL and Java mailing list. Archives and subscription information is available online at <http://lists.mysql.com/java>.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [Section 1.6.1, "MySQL Mailing Lists"](#).

Community support from experienced users is also available through the [JDBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [Section 1.6.2, "MySQL Community Support at the MySQL Forums"](#).

22.3.16.2. How to Report Connector/J Bugs or Problems

The normal place to report bugs is <http://bugs.mysql.com/>, which is the address for our bugs database. This database is public, and can be browsed and searched by anyone. If you log in to the system, you will also be able to enter new reports.

If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to [<secalert_us@oracle.com>](mailto:secalert_us@oracle.com). Exception: Support customers should report all problems, including security bugs, to Oracle Support at <http://support.oracle.com/>.

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release.

This section will help you write your report correctly so that you do not waste your time doing things that may not help us much or at all.

If you have a repeatable bug report, please report it to the bugs database at <http://bugs.mysql.com/>. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

To report other problems, you can use one of the MySQL mailing lists.

Remember that it is possible for us to respond to a message containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details do not matter.

A good principle is this: If you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of Connector/J or MySQL used, and (b) not fully describing the platform on which Connector/J is installed (including the JVM version, and the platform type and version number that MySQL itself is installed on).

This is highly relevant information, and in 99 cases out of 100, the bug report is useless without it. Very often we get questions like, "Why doesn't this work for me?" Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has already been fixed in newer MySQL versions.

Sometimes the error is platform-dependent; in such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If at all possible, create a repeatable, standalone testcase that doesn't involve any third-party classes.

To streamline this process, we ship a base class for testcases with Connector/J, named `'com.mysql.jdbc.util.BaseBugReport'`. To create a testcase for Connector/J using this class, create your own class that inherits from `com.mysql.jdbc.util.BaseBugReport` and override the methods `setUp()`, `tearDown()` and `runTest()`.

In the `setUp()` method, create code that creates your tables, and populates them with any data needed to demonstrate the bug.

In the `runTest()` method, create code that demonstrates the bug using the tables and data you created in the `setUp` method.

In the `tearDown()` method, drop any tables you created in the `setUp()` method.

In any of the above three methods, use one of the variants of the `getConnection()` method to create a JDBC connection to MySQL:

- `getConnection()` - Provides a connection to the JDBC URL specified in `getUrl()`. If a connection already exists, that connection is returned, otherwise a new connection is created.
- `getNewConnection()` - Use this if you need to get a new connection for your bug report (that is, there is more than one connection involved).
- `getConnection(String url)` - Returns a connection using the given URL.
- `getConnection(String url, Properties props)` - Returns a connection using the given URL and properties.

If you need to use a JDBC URL that is different from 'jdbc:mysql://test', override the method `getUrl()` as well.

Use the `assertTrue(boolean expression)` and `assertTrue(String failureMessage, boolean expression)` methods to create conditions that must be met in your testcase demonstrating the behavior you are expecting (vs. the behavior you are observing, which is why you are most likely filing a bug report).

Finally, create a `main()` method that creates a new instance of your testcase, and calls the `run` method:

```
public static void main(String[] args) throws Exception {
    new MyBugReport().run();
}
```

Once you have finished your testcase, and have verified that it demonstrates the bug you are reporting, upload it with your bug report to <http://bugs.mysql.com/>.

22.4. MySQL Connector/C++

MySQL Connector/C++ is a MySQL database connector for C++. It lets you develop applications in C++ that connect to the MySQL Server.

MySQL Connector/C++ Benefits

Using MySQL Connector/C++ instead of the MySQL C API (MySQL Client Library) offers the following benefits for C++ users:

- Convenience of pure C++, no C function calls required.
- Supports an industry standard API, JDBC 4.0.
- Supports the object-oriented programming paradigm.
- Reduces development time.
- MySQL Connector/C++ is licensed under the GPL with the FLOSS License Exception.
- MySQL Connector/C++ is available under a commercial license upon request.

JDBC Compatibility

The MySQL Connector/C++ is compatible with the JDBC 4.0 API. However, MySQL Connector/C++ does not implement all of the JDBC 4.0 API. The MySQL Connector/C++ current version features the following classes:

- `Connection`
- `DatabaseMetaData`
- `Driver`
- `PreparedStatement`
- `ResultSet`
- `ResultSetMetaData`
- `Savepoint`
- `Statement`

The JDBC 4.0 API defines approximately 450 methods for the above mentioned classes. MySQL Connector/C++ implements around 80% of these and makes them available in the current release.

Platform Support and Prerequisites

MySQL Connector/C++ supports MySQL 5.1 and later.

MySQL Connector/C++ supports only Microsoft Visual Studio 2003 and above on Windows.

The release has been successfully compiled and tested on the following platforms:

AIX

- 5.2 (PPC32, PPC64)
- 5.3 (PPC32, PPC64)

FreeBSD

- 6.0 (x86, x86_64)

HPUX

- 11.11 (PA-RISC 32bit, PA-RISC 64bit)

Linux

- Debian 3.1 (PPC32, x86)
- FC4 (x86)
- RHEL 3 (ia64, x86, x86_64)
- RHEL 4 (ia64, x86, x86_64)
- RHEL 5 (ia64, x86, x86_64)
- SLES 9 (ia64, x86, x86_64)

- SLES 10 (ia64, x86_64)
- SuSE 10.3, (x86_64)
- Ubuntu 8.04 (x86)
- Ubuntu 8.10 (x86_64)

Mac

- MacOSX 10.3 (PPC32, PPC64)
- MacOSX 10.4 (PPC32, PPC64, x86)
- MacOSX 10.5 (PPC32, PPC64, x86, x86_64)

SunOS

- Solaris 8 (SPARC32, SPARC64, x86)
- Solaris 9 (SPARC32, SPARC64, x86)
- Solaris 10 (SPARC32, SPARC64, x86, x86_64)

Windows

- XP Professional (32bit)
- 2003 (64bit)

Future versions will run on all platforms supported by the MySQL Server.

How to Get Connector/C++

MySQL Connector/C++ Download

You can download the source code for the MySQL Connector/C++ current release at the [MySQL Connector/C++ downloads](#).

MySQL Connector/C++ Source repository

The latest development version is also available through [Launchpad](#).

Bazaar is used for the MySQL Connector/C++ code repository. You can check out the latest source code using the `bzr` command line tool:

```
shell> bzr branch lp:~mysql/mysql-connector-cpp/trunk .
```

Binary distributions

Starting with 1.0.4 Beta, binary distributions were made available in addition to source code releases. The releases available are shown below.

Microsoft Windows platform:

- Without installer, a Zip file
- MSI installer package

Other platforms:

- Compressed GNU TAR archive (tar.gz)

**Note**

Note that source packages are available for all platforms in the Compressed GNU TAR archive (tar.gz) format.

Binary and source packages can be obtained from [MySQL Connector/C++ downloads](#).

Release Status

MySQL Connector/C++ is available as a GA version. We kindly ask users and developers to try it out and provide us with feedback.

Note that MySQL Workbench is successfully using MySQL Connector/C++.

If you have any queries please [contact us](#).

Licensing

The MySQL Connector/C++ is licensed under the terms of the GPL, like most MySQL Connectors. There are special exceptions to the terms and conditions of the GPL as it is applied to this software, see FLOSS License Exception. If you need a non-GPL license for commercial distribution, please contact us.

22.4.1. Installing the MySQL Connector/C++ Binary

**Caution**

One problem that can occur is when the tools you use to build your application are not compatible with the tools used to build the binary versions of MySQL Connector/C++. Ideally, build your application with the same tools that were used to build the MySQL Connector/C++ binaries. To help with this, the following resources are provided.

All distributions contain a [README](#) file, which contains platform-specific notes. At the end of the [README](#) file contained in the binary distribution you will find the settings used to build the binaries. If you experience build-related issues on a platform, it may help to check the settings used on the platform to build the binary.

Developers using Microsoft Windows must meet the following requirements:

1. Use a supported version of Visual Studio, either Visual Studio 2005 or Visual Studio 2008.
2. Ensure that your application uses the same run time library as that used to build MySQL Connector/C++. Visual Studio 2005 builds use Microsoft.VC80.CRT (8.0.50727.762), and Visual Studio 2008 builds use Microsoft.VC90.CRT (9.0.21022.8).
3. Your application should use the same linker configuration as MySQL Connector/C++, for example use one of `/MD`, `/MDd`, `/MT`, or `/MTd`.

To use a variation of the requirements previously listed, such as a different compiler version, release configuration, or run time library, compile MySQL Connector/C++ from source using your desired settings, and then ensure that your application is

built with these same settings. Ensure that the three variables of compiler version, run time library, and run time linker configuration settings are the same for both application and MySQL Connector/C++ itself, to avoid issues.

A better solution is to build your MySQL Connector/C++ libraries from the source code, using the same tools that you use for building your application. This ensures compatibility.

Downloading MySQL Connector/C++

Binary and source packages can be obtained from [MySQL Connector/C++ downloads](#).

Archive Package

Unpack the archive into an appropriate directory. If you plan to use a dynamically linked version of MySQL Connector/C++, make sure that your system can reference the MySQL Client Library. Consult your operating system documentation on how to modify and expand the search path for libraries. In case you cannot modify the library search path it may help to copy your application, the MySQL Connector/C++ library and the MySQL Client Library into the same directory. Most systems search for libraries in the current directory.

Windows MSI Installer

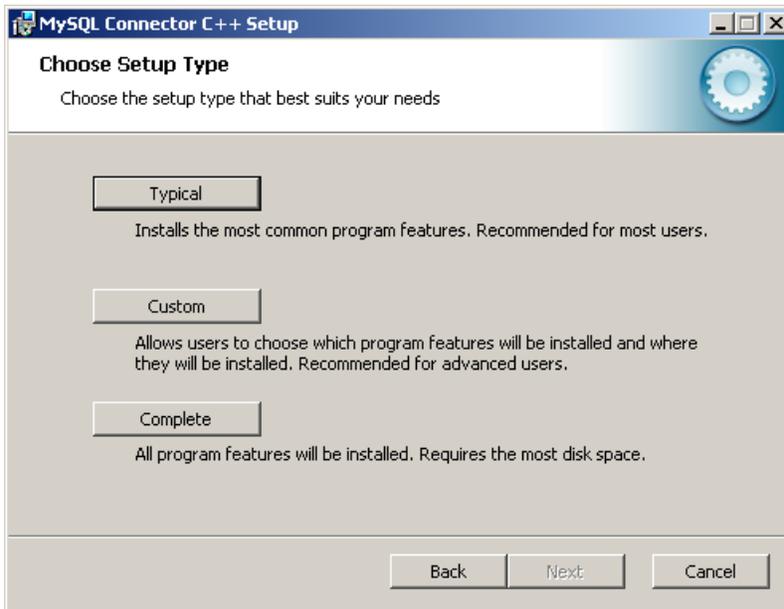
Windows users can choose between two binary packages:

1. Without installer (unzip in C:\)
2. Windows MSI Installer (x86)

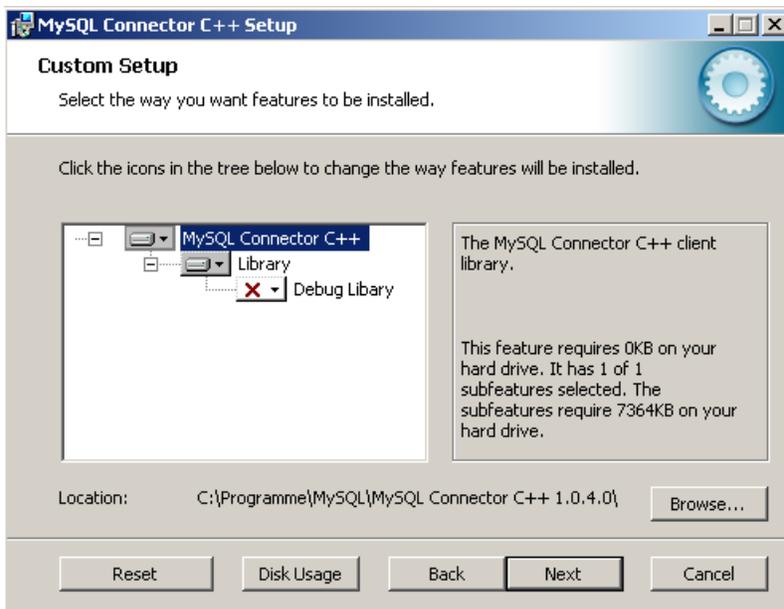
Using the MSI Installer may be the easiest solution. Running the MSI Installer does not require any administrative permissions as it simply copies files.

Figure 22.70. Windows Installer Welcome Screen



Figure 22.71. Windows Installer Overview Screen

The “Typical” installation consists of all required header files and the Release libraries. The only available “Custom” installation option enables you to install additional Debug versions of the connector libraries.

Figure 22.72. Windows Installer Custom Setup Screen

22.4.2. Installing MySQL Connector/C++ from Source

The MySQL Connector/C++ is based on the MySQL Client Library (MySQL C API). MySQL Connector/C++ is linked against the MySQL Client Library. You need the MySQL Client Library installed to compile MySQL Connector/C++.

You also need the cross-platform build tool [CMake](#) 2.4, or newer, and GLib 2.2.3 or newer installed. Check the [README](#) file included with the distribution for platform specific notes on building for Windows and SunOS.

Typically the MySQL Client Library is installed when the MySQL Server is installed. However, check your operating system documentation for other installation options.

As of MySQL Connector/C++ version 1.1.0, it is necessary to have the Boost C++ libraries 1.34.0 or newer installed. Boost is only required to build the connector, it is not required to use the connector. You can obtain Boost and its installation instructions from [the official site](#). Once Boost is installed, tell the make system where the Boost files are, by setting the define `-DBOOST_ROOT:STRING=`. This can be done when initially invoking [CMake](#), for example:

```
shell> CMake . -DBOOST_ROOT:STRING=/usr/local/boost_1_40_0
```

You might need to change `/usr/local/boost_1_40_0/` to match your installation. See the [Section 22.4.2.1, "Building MySQL Connector/C++ from Source on Unix, Solaris and Mac OS X"](#) and [Section 22.4.2.2, "Building MySQL Connector/C++ from Source on Windows"](#) for further details.

22.4.2.1. Building MySQL Connector/C++ from Source on Unix, Solaris and Mac OS X

1. Run [CMake](#) to build a [Makefile](#):

```
shell> me@host:/path/to/mysql-connector-cpp> cmake .
-- Check for working C compiler: /usr/local/bin/gcc
-- Check for working C compiler: /usr/local/bin/gcc -- works
[...]
-- Generating done
-- Build files have been written to: /path/to/mysql-connector-cpp/
```

On non-Windows systems, [CMake](#) first checks to see if the [CMake](#) variable `MYSQL_CONFIG_EXECUTABLE` is set. If it is not found [CMake](#) tries to locate `mysql_config` in the default locations.

If you have any problems with the configure process please check the troubleshooting instructions below.

2. Use `make` to build the libraries. First make sure you have a clean build:

```
shell> me@host:/path/to/mysql-connector-cpp> make clean
```

Then build the connector:

```
me@host:/path/to/mysql-connector-cpp> make
[ 1%] Building CXX object »
driver/CMakeFiles/mysqlcppconn.dir/mysql_connection.o
[ 3%] Building CXX object »
driver/CMakeFiles/mysqlcppconn.dir/mysql_constructed_resultset.o
[...]
[100%] Building CXX object examples/CMakeFiles/statement.dir/statement.o
Linking CXX executable statement
```

If all goes well, you will find the MySQL Connector/C++ library in `/path/to/cppconn/libmysqlcppconn.so`.

3. Finally make sure the header and library files are installed to their correct locations:

```
make install
```

Unless you have changed this in the configuration step, the header files are copied to the directory `/usr/local/include`. The header files copied are `mysql_connection.h` and `mysql_driver.h`.

Again, unless you have specified otherwise, the library files are copied to `/usr/local/lib`. The files copied are `libmysqlcppconn.so`, the dynamic library, and `libmysqlcppconn-static.a`, the static library.

If you encounter any errors, please first carry out the checks shown below:

1. `CMake` options: MySQL installation path, debug version and more

In case of configuration or compilation problems, check the list of `CMake` options:

```
shell> me@host:/path/to/mysql-connector-cpp> cmake -L
[...]
CMAKE_BACKWARDS_COMPATIBILITY:STRING=2.4
CMAKE_BUILD_TYPE:STRING=
CMAKE_INSTALL_PREFIX:PATH=/usr/local
EXECUTABLE_OUTPUT_PATH:PATH=
LIBRARY_OUTPUT_PATH:PATH=
MYSQLCPPCONN_GCOV_ENABLE:BOOL=0
MYSQLCPPCONN_TRACE_ENABLE:BOOL=0
MYSQL_CONFIG_EXECUTABLE:FILEPATH=/usr/bin/mysql_config
```

For example, if your MySQL Server installation path is not `/usr/local/mysql` and you want to build a debug version of the MySQL Connector/C++, use:

```
shell> me@host:/path/to/mysql-connector-cpp> cmake »
-D CMAKE_BUILD_TYPE:STRING=Debug »
-D MYSQL_CONFIG_EXECUTABLE=/path/to/my/mysql/server/bin/mysql_config .
```

2. Verify your settings with `cmake -L`:

```
shell> me@host:/path/to/mysql-connector-cpp> cmake -L
[...]
CMAKE_BACKWARDS_COMPATIBILITY:STRING=2.4
CMAKE_BUILD_TYPE:STRING=
CMAKE_INSTALL_PREFIX:PATH=/usr/local
EXECUTABLE_OUTPUT_PATH:PATH=
LIBRARY_OUTPUT_PATH:PATH=
MYSQLCPPCONN_GCOV_ENABLE:BOOL=0
MYSQLCPPCONN_TRACE_ENABLE:BOOL=0
MYSQL_CONFIG_EXECUTABLE=/path/to/my/mysql/server/bin/mysql_config
```

Proceed by carrying out a `make clean` command followed by a `make` command, as described above.

Once you have installed MySQL Connector/C++ you can carry out a quick test to check the installation. To do this you can compile and run one of the example programs, such as `examples/standalone_example.cpp`. This example is discussed in more detail later, but for now you can use it to test the connector has been correctly installed. This procedure assumes you have a working MySQL Server that you can connect to.

1. First compile the example. To do this change to the `examples` directory and type:

```
shell> g++ -o test_install -I/usr/local/include -I/usr/local/include/cppconn -Wl,-Bdynamic -lmysqlcppconn s
```

2. Make sure the dynamic library which is used in this case can be found at runtime:

```
shell> export LD_LIBRARY_PATH=/usr/local/lib
```

3. Now run the program to test your installation, substituting the appropriate host, user, password and database names for your system:

```
./test_install localhost root password database
```

You will see something similar to the following:

```
Connector/C++ standalone program example...

... running 'SELECT 'Welcome to Connector/C++' AS _message'
... MySQL replies: Welcome to Connector/C++
... say it again, MySQL
...MySQL replies: Welcome to Connector/C++

... find more at http://www.mysql.com
```

If you see any errors, take note of them and go through the troubleshooting procedures discussed earlier.

22.4.2.2. Building MySQL Connector/C++ from Source on Windows



Note

Please note the only compiler formally supported for Windows is Microsoft Visual Studio 2003 and above.

The basic steps for building the connector on Windows are the same as for Unix. It is important to use [CMake](#) 2.6.2 or newer to generate build files for your compiler and to invoke the compiler.



Note

On Windows, `mysql_config` is not present, so [CMake](#) will attempt to retrieve the location of MySQL from the environment variable `$ENV{MYSQL_DIR}`. If `MYSQL_DIR` is not set, [CMake](#) will then proceed to check for MySQL in the following locations: `$ENV{ProgramFiles}/MySQL/*/include`, and `$ENV{SystemDrive}/MySQL/*/include`.

[CMake](#) makes it easy for you to try other compilers. However, you may experience compile warnings, compile errors or linking issues not detected by Visual Studio. Patches are gratefully accepted to fix issues with other compilers.

Consult the [CMake](#) manual or check `cmake --help` to find out which build systems are supported by your [CMake](#) version:

```
C:\>cmake --help
cmake version 2.6-patch 2
Usage
[...]
Generators

The following generators are available on this platform:
  Borland Makefiles      = Generates Borland makefiles.
  MSYS Makefiles         = Generates MSYS makefiles.
  MinGW Makefiles        = Generates a make file for use with
                          mingw32-make.
  NMake Makefiles        = Generates NMake makefiles.
  Unix Makefiles         = Generates standard UNIX makefiles.
  Visual Studio 6         = Generates Visual Studio 6 project files.
  Visual Studio 7         = Generates Visual Studio .NET 2002 project
                          files.
  Visual Studio 7 .NET 2003 = Generates Visual Studio .NET 2003 project
                          files.
  Visual Studio 8 2005    = Generates Visual Studio .NET 2005 project
                          files.
  Visual Studio 8 2005 Win64 = Generates Visual Studio .NET 2005 Win64
                          project files.
```

```
Visual Studio 9 2008      = Generates Visual Studio 9 2008 project fil
Visual Studio 9 2008 Win64 = Generates Visual Studio 9 2008 Win64 proje
                           files.
[...]
```

It is likely that your [CMake](#) binary will support more compilers, known by [CMake](#) as *generators*, than supported by MySQL Connector/C++. We have built the connector using the following generators:

- Microsoft Visual Studio 8 (Visual Studio 2005)
- Microsoft Visual Studio 9 (Visual Studio 2008, Visual Studio 2008 Express)
- NMake

Please see the building instructions for Unix, Solaris and Mac OS X for troubleshooting and configuration hints.

The steps to build the connector are given below:

1. Run [CMake](#) to generate build files for your *generator*.

Visual Studio

```
C:\path_to_mysql_cpp>cmake -G "Visual Studio 9 2008"
-- Check for working C compiler: cl
-- Check for working C compiler: cl -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: cl
-- Check for working CXX compiler: cl -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- ENV{MYSQL_DIR} =
-- MySQL Include dir: C:/Programme/MySQL/MySQL Server 5.1/include
-- MySQL Library      : C:/Programs/MySQL/MySQL Server 5.1/lib/opt/mysqlclient.lib
-- MySQL Library dir: C:/Programs/MySQL/MySQL Server 5.1/lib/opt
-- MySQL CFLAGS:
-- MySQL Link flags:
-- MySQL Include dir: C:/Programs/MySQL/MySQL Server 5.1/include
-- MySQL Library dir: C:/Programs/MySQL/MySQL Server 5.1/lib/opt
-- MySQL CFLAGS:
-- MySQL Link flags:
-- Configuring cppconn
-- Configuring test cases
-- Looking for isinf
-- Looking for isinf - not found
-- Looking for isinf
-- Looking for isinf - not found.
-- Looking for finite
-- Looking for finite - not found.
-- Configuring C/J junit tests port
-- Configuring examples
-- Configuring done
-- Generating done
-- Build files have been written to: C:\path_to_mysql_cpp
C:\path_to_mysql_cpp>dir *.sln *.vcproj
[...]
```

19.11.2008	12:16	23.332	MYSQLCPPCONN.sln
[...]			
19.11.2008	12:16	27.564	ALL_BUILD.vcproj
19.11.2008	12:16	27.869	INSTALL.vcproj
19.11.2008	12:16	28.073	PACKAGE.vcproj
19.11.2008	12:16	27.495	ZERO_CHECK.vcproj

NMake

```
C:\path_to_mysql_cpp>cmake -G "NMake Makefiles"
-- The C compiler identification is MSVC
-- The CXX compiler identification is MSVC
[...]
-- Build files have been written to: C:\path_to_mysql_cpp
```

2. Use your compiler to build MySQL Connector/C++

Visual Studio - GUI

Open the newly generated project files in the Visual Studio GUI or use a Visual Studio command line to build the driver. The project files contain a variety of different configurations. Among them debug and nondebug versions.

Visual Studio - NMake

```
C:\path_to_mysql_cpp>nmake

Microsoft (R) Program Maintenance Utility Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.

Scanning dependencies of target mysqlcppconn
[ 2%] Building CXX object driver/CMakeFiles/mysqlcppconn.dir/mysql_connection.obj
mysql_connection.cpp
[...]
Linking CXX executable statement.exe
[100%] Built target statement
```

22.4.2.3. Dynamically Linking MySQL Connector/C++ against the MySQL Client Library



Note

Note this section refers to dynamic linking of the MySQL Connector/C++ with the client library, not the dynamic linking of the application to MySQL Connector/C++.

An application that uses MySQL Connector/C++ can be either statically or dynamically linked to the MySQL Connector/C++ libraries. MySQL Connector/C++ is usually statically linked to the underlying MySQL Client Library (or Connector/C). Note, that unless otherwise stated, reference to the MySQL Client Library is also taken to include Connector/C, which is a separately packaged, stand alone version of the MySQL Client Library. From MySQL Connector/C++ version 1.1.0 it is possible to also dynamically link to the underlying MySQL Client Library. The ability of MySQL Connector/C++ to dynamically link to MySQL Client Library is not enabled by default, and enabling this feature is done through a compile time option, when compiling the MySQL Connector/C++ source code.

To use the ability to dynamically link the client library to MySQL Connector/C++, define the `MYSQLCLIENT_STATIC_BINDING:BOOL` when building the MySQL Connector/C++ source code:

```
rm CMakeCache.txt
cmake -DMYSQLCLIENT_STATIC_BINDING:BOOL=1 .
make clean
make
make install
```

Note that precompiled binaries of MySQL Connector/C++ use static binding with the client library by default.

Now, in your application, when creating a connection, MySQL Connector/C++ will select and load a client library at runtime. It will choose the client library by searching defined locations and environment variables depending on the host operating system. It is also possible when creating a connection in an application to define an absolute path to the client library to be loaded at runtime. This can be convenient if you have

defined a standard location from which you want the client library to be loaded. This is sometimes done to circumvent possible conflicts with other versions of the client library that may be located on the system.

22.4.3. MySQL Connector/C++ Building Windows Applications with Microsoft Visual Studio

MySQL Connector/C++ is available as a static or dynamic library to use with your application. This section looks at how to link the library to your application.



Note

To avoid potential crashes the build configuration of MySQL Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of MySQL Connector/C++ with a debug build of the client application.

Static library

The MySQL Connector/C++ static library file is `mysqlcppconn-static.lib`. You link this library statically with your application. Also link against the files `libmysql.dll` and `libmysql.lib`. Once linking has been successfully completed, the application will require access to `libmysql.dll` at run time.

Dynamic library

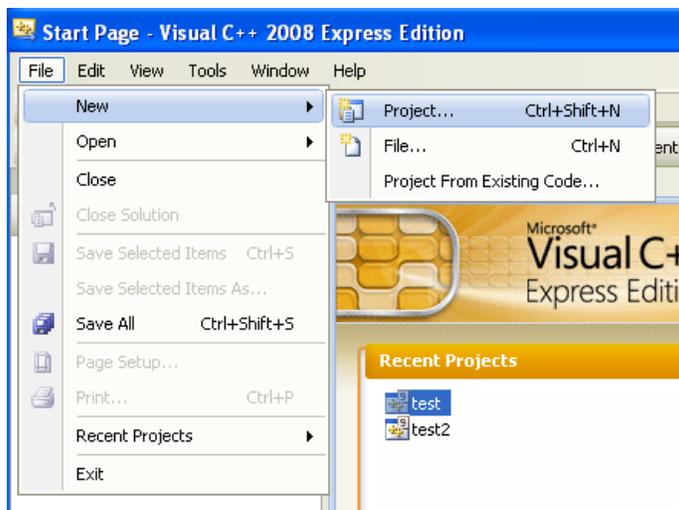
The MySQL Connector/C++ dynamic library file is `mysqlcppconn.dll`. To build your client application, link it with the file `mysqlcppconn.lib`. At run time, the application will require access to the files `mysqlcppconn.dll` and `libmysql.dll`.

Building a MySQL Connector/C++ application with Microsoft Visual Studio

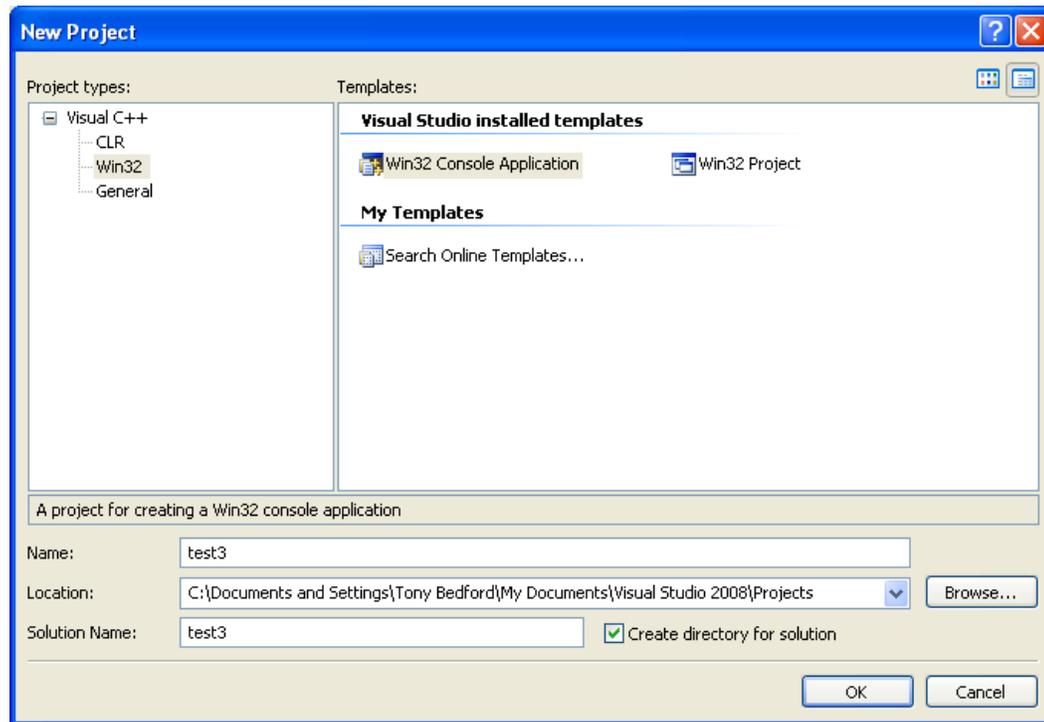
Initially, the procedure for building an application to use either the static or dynamic library is the same. You then carry out some additional steps depending on whether you are building your application to use the [static \[3002\]](#) or [dynamic \[3004\]](#) library.

1. Select **File**, **New**, **Project** from the main menu.

Figure 22.73. Creating a New Project

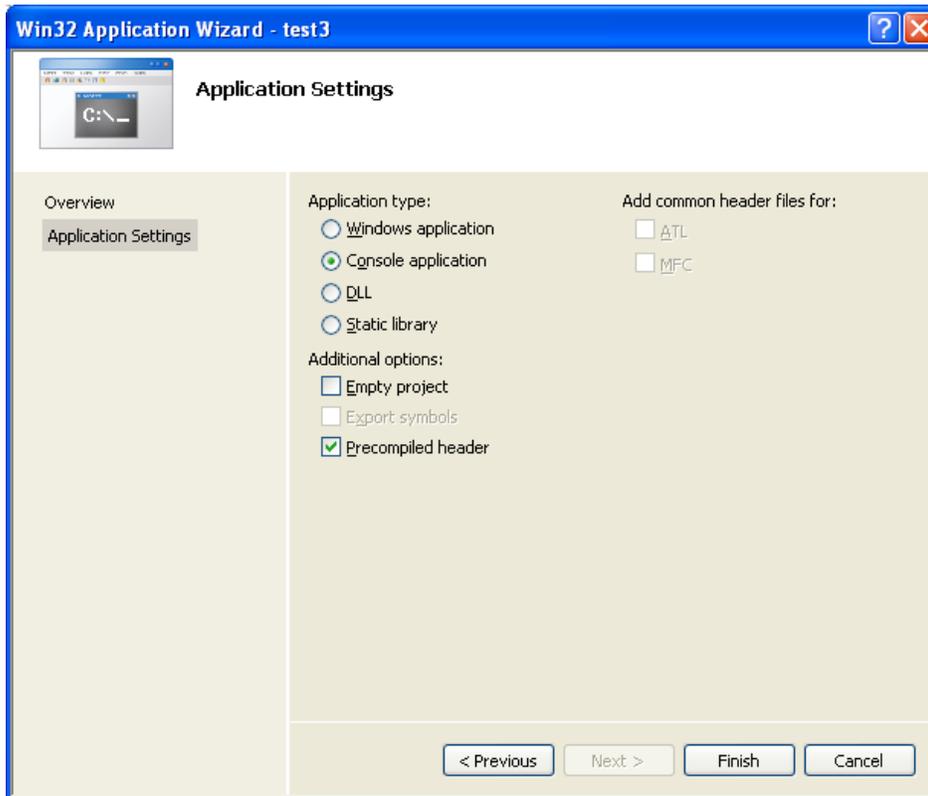


2. In the wizard, select **Visual C++**, **Win32**. From **Visual Studio Installed Templates**, select the application type **Win32 Console Application**. Enter a name for the application, then click **OK**, to move to the Win32 Application Wizard.

Figure 22.74. The New Project Dialog Box

3. In the Win32 Application Wizard, click **Application Settings** and ensure the defaults are selected. The radio button **Console application** and the check box **Precompiled headers** are selected. Click **Finish** to close the wizard.

Figure 22.75. The Win32 Application Wizard



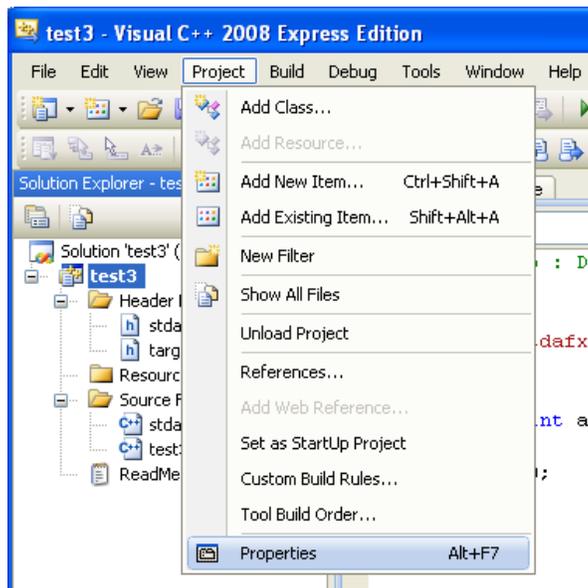
4. From the drop down list box on the toolbar, change from the default **Debug** build to the **Release** build.

Figure 22.76. Selecting the Release Build



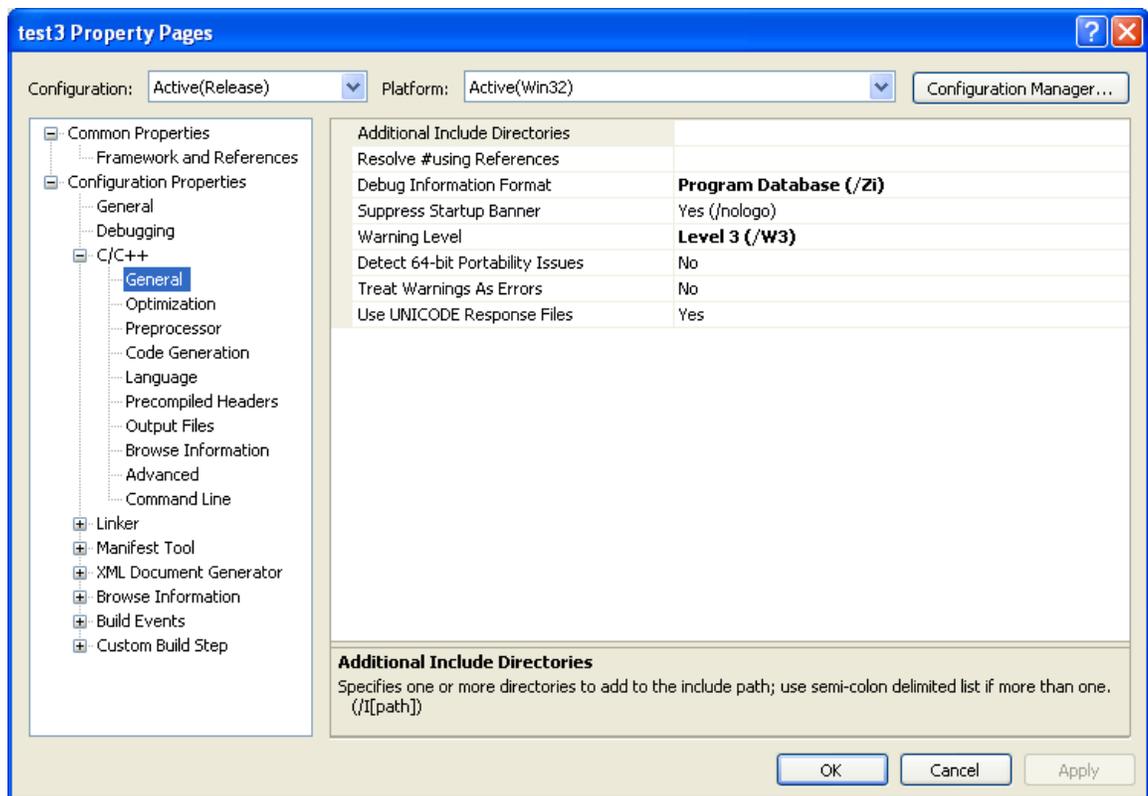
5. From the main menu select Project, Properties. This can also be accessed using the hot key **ALT + F7**.

Figure 22.77. Selecting Project Properties from the Main Menu



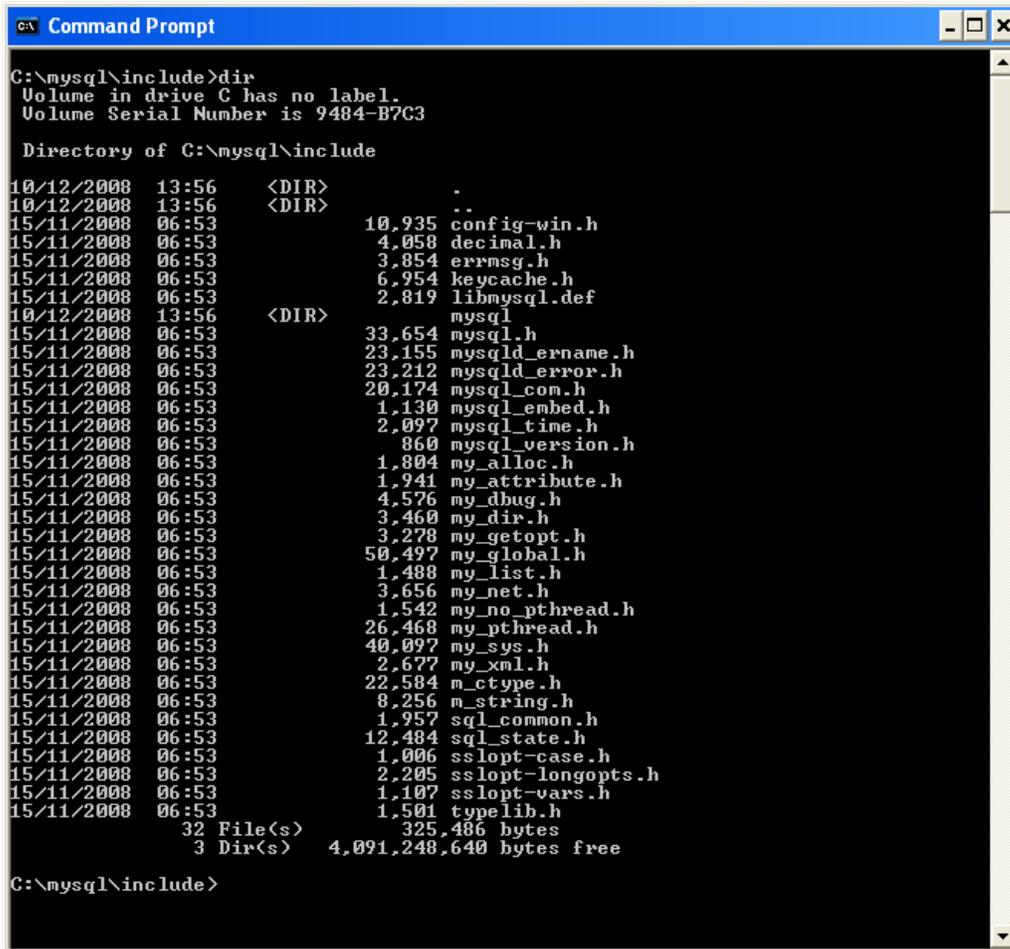
6. Under **Configuration Properties**, open the tree view.
7. Select **C++**, **General** in the tree view.

Figure 22.78. Setting Properties



- Now ensure that Visual Studio can find the MySQL include directory. This directory includes header files that can optionally be installed when installing MySQL Server.

Figure 22.79. MySQL Include Directory



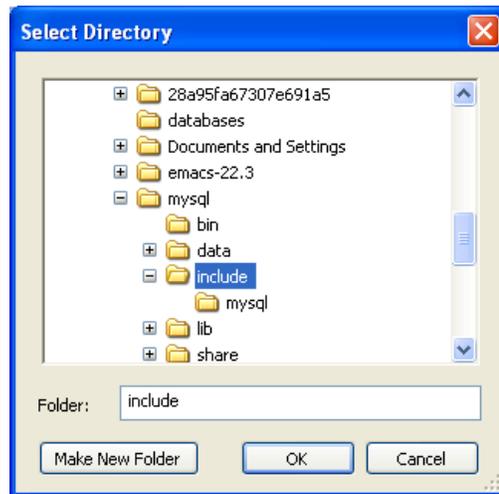
```
Command Prompt
C:\mysql\include>dir
Volume in drive C has no label.
Volume Serial Number is 9484-B7C3

Directory of C:\mysql\include

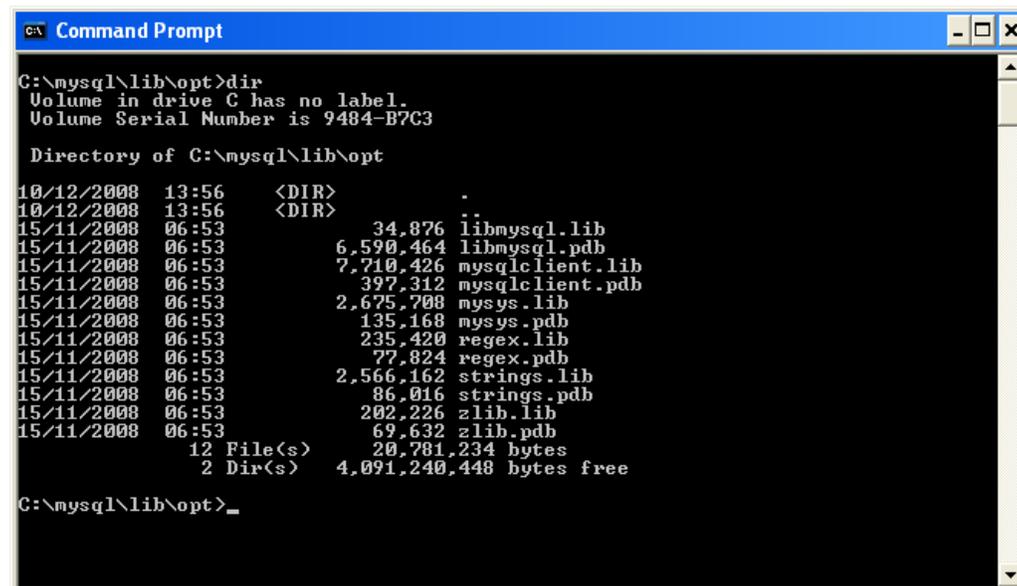
10/12/2008  13:56    <DIR>          .
10/12/2008  13:56    <DIR>          ..
15/11/2008  06:53                10,935  config-win.h
15/11/2008  06:53                4,058  decimal.h
15/11/2008  06:53                3,854  errmsg.h
15/11/2008  06:53                6,954  keycache.h
15/11/2008  06:53                2,819  libmysql.def
10/12/2008  13:56    <DIR>          mysql
15/11/2008  06:53            33,654  mysql.h
15/11/2008  06:53            23,155  mysqld_ename.h
15/11/2008  06:53            23,212  mysqld_error.h
15/11/2008  06:53            20,174  mysql_com.h
15/11/2008  06:53             1,130  mysql_embed.h
15/11/2008  06:53             2,097  mysql_time.h
15/11/2008  06:53             860   mysql_version.h
15/11/2008  06:53             1,804  my_alloc.h
15/11/2008  06:53             1,941  my_attribute.h
15/11/2008  06:53             4,576  my_debug.h
15/11/2008  06:53             3,460  my_dir.h
15/11/2008  06:53             3,278  my_getopt.h
15/11/2008  06:53            50,497  my_global.h
15/11/2008  06:53             1,488  my_list.h
15/11/2008  06:53             3,656  my_net.h
15/11/2008  06:53             1,542  my_no_pthread.h
15/11/2008  06:53            26,468  my_pthread.h
15/11/2008  06:53            40,097  my_sys.h
15/11/2008  06:53             2,677  my_xml.h
15/11/2008  06:53            22,584  mctype.h
15/11/2008  06:53             8,256  mstring.h
15/11/2008  06:53             1,957  sql_common.h
15/11/2008  06:53            12,484  sql_state.h
15/11/2008  06:53             1,006  sslopt-case.h
15/11/2008  06:53             2,205  sslopt-longopts.h
15/11/2008  06:53             1,107  sslopt-vars.h
15/11/2008  06:53             1,501  typelib.h
          32 File(s)          325,486 bytes
           3 Dir(s)      4,091,248,640 bytes free

C:\mysql\include>
```

- In the **Additional Include Directories** text field, add the MySQL `include/` directory.

Figure 22.80. Select Directory Dialog

10. Also set the location of additional libraries that Visual Studio needs to build the application. These are located in the MySQL `lib/opt` directory, a subdirectory of the MySQL Server installation directory.

Figure 22.81. Typical Contents of MySQL lib/opt Directory

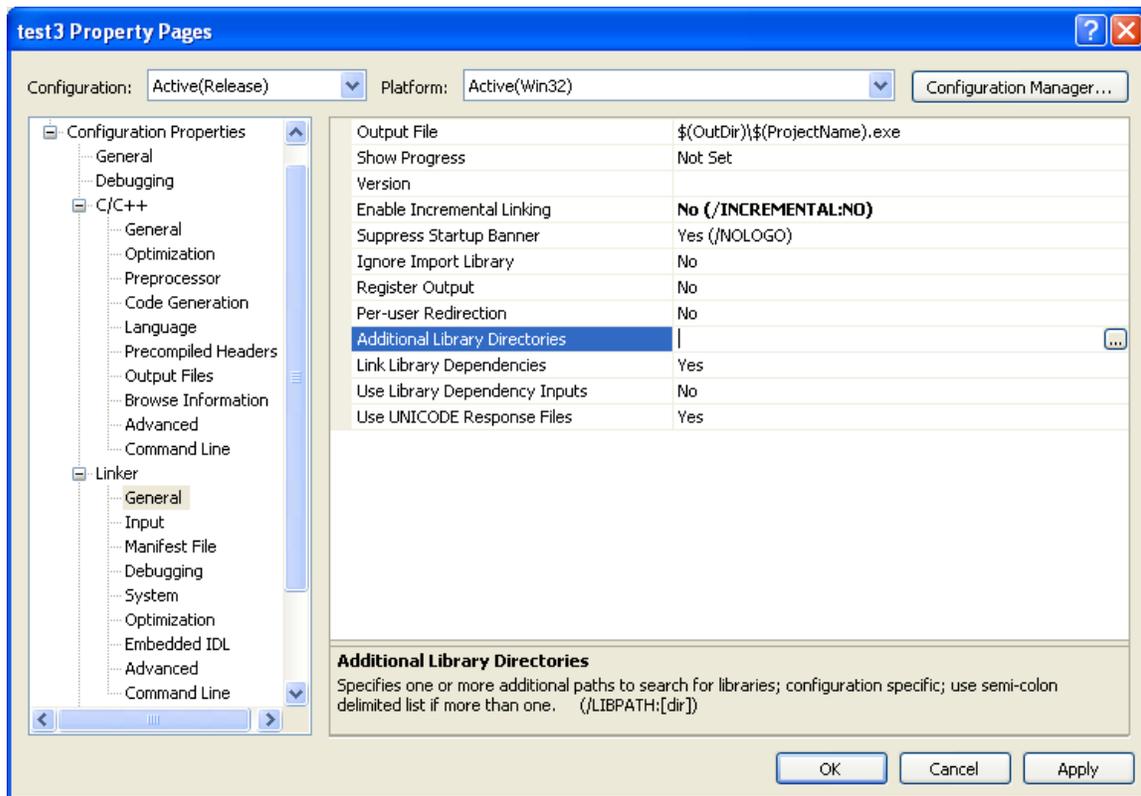
```
C:\mysql\lib\opt>dir
Volume in drive C has no label.
Volume Serial Number is 9484-B7C3

Directory of C:\mysql\lib\opt

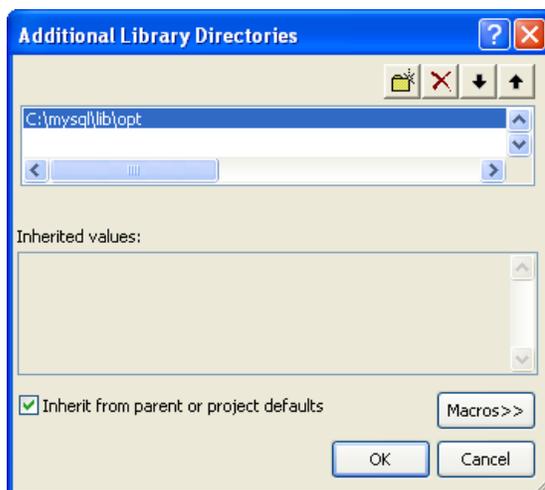
10/12/2008  13:56    <DIR>          .
10/12/2008  13:56    <DIR>          ..
15/11/2008  06:53             34,876  libmysql.lib
15/11/2008  06:53       6,590,464  libmysql.pdb
15/11/2008  06:53       7,710,426  mysqlclient.lib
15/11/2008  06:53       397,312   mysqlclient.pdb
15/11/2008  06:53       2,675,708  mysys.lib
15/11/2008  06:53       135,168   mysys.pdb
15/11/2008  06:53       235,420   regex.lib
15/11/2008  06:53       77,824    regex.pdb
15/11/2008  06:53       2,566,162  strings.lib
15/11/2008  06:53       86,016    strings.pdb
15/11/2008  06:53       202,226   zlib.lib
15/11/2008  06:53       69,632    zlib.pdb
                12 File(s)      20,781,234 bytes
                2 Dir(s)  4,091,240,448 bytes free

C:\mysql\lib\opt>
```

11. In the tree view, open **Linker, General, Additional Library Directories**.

Figure 22.82. Additional Library Directories

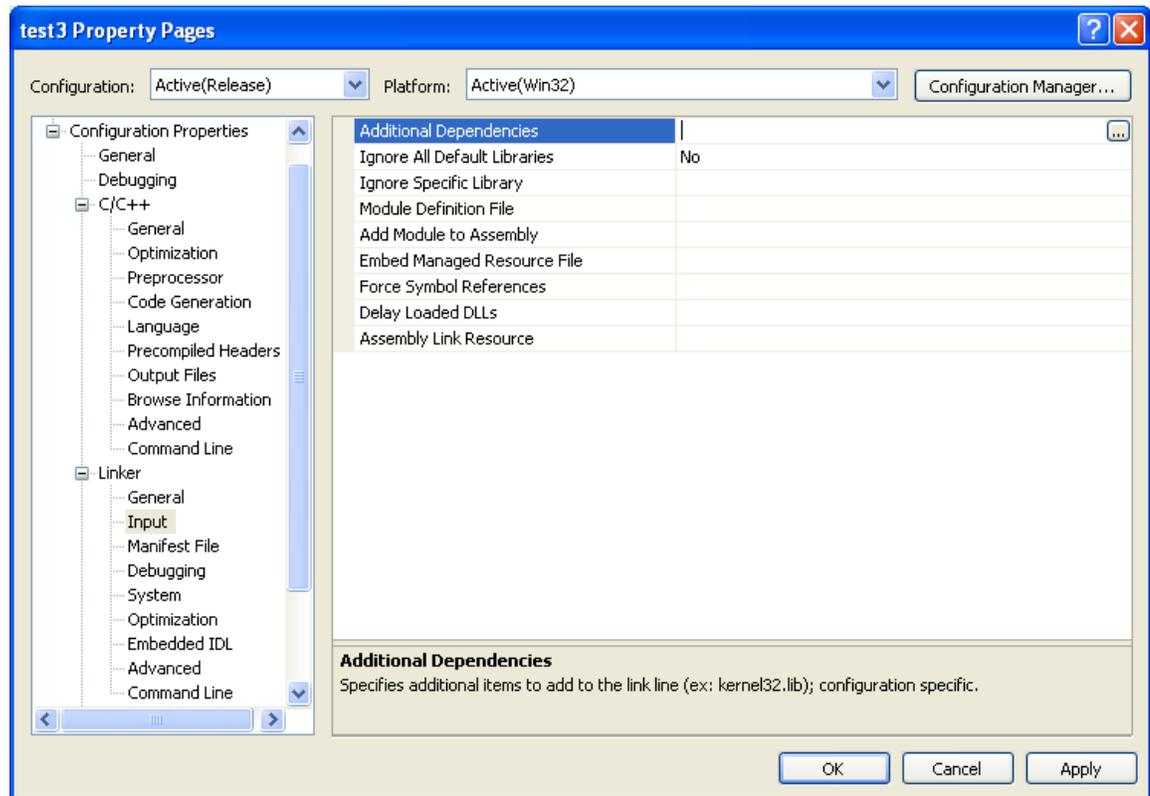
12. Add the `lib/opt` directory into the **Additional Library Directories** text field. This enables the library file `libmysql.lib` to be found.

Figure 22.83. Additional Library Directories Dialog

The remaining steps depend on whether you are building an application to use the MySQL Connector/C++ static or dynamic library. If you are building your application to use the dynamic library [go here \[3004\]](#). If you are building your application to use the static library, carry out the following steps:

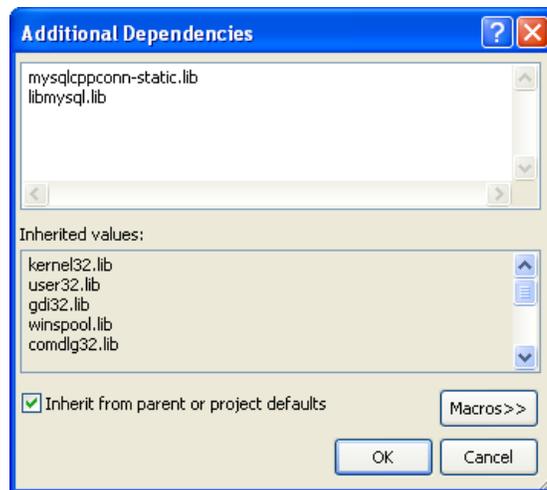
1. Then open **Linker, Input, Additional Dependencies**.

Figure 22.84. Additional Dependencies



2. Enter `mysqlcppconn-static.lib` and `libmysql.lib`.

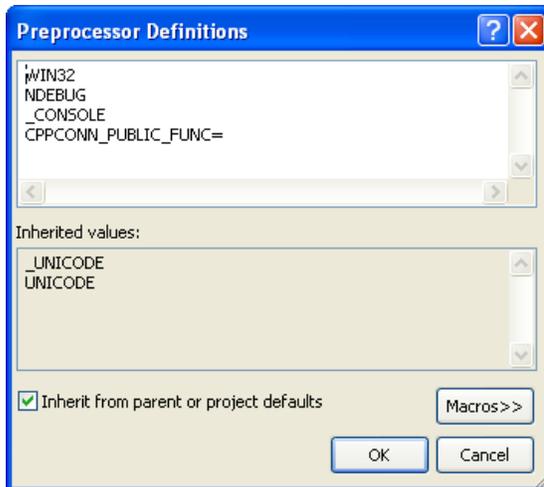
Figure 22.85. Adding Additional Dependencies



3. By default `CPPCONN_PUBLIC_FUNC` is defined to declare functions to be compatible with an application that calls a DLL. If building an application to call the static library, ensure that function prototypes are compatible with this. In this case, define `CPPCONN_PUBLIC_FUNC` to be an empty string, so that functions are declared with the correct prototype.

In the **Project, Properties** tree view, under **C++**, **Preprocessor**, enter `CPPCONN_PUBLIC_FUNC=` into the **Preprocessor Definitions** text field.

Figure 22.86. Setting the CPPCONN_PUBLIC_FUNC Define



Note

Make sure you enter `CPPCONN_PUBLIC_FUNC=` and not `CPPCONN_PUBLIC_FUNC`, so that it is defined as an empty string.

If building an application to use the MySQL Connector/C++ dynamically linked library carry out these steps:

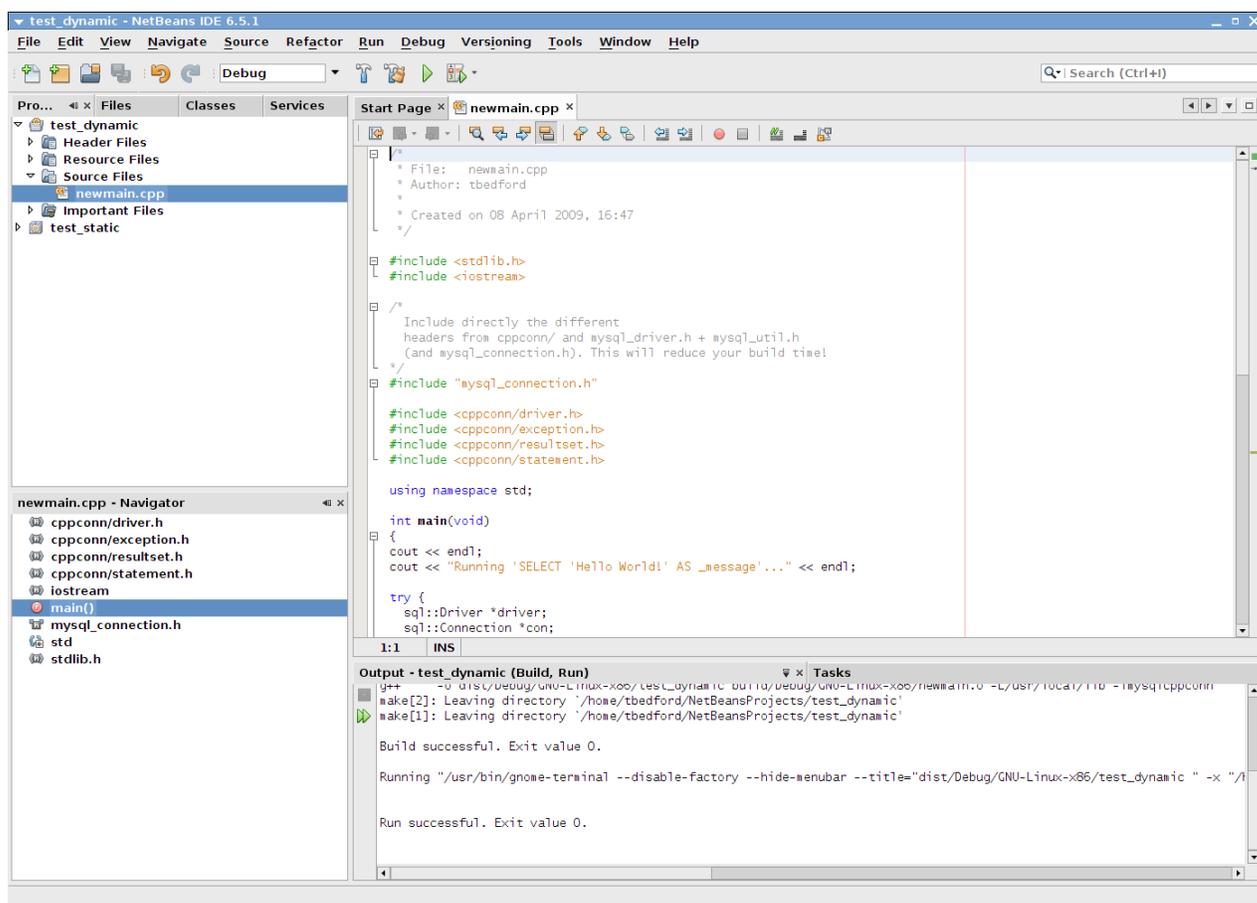
1. Under **Linker, Input**, add `mysqlcppconn.lib` into the **Additional Dependencies** text field.
2. `mysqlcppconn.dll` must be in the same directory as the application executable, or somewhere on the system's path, so that the application can access the MySQL Connector/C++ Dynamic Linked Library at runtime.

Copy `mysqlcppconn.dll` to the same directory as the application. Alternatively, extend the `PATH` environment variable using `SET PATH=%PATH%;C:\path\to\cpp`. Alternatively, you can copy `mysqlcppconn.dll` to the Windows installation Directory, typically `c:\windows`.

22.4.4. MySQL Connector/C++ Building Linux Applications with NetBeans

This section describes how to build MySQL Connector/C++ applications for Linux using the NetBeans IDE.

Figure 22.87. The NetBeans IDE



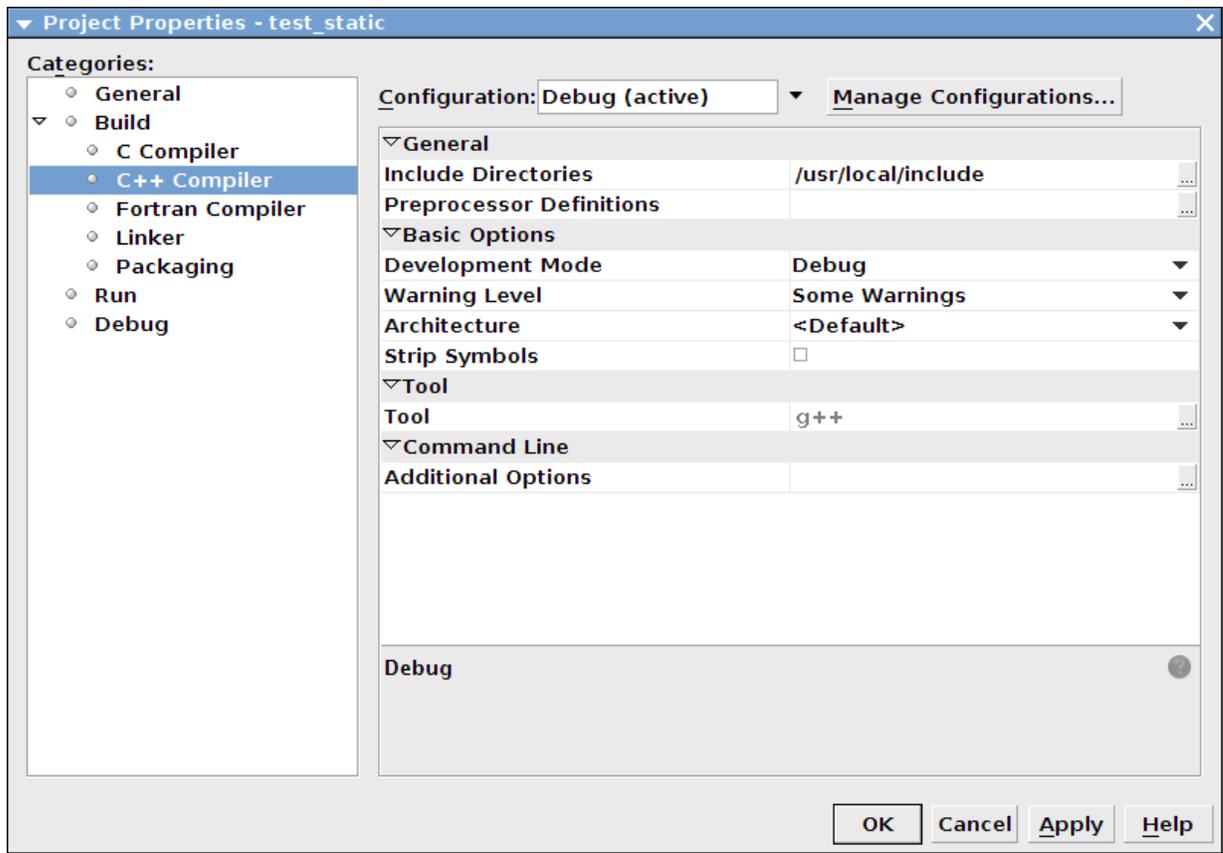
Note

To avoid potential crashes the build configuration of MySQL Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of MySQL Connector/C++ with a debug build of the client application.

1. The first step of building your application is to create a new project. Select **File**, **New Project**. Choose a **C/C++ Application** and click **Next**.
2. Give the project a name and click **Finish**. A new project is created.
3. In the **Projects** tab, right-click **Source Files** and select **New**, then **Main C++ File...**
4. Change the filename, or simply select the defaults and click **Finish** to add the new file to the project.
5. Now add some working code to your main source file. Explore your MySQL Connector/C++ installation and navigate to the [examples](#) directory.
6. Select a suitable example, such as [standalone_example_docs1.cpp](#). Copy all the code in this file, and use it to replace the code in your existing main source file. Amend the code to reflect the connection properties required for your test database. You now have a working example that will access a MySQL database using MySQL Connector/C++.
7. At this point, NetBeans is showing some errors in the source code. Direct NetBeans to the necessary header files to include. Select **File**, **Project Properties** from the main menu.

8. In the **Categories:** tree view panel, navigate to **Build, C++ Compiler**.
9. In the **General** panel, select **Include Directories**.
10. Click the **...** button.
11. Click **Add**, then navigate to the directory where the MySQL Connector/C++ header files are located. This will be `/usr/local/include` unless you have installed the files to a different location. Click **Select**. Click **OK**.

Figure 22.88. Setting the Header Include Directory



12. Click **OK** again to close the **Project Properties** dialog.

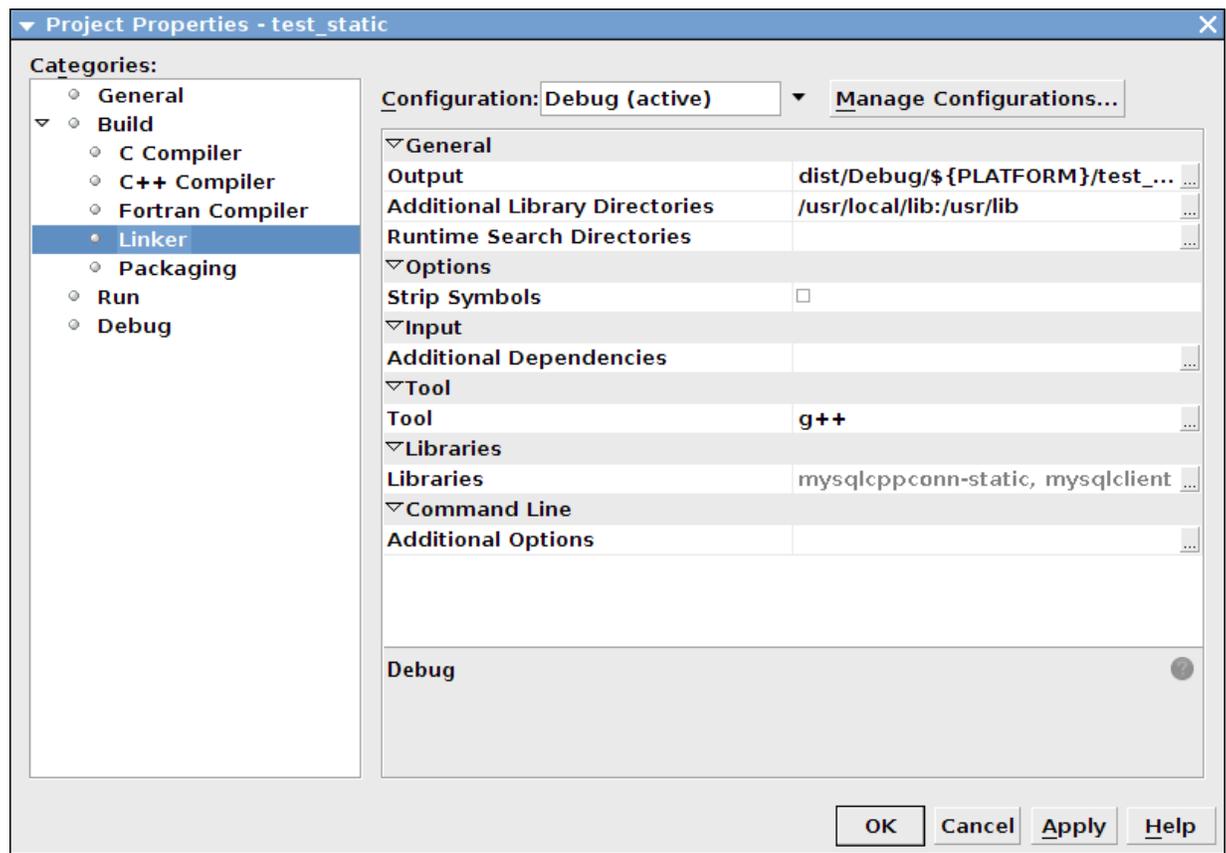
At this point, you have created a NetBeans project, containing a single C++ source file. You have also ensured that the necessary include files are accessible. Before continuing, decide whether your project is to use the MySQL Connector/C++ static or dynamic library. The project settings are slightly different in each case, because you link against a different library.

Using the static library

To use the static library, link against two library files, `libmysqlcppconn-static.a` and `libmysqlclient.a`. The locations of the files depend on your setup, but typically the former are in `/usr/local/lib` and the latter in `/usr/lib`. Note the file `libmysqlclient.a` is not part of MySQL Connector/C++, but is the MySQL Client Library file distributed with MySQL Server. Remember, the MySQL Client Library is an optional component as part of the MySQL Server installation process. Note the MySQL Client Library is also available as part of the MySQL Connector/C distribution.

1. The first step is to set the project to link the necessary library files. Select **File**, **Project Properties** from the main menu.
2. In the **Categories**: tree view, navigate to **Linker**.
3. In the **General** panel, select **Additional Library Directories**. Click the **...** button.
4. Select and add the `/usr/lib` and `/usr/local/lib` directories.
5. In the same panel, add the two library files required for static linking as discussed earlier. The properties panel should then look similar to the following screenshot:

Figure 22.89. Setting the Static Library Directories and File Names



6. Click OK to close the **Project Properties** dialog.

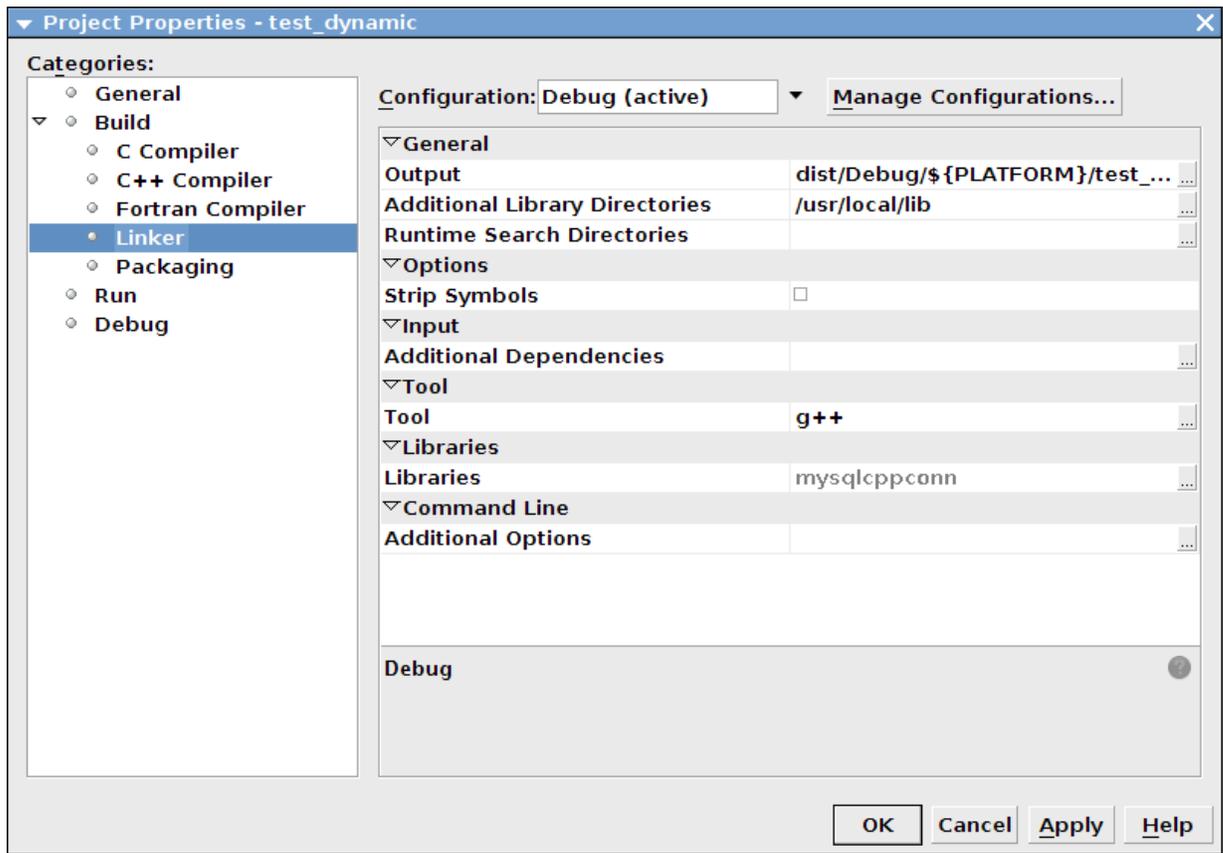
Using the dynamic library

For your application to use the MySQL Connector/C++ dynamic library, you link your project with a single library file, `libmysqlcppconn.so`. The location of this file depends on how you configured your installation of MySQL Connector/C++, but is typically `/usr/local/lib`.

1. The first step is to set the project to link the necessary library file. Select **File**, **Project Properties** from the main menu.
2. In the **Categories**: tree view, navigate to **Linker**.
3. In the **General** panel, select **Additional Library Directories**. Click the **...** button.

4. Select and add the `/usr/local/lib` directories.
5. In the same panel, add the library file required for static linking as discussed earlier. The properties panel should then look similar to the following screenshot:

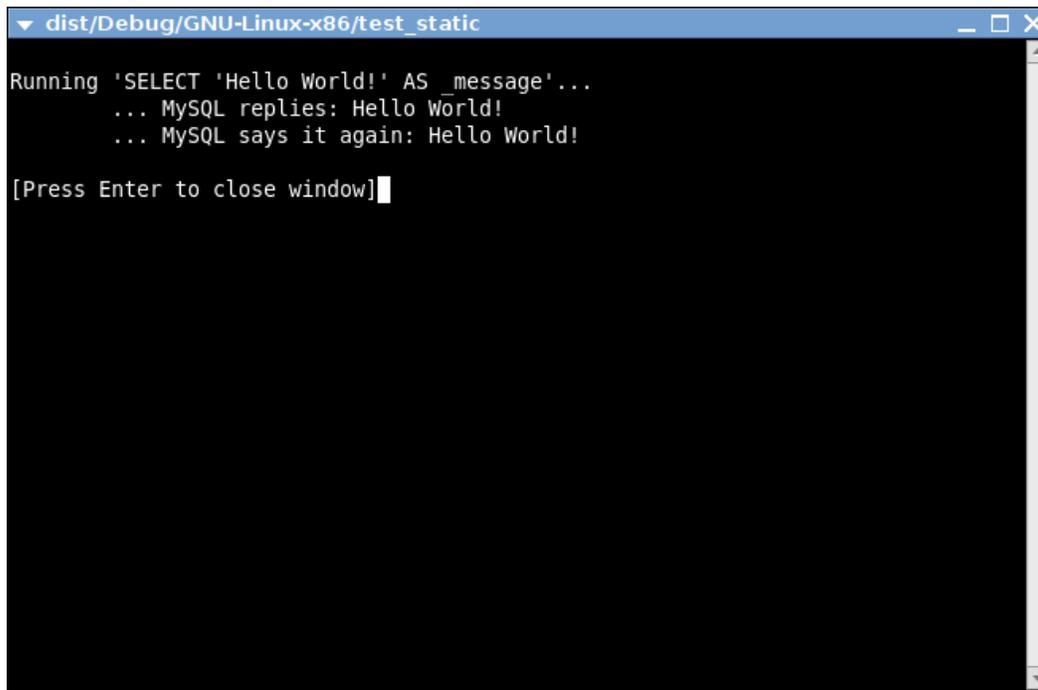
Figure 22.90. Setting the Dynamic Library Directory and File Name



6. Click OK to close the Project Properties dialog.

Having configured your project, you build it by selecting **Run, Build Main Project** from the main menu. You then run the project using **Run, Run Main Project**.

On running the application, you should see a screen similar to the following (this is actually the static version of the application shown):

Figure 22.91. The Example Application RunningA screenshot of a terminal window titled "dist/Debug/GNU-Linux-x86/test_static". The window contains the following text:

```
Running 'SELECT 'Hello World!' AS _message'...  
... MySQL replies: Hello World!  
... MySQL says it again: Hello World!  
[Press Enter to close window]
```

**Note**

Note the above settings and procedures were carried out for the default [Debug](#) configuration. To create a [Release](#) configuration, select that configuration before setting the Project Properties.

22.4.5. MySQL Connector/C++ Getting Started: Usage Examples

The download package contains usage examples in the directory [examples/](#). The examples explain the basic usage of the following classes:

- [Connection](#)
- [Driver](#)
- [PreparedStatement](#)
- [ResultSet](#)
- [ResultSetMetaData](#)
- [Statement](#)

The examples cover:

- Using the [Driver](#) class to connect to MySQL
- Creating tables, inserting rows, fetching rows using (simple) statements
- Creating tables, inserting rows, fetching rows using prepared statements
- Hints for working around prepared statement limitations
- Accessing result set metadata

The examples in this document are only code snippets. The code snippets provide a brief overview on the API. They are not complete programs. Please check the [examples/](#) directory of your MySQL Connector/C++ installation for complete programs. Please also read the [README](#) file in the [examples/](#) directory. To test the example code, first edit the [examples.h](#) file in the [examples/](#) directory, to add your connection information. Then rebuild the code by issuing a `make` command.

The examples in the [examples/](#) directory include:

- [examples/connect.cpp](#):
How to create a connection, insert data into MySQL and handle exceptions.
- [examples/connection_meta_schemaobj.cpp](#):
How to obtain metadata associated with a connection object, for example, a list of tables, databases, MySQL version, connector version.
- [examples/debug_output.cpp](#):
How to activate and deactivate the MySQL Connector/C++ debug protocol.
- [examples/exceptions.cpp](#):
A closer look at the exceptions thrown by the connector and how to fetch error information.
- [examples/prepared_statements.cpp](#):
How to run Prepared Statements including an example how to handle SQL commands that cannot be prepared by the MySQL Server.
- [examples/resultset.cpp](#):
How to fetch data and iterate over the result set (cursor).
- [examples/resultset_meta.cpp](#):
How to obtain metadata associated with a result set, for example, number of columns and column types.
- [examples/resultset_types.cpp](#):
Result sets returned from metadata methods - this is more a test than much of an example.
- [examples/standalone_example.cpp](#):
Simple standalone program not integrated into regular `CMake` builds.
- [examples/statements.cpp](#):
How to run SQL commands without using Prepared Statements.
- [examples/cpp_trace_analyzer.cpp](#):
This example shows how to filter the output of the [debug trace](#). Please see the inline comments for further documentation. This script is unsupported.

22.4.5.1. MySQL Connector/C++ Connecting to MySQL

A connection to MySQL is established by retrieving an instance of `sql::Connection` from a `sql::mysql::MySQL_Driver` object. A `sql::mysql::MySQL_Driver` object is returned by `sql::mysql::MySQL_Driver::get_mysql_driver_instance()`.

```

sql::mysql::MySQL_Driver *driver;
sql::Connection *con;

driver = sql::mysql::MySQL_Driver::get_mysql_driver_instance();
con = driver->connect("tcp://127.0.0.1:3306", "user", "password");

delete con;

```

Make sure that you free the `sql::Connection` object as soon as you do not need it any more. But do not explicitly free the connector object!

22.4.5.2. MySQL Connector/C++ Running a simple query

For running simple queries, you can use the methods `sql::Statement::execute()`, `sql::Statement::executeQuery()` and `sql::Statement::executeUpdate()`. Use the method `sql::Statement::execute()` if your query does not return a result set or if your query returns more than one result set. See the `examples/` directory for more on this.

```

sql::mysql::MySQL_Driver *driver;
sql::Connection *con;
sql::Statement *stmt;

driver = sql::mysql::get_mysql_driver_instance();
con = driver->connect("tcp://127.0.0.1:3306", "user", "password");

stmt = con->createStatement();
stmt->execute("USE " EXAMPLE_DB);
stmt->execute("DROP TABLE IF EXISTS test");
stmt->execute("CREATE TABLE test(id INT, label CHAR(1))");
stmt->execute("INSERT INTO test(id, label) VALUES (1, 'a')");

delete stmt;
delete con;

```

Note that you have to free `sql::Statement` and `sql::Connection` objects explicitly using `delete`.

22.4.5.3. MySQL Connector/C++ Fetching results

The API for fetching result sets is identical for (simple) statements and prepared statements. If your query returns one result set, use `sql::Statement::executeQuery()` or `sql::PreparedStatement::executeQuery()` to run your query. Both methods return `sql::ResultSet` objects. The previous version does buffer all result sets on the client to support cursors.

```

// ...
sql::Connection *con;
sql::Statement *stmt;
sql::ResultSet *res;
// ...
stmt = con->createStatement();
// ...

res = stmt->executeQuery("SELECT id, label FROM test ORDER BY id ASC");
while (res->next()) {
    // You can use either numeric offsets...
    cout << "id = " << res->getInt(1); // getInt(1) returns the first column
    // ... or column names for accessing results.
    // The latter is recommended.
    cout << ", label = " << res->getString("label") << " " << endl;
}

delete res;
delete stmt;
delete con;

```

**Note**

Note in the preceding code snippet that column indexing starts from 1.

Note that you have to free `sql::Statement`, `sql::Connection` and `sql::ResultSet` objects explicitly using delete.

The usage of cursors is demonstrated in the examples contained in the download package.

22.4.5.4. MySQL Connector/C++ Using Prepared Statements

If you are not familiar with Prepared Statements on MySQL have an extra look at the source code comments and explanations in the file `examples/prepared_statement.cpp`.

`sql::PreparedStatement` is created by passing an SQL query to `sql::Connection::prepareStatement()`. As `sql::PreparedStatement` is derived from `sql::Statement`, you will feel familiar with the API once you have learned how to use (simple) statements (`sql::Statement`). For example, the syntax for fetching results is identical.

```
// ...
sql::Connection *con;
sql::PreparedStatement *prep_stmt
// ...

prep_stmt = con->prepareStatement("INSERT INTO test(id, label) VALUES (?, ?)");

prep_stmt->setInt(1, 1);
prep_stmt->setString(2, "a");
prep_stmt->execute();

prep_stmt->setInt(1, 2);
prep_stmt->setString(2, "b");
prep_stmt->execute();

delete prep_stmt;
delete con;
```

As usual, you have to free `sql::PreparedStatement` and `sql::Connection` objects explicitly.

22.4.5.5. MySQL Connector/C++ Complete Example 1

The following code shows a complete example of how to use MySQL Connector/C++:

```
/* Copyright 2008, 2010, Oracle and/or its affiliates. All rights reserved.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; version 2 of the License.

There are special exceptions to the terms and conditions of the GPL
as it is applied to this software. View the full text of the
exception in file EXCEPTIONS-CONNECTOR-C++ in the directory of this
software distribution.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
*/
```

```
/* Standard C++ includes */
#include <stdlib.h>
#include <iostream>

/*
   Include directly the different
   headers from cppconn/ and mysql_driver.h + mysql_util.h
   (and mysql_connection.h). This will reduce your build time!
*/
#include "mysql_connection.h"

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>

using namespace std;

int main(void)
{
    cout << endl;
    cout << "Running 'SELECT 'Hello World!' AS _message' >
        AS _message'..." << endl;

    try {
        sql::Driver *driver;
        sql::Connection *con;
        sql::Statement *stmt;
        sql::ResultSet *res;

        /* Create a connection */
        driver = get_driver_instance();
        con = driver->connect("tcp://127.0.0.1:3306", "root", "root");
        /* Connect to the MySQL test database */
        con->setSchema("test");

        stmt = con->createStatement();
        res = stmt->executeQuery("SELECT 'Hello World!' AS _message");
        while (res->next()) {
            cout << "\t... MySQL replies: ";
            /* Access column data by alias or column name */
            cout << res->getString("_message") << endl;
            cout << "\t... MySQL says it again: ";
            /* Access column data by numeric offset, 1 is the first column */
            cout << res->getString(1) << endl;
        }
        delete res;
        delete stmt;
        delete con;

    } catch (sql::SQLException &e) {
        cout << "# ERR: SQLException in " << __FILE__;
        cout << "(" << __FUNCTION__ << ") on line " >
            << __LINE__ << endl;
        cout << "# ERR: " << e.what();
        cout << " (MySQL error code: " << e.getErrorCode();
        cout << ", SQLState: " << e.getSQLState() << ")" << endl;
    }

    cout << endl;

    return EXIT_SUCCESS;
}
```

22.4.5.6. MySQL Connector/C++ Complete Example 2

The following code shows a complete example of how to use MySQL Connector/C++:

```
/* Copyright 2008, 2010, Oracle and/or its affiliates. All rights reserved.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; version 2 of the License.

There are special exceptions to the terms and conditions of the GPL
as it is applied to this software. View the full text of the
exception in file EXCEPTIONS-CONNECTOR-C++ in the directory of this
software distribution.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
*/

/* Standard C++ includes */
#include <stdlib.h>
#include <iostream>

/*
   Include directly the different
   headers from cppconn/ and mysql_driver.h + mysql_util.h
   (and mysql_connection.h). This will reduce your build time!
*/
#include "mysql_connection.h"

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>
#include <cppconn/prepared_statement.h>

using namespace std;

int main(void)
{
    cout << endl;
    cout << "Let's have MySQL count from 10 to 1..." << endl;

    try {
        sql::Driver *driver;
        sql::Connection *con;
        sql::Statement *stmt;
        sql::ResultSet *res;
        sql::PreparedStatement *pstmt;

        /* Create a connection */
        driver = get_driver_instance();
        con = driver->connect("tcp://127.0.0.1:3306", "root", "root");
        /* Connect to the MySQL test database */
        con->setSchema("test");

        stmt = con->createStatement();
        stmt->execute("DROP TABLE IF EXISTS test");
        stmt->execute("CREATE TABLE test(id INT)");
        delete stmt;
    }
}
```

```

/* '?' is the supported placeholder syntax */
pstmt = con->prepareStatement("INSERT INTO test(id) VALUES (?)");
for (int i = 1; i <= 10; i++) {
    pstmt->setInt(1, i);
    pstmt->executeUpdate();
}
delete pstmt;

/* Select in ascending order */
pstmt = con->prepareStatement("SELECT id FROM test ORDER BY id ASC");
res = pstmt->executeQuery();

/* Fetch in reverse = descending order! */
res->afterLast();
while (res->previous())
    cout << "\t... MySQL counts: " << res->getInt("id") << endl;
delete res;

delete pstmt;
delete con;

} catch (sql::SQLException &e) {
    cout << "# ERR: SQLException in " << __FILE__;
    cout << "(" << __FUNCTION__ << " on line " >
        << __LINE__ << endl;
    cout << "# ERR: " << e.what();
    cout << " (MySQL error code: " << e.getErrorCode();
    cout << ", SQLState: " << e.getSQLState() << >
        " )" << endl;
}

cout << endl;

return EXIT_SUCCESS;
}

```

22.4.6. MySQL Connector/C++ Tutorials

Here are some tutorials on using MySQL Connector/C++. Also consult the examples in [Section 22.4.5, “MySQL Connector/C++ Getting Started: Usage Examples”](#).

Setting up the World database for use in the tutorials

These tutorials refer to tables and sample data from the [World](#) database, which you can download from the [MySQL Documentation](#) page. Look for the section called “Example Databases”.

Tutorial framework code

These examples all use the following framework, to avoid repeating code:

```

#include <stdlib.h>
#include <iostream>
#include <sstream>
#include <stdexcept>

#include "mysql_connection.h"

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>
#include <cppconn/prepared_statement.h>

#define EXAMPLE_HOST "localhost"

```

```

#define EXAMPLE_USER "root"
#define EXAMPLE_PASS ""
#define EXAMPLE_DB "world"

using namespace std;

int main(int argc, const char **argv)
{
    string url(argc >= 2 ? argv[1] : EXAMPLE_HOST);
    const string user(argc >= 3 ? argv[2] : EXAMPLE_USER);
    const string pass(argc >= 4 ? argv[3] : EXAMPLE_PASS);
    const string database(argc >= 5 ? argv[4] : EXAMPLE_DB);

    cout << "Connector/C++ tutorial framework..." << endl;
    cout << endl;

    try {

/* INSERT TUTORIAL CODE HERE! */

    } catch (sql::SQLException &e) {
        /*
         * The MySQL Connector/C++ throws three different exceptions:
         *
         * - sql::MethodNotImplementedException (derived from sql::SQLException)
         * - sql::InvalidArgumentException (derived from sql::SQLException)
         * - sql::SQLException (derived from std::runtime_error)
         */
        cout << "# ERR: SQLException in " << __FILE__ << endl;
        cout << "(" << __FUNCTION__ << ") on line " << __LINE__ << endl;
        /* Use what() (derived from std::runtime_error) to fetch the error message */
        cout << "# ERR: " << e.what();
        cout << " (MySQL error code: " << e.getErrorCode();
        cout << ", SQLState: " << e.getSQLState() << " )" << endl;

        return EXIT_FAILURE;
    }

    cout << "Done." << endl;
    return EXIT_SUCCESS;
}

```

To compile and run the framework

First, copy and paste the framework code to a file such as `frmwk.cpp`. Edit the `#define` statements to reflect your connection details (server, user, password, database).

To compile the framework, for example on Mac OS X, type:

```
shell> g++ -o frmwk -I/usr/local/include -I/usr/local/include/cppconn -lmysqlcppconn frmwk.cpp
```

To run the framework, enter the following:

```
shell> ./frmwk
```

You will see a simple message. You are now ready to continue to the tutorials.

22.4.6.1. Tutorial: Calling Stored Procedures with Statements in MySQL Connector/C++

Stored procedures can be called using both Statements and Prepared Statements. This tutorial looks at calling stored procedures using Statements. The following tutorial [Section 22.4.6.2, "Tutorial: Calling Stored Procedures with Prepared Statements in MySQL Connector/C++"](#) will cover the use of Prepared Statements.

You can construct and call various combinations of stored procedures:

1. A stored procedure that does not return any result. For example, such a stored procedure can log non-critical information, or change database data in a straightforward way.
2. A stored procedure that returns an output parameter. For example, such a procedure can indicate success or failure, query a single item of data, or combine multiple data items into a single result.
3. A stored procedure that returns a result set. The procedure can execute a query that matches an arbitrary number of table rows. Your application loops through the result set to display, transform, or otherwise process each item.

The following stored procedures illustrate each of these scenarios.

The following routine adds a country into the World database, but does not return a result. This corresponds to Scenario 1 above.

```
CREATE PROCEDURE add_country (IN country_code CHAR(3), IN country_name CHAR(52), IN continent_name CHAR(30))
BEGIN
    INSERT INTO Country(Code, Name, Continent) VALUES (country_code, country_name, continent_name);
END
```

The next routine returns the population of a specified country, and corresponds to Scenario 2 above:

```
CREATE PROCEDURE get_pop (IN country_name CHAR(52), OUT country_pop INT(11))
BEGIN
    SELECT Population INTO country_pop FROM Country WHERE Name = country_name;
END
```

The next routine is an example of a procedure returning a result set containing multiple records. This routine corresponds to Scenario 3 above.

```
CREATE PROCEDURE get_data ()
BEGIN
    SELECT Code, Name, Population, Continent FROM Country WHERE Continent = "Oceania" AND Population < 1000000;
    SELECT Code, Name, Population, Continent FROM Country WHERE Continent = "Europe" AND Population < 1000000;
    SELECT Code, Name, Population, Continent FROM Country WHERE Continent = "North America" AND Population < 1000000;
END
```

Enter and test the stored procedures to ensure no errors have been introduced. You are now ready to start writing applications using Connector/C++ that call stored procedures.

Scenario 1 - Stored procedure does not return a result set

The first case illustrates Scenario 1, calling a Stored procedure that does not return a result set.

1. Make a copy of the tutorial framework code.
2. Insert the following code into the framework at the correct location (denoted by an INSERT HERE comment in the framework).

```
sql::Driver* driver = get_driver_instance();
std::auto_ptr<sql::Connection> con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr<sql::Statement> stmt(con->createStatement());

// We don't need to check the return value explicitly, if it indicates
// an error Connector/C++ will generate an exception.
stmt->execute("CALL add_country(\"ATL\", \"Atlantis\", \"North America\")");
```

3. Compile the program using the following command:

```
shell> g++ -o sp_scenario1 -I/usr/local/include/cppconn/ -lmysqlcppconn sp_scenario1.cpp
```

4. Run the program by typing:

```
shell> ./sp_scenario1
```

5. Using the `mysql` Command Line Client, or other suitable tool, check the World database to determine that it has been updated correctly. You can use a query such as:

```
SELECT Code, Name, Continent FROM Country WHERE Code="ATL";
```

The code in this case simply creates a statement and then invokes the execute method on it, passing the call to the stored procedure as a parameter. The stored procedure itself does not return a value, although it is important to note there will always be a return value from the call - this is simply the call status. MySQL Connector/C++ handles this status for you, so you do not need code to handle it explicitly. If the call fails for some reason, an exception will be raised, and this will be handled by the `catch` statement in the code.

Scenario 2 - Stored procedure returns an output parameter

You will now see how to handle a stored procedure that returns an output parameter.

1. Enter the following code into the tutorial framework code:

```
sql::Driver* driver = get_driver_instance();
std::auto_ptr<sql::Connection> con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr<sql::Statement> stmt(con->createStatement());

stmt->execute("CALL get_pop(\"Uganda\", @pop)");

std::auto_ptr<sql::ResultSet> res(stmt->executeQuery("SELECT @pop AS _reply"));
while (res->next())
    cout << "Population of Uganda: " << res->getString("_reply") << endl;

stmt->execute("CALL get_pop_continent(\"Asia\", @pop)");

res.reset(stmt->executeQuery("SELECT @pop AS _reply"));
while (res->next())
    cout << "Population of Asia: " << res->getString("_reply") << endl;

stmt->execute("CALL get_world_pop(@pop)");

res.reset(stmt->executeQuery("SELECT @pop AS _reply"));
while (res->next())
    cout << "Population of World: " << res->getString("_reply") << endl;
```

2. Compile the program using the following command:

```
shell> g++ -o sp_scenario2 -I/usr/local/include/cppconn/ -lmysqlcppconn sp_scenario2.cpp
```

3. Run the program by typing:

```
shell> ./sp_scenario2
```

Note the output generated by the program.

In this scenario, the stored procedure sets an output parameter. This is not returned as such, but needs to be obtained using a query. If running the SQL statements directly, this might be similar to the following:

```
CALL get_world_pop(@pop);
SELECT @pop;
```

In the C++ code, a similar sequence is carried out. First, the `CALL` is executed as seen earlier. To obtain the output parameter, an additional query must be executed. This query results in a `ResultSet` that can then be processed in a `while` loop. The simplest way to retrieve the data in this case is to use a `getString` method on the `ResultSet`, passing the name of the variable to access. In this example `_reply` is used as a placeholder for the variable and therefore is used as the key to access the correct element of the result dictionary.

Scenario 3 - Stored procedure returns a Result Set

You will now see how to handle a stored procedure that returns a result set.

1. Enter the following code into the tutorial framework code:

```
sql::Driver* driver = get_driver_instance();
std::auto_ptr<sql::Connection> con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr<sql::Statement> stmt(con->createStatement());

stmt->execute("CALL get_stats()");
std::auto_ptr< sql::ResultSet > res;
do {
    res.reset(stmt->getResultSet());
    while (res->next()) {
        cout << "Result: " << res->getString(1) << endl;
    }
} while (stmt->getMoreResults());
```

2. Compile the program using the following command:

```
shell> g++ -o sp_scenario3 -I/usr/local/include/cppconn/ -lmysqlcppconn sp_scenario3.cpp
```

3. Run the program by typing:

```
shell> ./sp_scenario3
```

Note the output generated by the program.

The code is similar to the examples you have previously seen. The code of particular interest in this case is:

```
do {
    res.reset(stmt->getResultSet());
    while (res->next()) {
        cout << "Name: " << res->getString("Name")
            << " Population: " << res->getInt("Population")
            << endl;
    }
} while (stmt->getMoreResults());
```

The `CALL` is executed as before, with the results being returned into multiple `ResultSets`. This is because the Stored Procedure in this case uses multiple `SELECT` statements. In this example, the output shows that three Result Sets are processed, because there are three `SELECT` statements in the Stored Procedure. All of the Result Sets have more than one row.

The results are processed using the pattern:

```
do {
    Get Result Set
    while (Get Result) {
        Process Result
```

```

}
} while (Get More Result Sets);

```



Note

This pattern would be used even if the Stored Procedure carried out a single `SELECT` and you knew there was only one result set. This is a requirement of the underlying protocol.

22.4.6.2. Tutorial: Calling Stored Procedures with Prepared Statements in MySQL Connector/C++

Before working through this tutorial, it is recommended you first work through the previous tutorial [Section 22.4.6.1, "Tutorial: Calling Stored Procedures with Statements in MySQL Connector/C++"](#).

Scenario 1 - Using a Prepared Statement to prepare a Stored Procedure that does not return a result set

1. Add the following code to the `try` block of the tutorial framework:

```

vector<string> code_vector;
code_vector.push_back("SLD");
code_vector.push_back("DSN");
code_vector.push_back("ATL");

vector<string> name_vector;
name_vector.push_back("Sealand");
name_vector.push_back("Disneyland");
name_vector.push_back("Atlantis");

vector<string> cont_vector;
cont_vector.push_back("Europe");
cont_vector.push_back("North America");
cont_vector.push_back("Oceania");

sql::Driver * driver = get_driver_instance();

std::auto_ptr< sql::Connection > con(driver->connect(url, user, pass));
con->setSchema(database);

std::auto_ptr< sql::PreparedStatement > pstmt;

pstmt.reset(con->prepareStatement("CALL add_country(?,?,?)"));
for (int i=0; i<3; i++)
{
    pstmt->setString(1,code_vector[i]);
    pstmt->setString(2,name_vector[i]);
    pstmt->setString(3,cont_vector[i]);

    pstmt->execute();
}

```

Also, add `#include <vector>` to the top of your code, because vectors are used to store sample data.

2. Compile the code using the following command:

```
g++ -o ps_scenario1 -I/usr/local/include/cppconn/ -lmysqlcppconn ps_scenario1.cpp
```

3. Run the code using the command:

```
./ps_scenario1
```

4. You can test the database has been updated correctly by using a query such as:

```
SELECT Code, Name, Continent FROM Country WHERE Code = "DSN" OR Code="ATL" OR Code="SLD";
```

The code is relatively simple, as no processing is required to handle Result Sets. The procedure call, `CALL add_country(?, ?, ?)`, is made using placeholders for input parameters denoted by '?'. These placeholders are replaced by values using the Prepared Statement's `setString` method in this case. The `for` loop is set up to iterate 3 times, as there are three data sets in this example. The same Prepared Statement is executed three times, each time with different input parameters.

Scenario 2 - Using a Prepared Statement to prepare a Stored Procedure that uses an output parameter

In this scenario a different Stored Procedure is going to be used compared to the one used in the tutorial [Section 22.4.6.1, "Tutorial: Calling Stored Procedures with Statements in MySQL Connector/C++"](#). This is to illustrate passing an input parameter as well as fetching an output parameter. The stored routine is as follows:

```
CREATE PROCEDURE get_pop_continent (IN continent_name CHAR(30), OUT continent_pop INT(11))
BEGIN
    SELECT SUM(Population) INTO continent_pop FROM Country WHERE Continent = continent_name;
END
```

1. Copy the following code into the try block of the tutorial framework code:

```
vector<string> cont_vector;
cont_vector.push_back("Europe");
cont_vector.push_back("North America");
cont_vector.push_back("Oceania");

sql::Driver * driver = get_driver_instance();

std::auto_ptr< sql::Connection > con(driver->connect(url, user, pass));
con->setSchema(database);

std::auto_ptr< sql::Statement > stmt(con->createStatement());
std::auto_ptr< sql::PreparedStatement > pstmt;
std::auto_ptr< sql::ResultSet > res;

pstmt.reset(con->prepareStatement("CALL get_pop_continent(?, @pop)"));

for (int i=0; i<3; i++)
{
    pstmt->setString(1, cont_vector[i]);
    pstmt->execute();
    res.reset(stmt->executeQuery("SELECT @pop AS _population"));
    while (res->next())
        cout << "Population of " << cont_vector[i] << " is " << res->getString("_population") << endl;
}
```

You will also need to add the line `#include <vector>` to the top of the code, as vectors are used in this example.

2. Compile the code using:

```
shell> g++ -o ps_scenario2 -I/usr/local/include/cppconn/ -lmysqlcppconn ps_scenario2.cpp
```

3. Run the code using:

```
shell> ./ps_scenario2
```

4. Make a note of the output.

In this scenario a Prepared Statement is created that calls the Stored Procedure `get_pop_continent`. This procedure takes an input parameter, and also returns an output parameter. The approach used is to create another statement that can be used to fetch the output parameter using a `SELECT` query. Note that when the Prepared Statement is created, the input parameter to the Stored Procedure is denoted by '?'. Prior to execution of Prepared Statement it is necessary to replace this placeholder by an actual value. This is done using methods such as `setString` and `setInt`, for example:

```
pstmt->setString(1,cont_vector[i]);
```

Although for the query used to obtain the output parameter a single result set is expected, it is important to use the `while` loop to catch more than one result, to avoid the possibility of the connection becoming unstable.

Scenario 3 - Using a Prepared Statement to prepare a Stored Procedure that returns multiple Result Sets



Note

Note this scenario is not supported on versions of MySQL prior to 5.5.3. This is due to a limitation in the client/server protocol.

1. Enter the following code into the `try` block of the tutorial framework:

```
sql::Driver * driver = get_driver_instance();

std::auto_ptr< sql::Connection > con(driver->connect(url, user, pass));
con->setSchema(database);

std::auto_ptr< sql::PreparedStatement > pstmt;
std::auto_ptr< sql::ResultSet > res;

pstmt.reset(con->prepareStatement("CALL get_data("));
res.reset(pstmt->executeQuery());

do {
    res.reset(pstmt->getResultSet());
    while (res->next()) {
        cout << "Name: " << res->getString("Name")
             << " Population: " << res->getInt("Population")
             << endl;
    }
} while (pstmt->getMoreResults());
```

2. Compile the code using the following command:

```
shell> g++ -o ps_scenario3 -I/usr/local/include/cppconn/ -lmysqlcppconn ps_scenario3.cpp
```

3. Run the program using the command:

```
shell> ./ps_scenario3
```

4. Make a note of the output generated.

The code executes the Stored Procedure using a Prepared Statement. The standard do-while construct is used to ensure that all Result Sets are fetched. In this case the returned values are fetched from the Result Sets using the `getInt` and `getString` methods.

22.4.7. MySQL Connector/C++ Debug Tracing

Although a debugger can be used to debug your application, you may find it beneficial to turn on the debug traces of the connector. Some problems happen randomly which makes them difficult to debug using a

debugger. In such cases debug traces and protocol files are more useful because they allow you to trace the activities of all instances of your program.

DTrace is a very powerful technology to trace any application without having to develop an extra trace module for your application. Unfortunately, DTrace is currently only available on Solaris, MacOS 10.5, and FreeBSD.

The MySQL Connector/C++ can write two trace files:

1. Trace file generated by the MySQL Client Library
2. Trace file generated internally by MySQL Connector/C++

The first trace file can be generated by the underlying MySQL Client Library (libmysql). To enable this trace, the connector will call the C-API function `mysql_debug()` internally. As only debug versions of the MySQL Client Library are capable of writing a trace file, compile MySQL Connector/C++ against a debug version of the library if you want to utilize this trace. The trace shows the internal function calls and the addresses of internal objects as shown below:

```
>mysql_stmt_init
| >_mymalloc
| | enter: Size: 816
| | exit: ptr: 0x68e7b8
| <_mymalloc | >init_alloc_root
| | enter: root: 0x68e7b8
| | >_mymalloc
| | | enter: Size: 2064
| | | exit: ptr: 0x68eb28
| [...]

```

The second trace is the MySQL Connector/C++ internal trace. It is available with debug and nondebug builds of the connector as long as you have enabled the tracing module at compile time using `cmake -DMYSQLCPPCONN_TRACE_ENABLE=BOOL=1`. By default, the tracing functionality is not available and calls to trace functions are removed by the preprocessor.

Compiling the connector with tracing functionality enabled will cause two additional tracing function calls per each connector function call. Run your own benchmark to find out how much this will impact the performance of your application.

A simple test using a loop running 30,000 INSERT SQL statements showed no significant real-time impact. The two variants of this application using a trace enabled and trace disabled version of the connector performed equally well. The run time measured in real-time was not significantly impacted as long as writing a debug trace was not enabled. However, there will be a difference in the time spent in the application. When writing a debug trace the IO subsystem may become a bottleneck.

In summary, use connector builds with tracing enabled carefully. Trace enabled versions may cause higher CPU usage even if the overall run time of your application is not impacted significantly.

```
| INF: Tracing enabled
| <MySQL_Connection::setClientOption
| >MySQL_Prepared_Statement::setInt
| INF: this=0x69a2e0
| >MySQL_Prepared_Statement::checkClosed
| <MySQL_Prepared_Statement::checkClosed
| <MySQL_Prepared_Statement::setInt
| [...]

```

The example from `examples/debug_output.cpp` demonstrates how to activate the debug traces in your program. Currently they can only be activated through API calls. The traces are controlled on a per-connection basis. You can use the `setClientOptions()` method of a connection object to activate and

deactivate the generation of a trace. The MySQL Client Library trace is always written into a file, whereas the connector's protocol messages are printed to standard out.

```
sql::Driver *driver;
int on_off = 1;

/* Using the Driver to create a connection */
driver = get_driver_instance();
std::auto_ptr< sql::Connection > con(driver->connect(host, user, pass));

/*
Activate debug trace of the MySQL Client Library (C-API)
Only available with a debug build of the MySQL Client Library!
*/
con->setClientOption("libmysql_debug", "d:t:0,client.trace");

/*
Tracing is available if you have compiled the driver using
cmake -DMYSQLCPPCONN_TRACE_ENABLE=BOOL=1
*/
con->setClientOption("client_trace", &on_off);
```

22.4.8. MySQL Connector/C++ Usage Notes

See the [JDBC overview](#) for information on JDBC 4.0. Please also check the [examples/](#) directory of the download package.

- `DatabaseMetaData::supportsBatchUpdates()` returns `true` because MySQL supports batch updates in general. However, no API calls for batch updates are provided by the MySQL Connector/C++ API.
- Two non-JDBC methods let you fetch and set unsigned integers: `getUInt64()` and `getUInt()`. These are available for `ResultSet` and `PreparedStatement`:
 - `ResultSet::getUInt64()`
 - `ResultSet::getUInt()`
 - `PreparedStatement::setUInt64()`
 - `PreparedStatement::setUInt()`

The corresponding `getLong()` and `setLong()` methods have been removed.

- The method `DatabaseMetaData::getColumns()` has 23 columns in its result set, rather than the 22 columns defined by JDBC. The first 22 columns are as described in the JDBC documentation, but column 23 is new:

23. `IS_AUTOINCREMENT`: String which is "YES" if the column is an auto-increment column. Otherwise the string contains "NO".

- MySQL Connector/C++ may return different metadata for the same column.

When you have any column that accepts a charset and a collation in its specification and you specify a binary collation, such as:

```
CHAR(250) CHARACTER SET 'latin1' COLLATE 'latin1_bin'
```

The server sets the `BINARY` flag in the result set metadata of this column. The method `ResultSetMetadata::getColumnTypeName()` uses the metadata and will report, due to the `BINARY` flag, that the column type name is `BINARY`. This is illustrated below:

```
mysql> create table varbin(a varchar(20) character set utf8 collate utf8_bin);
Query OK, 0 rows affected (0.00 sec)

mysql> select * from varbin;
Field 1: `a`
Catalog: `def`
Database: `test`
Table: `varbin`
Org_table: `varbin`
Type: VAR_STRING
Collation: latin1_swedish_ci (8)
Length: 20
Max_length: 0
Decimals: 0
Flags: BINARY

0 rows in set (0.00 sec)

mysql> select * from information_schema.columns where table_name='varbin'\G
***** 1. row *****
      TABLE_CATALOG: NULL
      TABLE_SCHEMA: test
      TABLE_NAME: varbin
      COLUMN_NAME: a
      ORDINAL_POSITION: 1
      COLUMN_DEFAULT: NULL
      IS_NULLABLE: YES
      DATA_TYPE: varchar
CHARACTER_MAXIMUM_LENGTH: 20
CHARACTER_OCTET_LENGTH: 60
      NUMERIC_PRECISION: NULL
      NUMERIC_SCALE: NULL
CHARACTER_SET_NAME: utf8
      COLLATION_NAME: utf8_bin
      COLUMN_TYPE: varchar(20)
      COLUMN_KEY:
      EXTRA:
      PRIVILEGES: select,insert,update,references
      COLUMN_COMMENT:
1 row in set (0.01 sec)
```

However, `INFORMATION_SCHEMA` gives no hint in its `COLUMNS` table that metadata will contain the `BINARY` flag. `DatabaseMetaData::getColumns()` uses `INFORMATION_SCHEMA`. It will report the type name `CHAR` for the same column. Note, a different type code is also returned.

- The MySQL Connector/C++ class `sql::DataType` defines the following JDBC standard data types: `UNKNOWN`, `BIT`, `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INTEGER`, `BIGINT`, `REAL`, `DOUBLE`, `DECIMAL`, `NUMERIC`, `CHAR`, `BINARY`, `VARCHAR`, `VARBINARY`, `LONGVARCHAR`, `LONGVARBINARY`, `TIMESTAMP`, `DATE`, `TIME`, `GEOMETRY`, `ENUM`, `SET`, `SQLNULL`.

However, the following JDBC standard data types are *not* supported by MySQL Connector/C++: `ARRAY`, `BLOB`, `CLOB`, `DISTINCT`, `FLOAT`, `OTHER`, `REF`, `STRUCT`.

- When inserting or updating `BLOB` or `TEXT` columns, MySQL Connector/C++ developers are advised not to use `setString()`. Instead, use the dedicated API function `setBlob()`.

The use of `setString()` can cause a `Packet too large` error message. The error will occur if the length of the string passed to the connector using `setString()` exceeds `max_allowed_packet` (minus a few bytes reserved in the protocol for control purposes). This situation is not handled in MySQL Connector/C++, as this could lead to security issues, such as extremely large memory allocation requests due to malevolently long strings.

However, if `setBlob()` is used, this problem does not arise. This is because `setBlob()` takes a streaming approach based on `std::istream`. When sending the data from the stream to MySQL Server, MySQL Connector/C++ will split the stream into chunks appropriate for MySQL Server and observe the `max_allowed_packet` setting currently being used.



Caution

When using `setString()` it is not possible to set `max_allowed_packet` [519] to a value large enough for the string, prior to passing it to MySQL Connector/C++. That configuration option cannot be changed within a session.

This difference with the JDBC specification ensures that MySQL Connector/C++ is not vulnerable to memory flooding attacks.

- In general, MySQL Connector/C++ works with MySQL 5.0, but it is not completely supported. Some methods may not be available when connecting to MySQL 5.0. This is because the Information Schema is used to obtain the requested information. There are no plans to improve the support for 5.0 because the current GA version of MySQL Server is 5.5. As a new product, MySQL Connector/C++ is primarily targeted at the MySQL Server GA version that was available on its release.

The following methods will throw a `sql::MethodNotImplemented` exception when you connect to MySQL earlier than 5.1.0:

- `DatabaseMetadata::getCrossReference()`
- `DatabaseMetadata::getExportedKeys()`
- MySQL Connector/C++ includes a method `Connection::getClientOption()` which is not included in the JDBC API specification. The prototype is:

```
void getClientOption(const std::string & optionName, void * optionValue)
```

The method can be used to check the value of connection properties set when establishing a database connection. The values are returned through the `optionValue` argument passed to the method with the type `void *`.

Currently, `getClientOption()` supports fetching the `optionValue` of the following options:

- `metadataUseInfoSchema`
- `defaultStatementResultType`
- `defaultPreparedStatementResultType`

The connection option `metadataUseInfoSchema` controls whether to use the `Information_Schemata` for returning the meta data of `SHOW` commands. In the case of `metadataUseInfoSchema`, interpret the `optionValue` argument as a boolean upon return.

In the case of both `defaultStatementResultType` and `defaultPreparedStatementResultType`, interpret the `optionValue` argument as an integer upon return.

The connection property can be either set when establishing the connection through the connection property map or using `void Connection::setClientOption(const std::string & optionName, const void * optionValue)` where `optionName` is assigned the value `metadataUseInfoSchema`.

Some examples are given below:

```
int defaultStmtResType;
int defaultPStmtResType;
conn->getClientOption("defaultStatementResultType", (void *) &defaultStmtResType);
conn->getClientOption("defaultPreparedStatementResultType", (void *) &defaultPStmtResType);

bool isInfoSchemaUsed;
conn->getClientOption("metadataUseInfoSchema", (void *) &isInfoSchemaUsed);
```

- MySQL Connector/C++ also supports the following methods not found in the JDBC API standard:

```
std::string MySQL_Connection::getSessionVariable(const std::string & varname)
```

```
void MySQL_Connection::setSessionVariable(const std::string & varname, const std::string & value)
```

Note that both methods are members of the `MySQL_Connection` class. The methods get and set MySQL session variables.

`setSessionVariable()` is equivalent to executing:

```
SET SESSION <varname> = <value>
```

`getSessionVariable()` is equivalent to executing the following and fetching the first return value:

```
SHOW SESSION VARIABLES LIKE "<varname>"
```

You can use “%” and other placeholders in <varname>, if the underlying MySQL server supports this.

- Fetching the value of a column can sometimes return different values depending on whether the call is made from a Statement or Prepared Statement. This is because the protocol used to communicate with the server differs depending on whether a Statement or Prepared Statement is used.

To illustrate this, consider the case where a column has been defined as of type `BIGINT`. The most negative `BIGINT` value is then inserted into the column. If a Statement and Prepared Statement are created that perform a `GetUInt64()` call, then the results will be different in each case. The Statement returns the maximum positive value for `BIGINT`. The Prepared Statement returns 0.

The reason for the different results is due to the fact that Statements use a text protocol, and Prepared Statements use a binary protocol. With the binary protocol in this case, a binary value is returned from the server that can be interpreted as an `int64`. In the above scenario a very large negative value was fetched with `GetUInt64()`, which fetches unsigned integers. As the large negative value cannot be sensibly converted to an unsigned value 0 is returned.

In the case of the Statement, which uses the text protocol, values are returned from the server as strings, and then converted as required. When a string value is returned from the server in the above scenario, the large negative value must be converted by the runtime library function `strtoul()`, which `GetUInt64()` calls. The behavior of `strtoul()` is dependent upon the specific runtime and host operating system, so the results can be variable. In the case given a large positive value was actually returned.

Although it is very rare, there are some cases where Statements and Prepared Statements can return different values unexpectedly, but this usually only happens in extreme cases such as the one mentioned.

- The JDBC documentation [lists many fields](#) for the `DatabaseMetaData` class. JDBC also appears to [define certain values](#) for those fields. However, MySQL Connector/C++ does not define certain values

for those fields. Internally enumerations are used and the compiler determines the values to assign to a field.

To compare a value with the field, use code such as the following, rather than making assumptions about specific values for the attribute:

```
// dbmeta is an instance of DatabaseMetaData
if (myvalue == dbmeta->attributeNoNulls) {
    ...
}
```

Usually *myvalue* will be a column from a result set holding metadata information. MySQL Connector/C++ does not guarantee that *attributeNoNulls* is 0. It can be any value.

- When programming Stored Procedures, JDBC has available an extra class, an extra abstraction layer for callable statements, the `CallableStatement` class. As this class is not present in MySQL Connector/C++, use the methods from the `Statement` and `PreparedStatement` classes to run a Stored Procedure using `CALL`.

22.4.9. MySQL Connector/C++ Known Bugs and Issues



Note

Please report bugs through [MySQL Bug System](#).

Known bugs:

None.

Known issues:

- When linking against a static library for 1.0.3 on Windows, define `CPPDBC_PUBLIC_FUNC` either in the compiler options (preferable) or with `/D "CPPCONN_PUBLIC_FUNC="`. You can also explicitly define it in your code by placing `#define CPPCONN_PUBLIC_FUNC` before the header inclusions.
- Generally speaking, C++ library binaries are less portable than C library binaries. Issues can be caused by name mangling, different Standard Template Library (STL) versions and using different compilers and linkers for linking against the libraries than were used for building the library itself.

Even a small change in the compiler version can, but does not have to, cause problems. If you obtain error messages, that you suspect are related to binary incompatibilities, build MySQL Connector/C++ from source, using the same compiler and linker that you will use to build and link your application.

Due to the variations between Linux distributions, compiler and linker versions and STL versions, it is not possible to provide binaries for each and every possible configuration. However, the MySQL Connector/C++ binary distributions contain a [README](#) file that describes the environment and settings used to build the binary versions of the libraries.

- To avoid potential crashes the build configuration of MySQL Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of MySQL Connector/C++ with a debug build of the client application.

For release notes detailing the changes in each release of Connector/C++, see [MySQL Connector/C++ Release Notes](#).

22.4.10. MySQL Connector/C++ Feature requests

You can suggest new features in the first instance by joining the mailing list or forum and talking with the developers directly. See [Section 22.4.11, “MySQL Connector/C++ Support”](#)

The following feature requests are currently being worked on:

- C++ references for `Statements`, `ResultSets`, and exceptions, are being considered, instead of pointers to heap memory. This reduces the exception handling burden for the programmer.
- Adopt STL (suggestions are welcome).
- JDBC compliance: data type interfaces and support through `ResultSet::getType()` and `PreparedStatement::bind()`. Introduce `sql::Blob`, `sql::Clob`, `sql::Date`, `sql::Time`, `sql::Timestamp`, `sql::URL`. Support `get|setBlob()`, `get|setClob()`, `get|setDate()`, `get|setTime()`, `get|setTimestamp()`, `get|setURL()`
- Add support for all C-API connection options. Improved support for `mysql_options`.
- Add connect method which supports passing options using HashMaps.
- Create Windows installer.

22.4.11. MySQL Connector/C++ Support

For general discussion of the MySQL Connector/C++ please use the [C/C++ community forum](#) or join the [MySQL Connector/C++ mailing list](#).

Bugs can be reported at the [MySQL bug Web site](#).

For release notes detailing the changes in each release of MySQL Connector/C++, see [MySQL Connector/C++ Release Notes](#).

For Licensing questions, and to purchase MySQL Products and Services, please see <http://www.mysql.com/buy-mysql/>

22.5. MySQL Connector/C

What is MySQL Connector/C?

MySQL Connector/C is a C client library for client/server communication. It is a standalone replacement for the MySQL Client Library shipped with the MySQL Server.

Why have a replacement for MySQL Client Library?

There is no need to compile or install the MySQL Server package if you need only the client library.

MySQL Connector/C does not rely on the MySQL Server release cycle, so bug fixes and new features can be distributed independently of MySQL Server releases.

MySQL Connector/C API documentation is available here [Section 22.8.3, “C API Function Descriptions”](#).

For release notes detailing the changes in each release of MySQL Connector/C, see [MySQL Connector/C Release Notes](#).

Supported platforms include:

- Windows
- Windows x64
- Linux
- Solaris
- FreeBSD
- Mac OS X
- HP-UX
- IBM AIX

22.5.1. Building MySQL Connector/C from the Source Code

Obtaining the Source Code

You can [download a tar file containing the source code](#) from the MySQL Developers site. Select the source code package from the drop down list.

The source code for development releases of the connector can be found at <http://launchpad.net/libmysql> on the Launchpad site.

The [source code for the 1.0 branch](#) is also available on the Launchpad site.

To get the code from the Launchpad site, you must have Bazaar installed. Use the command `bzr branch lp:libmysql`.

Building on Unix

Examples of supported Unix or Unix-like operating systems include:

- Solaris
- Linux
- HP-UX
- AIX
- OS X

Compiler Tools

Ideally, the native compiler tool set for the target platform is used for compilation. For example, this would be SunStudio for Solaris or aCC for HP-UX. However, the GNU tool-chain can be used across all platforms.

You also need [CMake](#) 2.6 or newer, which is available [online](#).

To Build

Change location to the top-level MySQL Connector/C source directory and use the following procedure:

1. To generate the `Makefile`, enter this command:

```
shell> cmake -G "Unix Makefiles"
```

Or, for a Debug build:

```
shell> cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug
```

2. Build the project:

```
shell> make
```

To Install

By default, `make install` installs the MySQL Connector/C files in the `/usr/local` directory. To change this location, specify another directory when generating the `Makefile`:

```
shell> cmake -G "Unix Makefiles" -DCMAKE_INSTALL_PREFIX=/mypath
```

Now, as `root`, enter the following command to install the MySQL Connector/C libraries and tools:

```
root-shell> make install
```

At this point, all of the MySQL Connector/C files will be in place.

Building on Microsoft Windows

Older versions of Microsoft Windows are not supported. Supported versions are Windows 2000, Windows XP, Windows Vista, Windows Server 2003, or Windows Server 2008.

Compiler Tools

Microsoft Visual Studio 8 or 9 is recommended. The Express Edition of Visual Studio and other compilers might work, but are untested.

You also need `CMake` 2.6 or newer, which is available [online](#).

To Build

1. Set the environment variables for the Visual Studio toolchain. Visual Studio includes a batch file to set these for you, and installs a shortcut in the **Start** menu to open a command prompt with these variables set.
2. Change location to the top-level MySQL Connector/C source directory.

To build MySQL Connector/C using the `CMake` command-line tool, enter the following command a command prompt window:

```
shell> cmake -G "Visual Studio 9 2008"
```

The result is a project file, `libmysql.sln`, that you can either open with Visual Studio, or build from the command line with either of these commands:

```
shell> devenv.com libmysql.sln /build Release
```

```
shell> devenv.com libmysql.sln /build RelWithDebInfo
```

For other versions of Visual Studio or `nmake` based build, run the following command:

```
shell> cmake --help
```

to check the supported generators.

To compile the Debug build, you must run set the `CMake` build type so the correct external library versions are used:

```
shell> cmake -G "Visual Studio 9 2008" -DCMAKE_BUILD_TYPE=Debug
```

Followed by:

```
shell> devenv.com libmysql.sln /build Debug
```

To Install

To create a install package you can choose between two packaging formats:

- **Zip package**

To create a Zip package, run the `cpack` command from the top-level MySQL Connector/C source directory.

- **MSI Install package**

The required tools include Windows XML Installer toolset (WIX), which is available [online](#).

To create the MSI install package, change location to the subdirectory `win` of the MySQL Connector/C source tree and generate the `Makefile`:

```
shell> cmake -G "NMake Makefiles"
```

Create the MSI install package by invoking `nmake`:

```
shell> nmake
```

Other Build Options

The following options can be used when building the MySQL Connector/C source code:

Table 22.28. Build Options for MySQL Connector/C

Build Option	Description
-DWITH_OPENSSL=1	Enables dynamic linking to the system OpenSSL library.
-DWITH_EXTERNAL_ZLIB=1	Enables dynamic linking to the system Zlib library.

22.5.2. Testing MySQL Connector/C

To test MySQL Connector/C, you need a running MySQL server instance. Before you run the test suite, specify the following environment variables:

- `MYSQL_TEST_HOST`: The host where the MySQL server is running (default `localhost`)
- `MYSQL_TEST_USER`: The user name of the MySQL account to use
- `MYSQL_TEST_PASSWD`: The password of the MySQL account to use

- `MYSQL_TEST_PORT`: The port to connect to
- `MYSQL_TEST_SOCKET`: The socket file to connect to
- `MYSQL_TEST_DB`: The default database to use (default `test`)

To run the test suite, execute `ctest` from the command line:

```
shell> ctest
```

22.5.3. MySQL Connector/C FAQ

Questions

- [23.5.3.1: \[3033\]](#) What is the “MySQL Native C API”? What are its typical benefits and use cases?
- [23.5.3.2: \[3033\]](#) What is “libmysql”?
- [23.5.3.3: \[3033\]](#) What is “libmysqld”?
- [23.5.3.4: \[3033\]](#) What is “MySQL Connector/C”?
- [23.5.3.5: \[3033\]](#) What is the difference between “Native C API”, “libmysql”, “libmysqld” and “MySQL Connector/C”?
- [23.5.3.6: \[3034\]](#) Does MySQL Connector/C replace any of “Native C API”, “libmysql” and “libmysqld”?

Questions and Answers

23.5.3.1: What is the “MySQL Native C API”? What are its typical benefits and use cases?

MySQL Connector/C, also known as `libmysql`, or MySQL Native C API, is a standalone, C-based API and library that you can use in C applications to connect to the MySQL Server. It implements the same MySQL client API that has been in use for a decade.

It is also used as the foundation for drivers for standard database APIs such as ODBC, Perl DBI, and Python DB API.

23.5.3.2: What is “libmysql”?

`libmysql` is the name of the library that MySQL Connector/C provides.

23.5.3.3: What is “libmysqld”?

`libmysqld` is an embedded database server with the same API as MySQL Connector/C. It is included with the MySQL Server distribution.

23.5.3.4: What is “MySQL Connector/C”?

MySQL Connector/C is a standalone distribution of the `libmysql` library, which was previously only available as part of the MySQL Server distribution. The version of `libmysql` included with MySQL Connector/C and the version bundled with the server are functionally equivalent, but the cross-platform build system for MySQL Connector/C uses `CMake`.

23.5.3.5: What is the difference between “Native C API”, “libmysql”, “libmysqld” and “MySQL Connector/C”?

MySQL Connector/C and `libmysql` are the “native C API for MySQL”, and all three terms can be used interchangeably. “`libmysqld`” is the embedded version of the MySQL Server, and is included in the server distribution.

23.5.3.6: Does MySQL Connector/C replace any of “Native C API”, “`libmysql`” and “`libmysqld`”?

MySQL Connector/C contains `libmysql`, and implements a native C API. It does not include `libmysqld`, which can be found with the MySQL server distribution.

22.6. MySQL Connector/Python

MySQL Connector/Python allows Python programs to access MySQL databases, using an API that is compliant with the [Python DB API version 2.0](#). It is written in pure Python and does not have any dependencies except for the [Python Standard Library](#).

MySQL Connector/Python includes support for:

- Almost all features provided by MySQL Server up to and including MySQL Server version 5.5.
- Converting parameter values back and forth between Python and MySQL data types, for example Python `datetime` and MySQL `DATETIME`. You can turn automatic conversion on for convenience, or off for optimal performance.
- All MySQL extensions to standard SQL syntax.
- Protocol compression, which enables compressing the data stream between the client and server.
- Connections using TCP/IP sockets and on Unix using Unix sockets.
- Secure TCP/IP connections using SSL.
- Self-contained driver. Connector/Python does not require the MySQL client library or any Python modules outside the standard library.

MySQL Connector/Python supports from Python version 2.4 through 2.7, and Python 3.1 and later. Note that Connector/Python does not support the old MySQL Server authentication methods, which means that MySQL versions prior to 4.1 will not work.

For release notes detailing the changes in each release of Connector/Python, see [MySQL Connector/Python Release Notes](#).

22.6.1. Guidelines for Python Developers

The following guidelines cover aspects of developing MySQL applications that might not be immediately obvious to developers coming from a Python background:

- For security, do not hardcode the values needed to connect and log into the database in your main script. Python has the convention of a `config.py` module, where you can keep such values separate from the rest of your code.
- Python scripts often build up and tear down large data structures in memory, up to the limits of available RAM. Because MySQL often deals with data sets that are many times larger than available memory, techniques that optimize storage space and disk I/O are especially important. For example, in MySQL tables, you typically use numeric IDs rather than string-based dictionary keys, so that the key values are compact and have a predictable length. This is especially important for columns that make up the [primary key](#) for an `InnoDB` table, because those column values are duplicated within each [secondary index](#).

- Any application that accepts input must expect to handle bad data.

The bad data might be accidental, such as out-of-range values or misformatted strings. The application can use server-side checks such as [unique constraints](#) and [NOT NULL constraints](#), to keep the bad data from ever reaching the database. On the client side, use techniques such as exception handlers to report any problems and take corrective action.

The bad data might also be deliberate, representing a “SQL injection” attack. For example, input values might contain quotation marks, semicolons, % and _ wildcard characters and other characters significant in SQL statements. Validate input values to make sure they have only the expected characters. Escape any special characters that could change the intended behavior when substituted into a SQL statement. Never concatenate a user input value into a SQL statement without doing validation and escaping first. Even when accepting input generated by some other program, expect that the other program could also be hacked and be sending you incorrect or malicious data.

- Because the result sets from SQL queries can be very large, use the appropriate method to retrieve items from the result set as you loop through them. [fetchone\(\)](#) retrieves a single item, when you know the result set contains a single row. [fetchall\(\)](#) retrieves all the items, when you know the result set contains a limited number of rows that can fit comfortably into memory. [fetchmany\(\)](#) is the general-purpose method when you cannot predict the size of the result set: you keep calling it and looping through the returned items, until there are no more results to process.
- Since Python already has convenient modules such as [pickle](#) and [cPickle](#) to read and write data structures on disk, the data that you choose store in MySQL instead is likely to have special characteristics:
 - **Too large to all fit in memory at one time.** You use [SELECT](#) statements to query only the precise items you need, and [aggregate functions](#) to perform calculations across multiple items. You configure the [innodb_buffer_pool_size \[1549\]](#) option within the MySQL server to dedicate a certain amount of RAM for caching query results.
 - **Too complex to be represented by a single data structure.** You divide the data between different SQL tables. You can recombine data from multiple tables by using a [join](#) query. You make sure that related data is kept in sync between different tables by setting up [foreign key](#) relationships.
 - **Updated frequently, perhaps by multiple users simultaneously.** The updates might only affect a small portion of the data, making it wasteful to write the whole structure each time. You use the SQL [INSERT](#), [UPDATE](#), and [DELETE](#) statements to update different items concurrently, writing only the changed values to disk. You use [InnoDB](#) tables and [transactions](#) to keep write operations from conflicting with each other, and to return consistent query results even as the underlying data is being updated.
- Building in MySQL best practices for performance can help your application to scale without requiring major rewrites and architectural changes. See [Chapter 8, Optimization](#) for best practices for MySQL performance. It offers guidelines and tips for SQL tuning, database design, and server configuration.
- You can avoid reinventing the wheel by learning the MySQL SQL statements for common operations: operators to use in queries, techniques for bulk loading data, and so on. Some statements and clauses are extensions to the basic ones defined by the SQL standard. See [Section 13.2, “Data Manipulation Statements”](#), [Section 13.1, “Data Definition Statements”](#), and [Section 13.2.9, “SELECT Syntax”](#) for the main classes of statements.
- Issuing SQL statements from Python typically involves declaring very long, possibly multi-line string literals. Because string literals within the SQL statements could be enclosed by single quotation, double quotation marks, or contain either of those characters, for simplicity you can use Python's triple-quoting mechanism to enclose the entire statement. For example:

```
'''It doesn't matter if this string contains 'single'
or "double" quotes, as long as there aren't 3 in a
row.'''
```

You can use either of the ' or " characters for triple-quoting multi-line string literals.

- Many of the secrets to a fast, scalable MySQL application involve using the right syntax at the very start of your setup procedure, in the `CREATE TABLE` statements. For example, Oracle recommends the `ENGINE=INNODB` clause for most tables, and makes it the default in MySQL 5.5 and up. Using `InnoDB` tables enables transactional behavior that helps scalability of read-write workloads and offers automatic [crash recovery](#). A follow-on recommendation is to declare a numeric [primary key](#) for each table, which offers the fastest way to look up values and can act as a pointer to associated values in other tables (a [foreign key](#)). Also within the `CREATE TABLE` statement, using the most compact column data types that meet your application requirements helps performance and scalability, as the database server moves large amounts of data back and forth between memory and disk.

22.6.2. Connector/Python Versions

MySQL Connector/Python v1.0.x series went going through a series of beta releases, leading to the first generally available (GA) version 1.0.7. Any development releases prior to general availability are not supported now that the GA version is released.

The following table summarizes the available Connector/Python versions:

Table 22.29. Connector/Python Version Reference

Connector/Python Version	MySQL Server Versions	Python Versions	Support Status for Connector
1.0	5.6, 5.5 (5.1, 5.0, 4.1)	2.7, 2.6 (2.5, 2.4); 3.1 and later	Recommended version



Note

MySQL server and Python versions within brackets are known to work with Connector/Python, but are not officially supported. Bugs might not get fixed for those versions.

22.6.3. Connector/Python Installation

Connector/Python runs on any platform where Python is installed. Python comes pre-installed on almost any Linux distribution or UNIX-like system such as Apple Mac OS X and FreeBSD. On Microsoft Windows systems, you can install Python using the installer found on the [Python Download website](#).

Connector/Python is a pure Python implementation of the MySQL Client/Server protocol, meaning it does not require any other MySQL client libraries or other components. It also has no third-party dependencies. If you need SSL support, verify that your Python installation has been compiled using the [OpenSSL](#) libraries.

The installation of Connector/Python is similar on every platform and follows the standard [Python Distribution Utilities](#) or [Distutils](#). Some platforms have specific packaging, for example RPM, and, when made available, the installation of these will be covered in this manual.

Python terminology regarding distributions:

- **Source Distribution** is a distribution that contains only source files and is generally platform independent.
- **Built Distribution** can be regarded as a binary package. It contains both sources and platform-independent bytecode.

22.6.3.1. Installing Connector/Python Source Distribution on Linux, UNIX, or OS X

On UNIX-like systems such as Linux distributions, Solaris, Apple Mac OS X, and FreeBSD, you can download Connector/Python as a `tar` archive from <http://dev.mysql.com/downloads/connector/python/>.

To install Connector/Python from the `.tar.gz` file, download the latest version and follow these steps:

```
shell> gunzip mysql-connector-python-1.0.6b1.tar.gz
shell> tar xf mysql-connector-python-1.0.6b1.tar
shell> cd mysql-connector-python-1.0.6b1
shell> sudo python setup.py install
```

On UNIX-like systems, Connector/Python gets installed in the default location `/prefix/lib/pythonX.Y/site-packages/`, where `prefix` is the location where Python was installed and `X.Y` is the version of Python. See [How installation works](#) in the Python manual.

If you are not sure where Connector/Python was installed, do the following to retrieve the location:

```
shell> python
>>> from distutils.sysconfig import get_python_lib

>>> print get_python_lib()           # Python v2.x
/Library/Python/2.7/site-packages

>>> print(get_python_lib())         # Python v3.x
/Library/Frameworks/Python.framework/Versions/3.1/lib/python3.1/site-packages
```



Note

The above example shows the default installation location on Mac OS X 10.7.

22.6.3.2. Installing Connector/Python Source Distribution on Microsoft Windows

On Microsoft Windows systems, you can download Connector/Python as a `zip` archive from <http://dev.mysql.com/downloads/connector/python/>.

Make sure that the Python executable is available in the Windows `%PATH%` setting. For more information about installation and configuration of Python on Windows, see the section [Using Python on Windows](#) in the Python documentation.

To install Connector/Python from the `.zip` file, download the latest version and follow these steps:

1. Unpack the downloaded `zip` archive into a directory of your choice. For example, into the folder `C:\mysql-connector\`. Use the appropriate unzip command for your system, for example, `unzip`, `pkunzip`, and so on.
2. Start a console window (or a DOS window) and change to the folder where you unpacked the Connector/Python `zip` archive.

```
shell> cd C:\mysql-connector\
```

3. Once inside the Connector/Python folder, do the following:

```
shell> python setup.py install
```

On Windows, Connector/Python gets installed in the default location `C:\PythonX.Y\Lib\site-packages\` where `X.Y` is the Python version you used to install the connector.

If you are not sure where Connector/Python ended up, do the following to retrieve the location where packages get installed:

```
shell> python
>>> from distutils.sysconfig import get_python_lib
>>> print get_python_lib()           # Python v2.x
>>> print(get_python_lib())         # Python v3.x
```

22.6.3.3. Verifying Your Connector/Python Installation

To test that your Connector/Python installation is working and is able to connect to a MySQL database server, you can run a very simple program where you substitute the login credentials and host information of the MySQL server. See [Section 22.6.4.1, “Connecting to MySQL Using Connector/Python”](#) for an example.

22.6.4. Connector/Python Coding Examples

These coding examples illustrate how to develop Python applications and scripts which connect to a MySQL Server using MySQL Connector/Python.

22.6.4.1. Connecting to MySQL Using Connector/Python

The `connect()` constructor is used for creating a connection to the MySQL server and returns a `MySQLConnection` object.

The following example shows how to connect to the MySQL server:

```
import mysql.connector

cnx = mysql.connector.connect(user='scott', password='tiger',
                              host='127.0.0.1',
                              database='employees')

cnx.close()
```

See [Section 22.6.6, “Connector/Python Connection Arguments”](#) for all possible connection arguments.

It is also possible to create connection objects using the `connection.MySQLConnection()` class. Both methods, using the `connect()` constructor, or the class directly, are valid and functionally equal, but using `connector()` is preferred and will be used in most examples in this manual.

To handle connection errors, use the `try` statement and catch all errors using the `errors.Error` exception:

```
import mysql.connector
from mysql.connector import errorcode

try:
    cnx = mysql.connector.connect(user='scott',
                                  database='testt')
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("Something is wrong your username or password")
    elif err.errno == errorcode.ER_BAD_DB_ERROR:
```

```

    print("Database does not exists")
else:
    print(err)
else:
    cnx.close()

```

If you have lots of connection arguments, it's best to keep them in a dictionary and use the `**`-operator. Here is an example:

```

import mysql.connector

config = {
    'user': 'scott',
    'password': 'tiger',
    'host': '127.0.0.1',
    'database': 'employees',
    'raise_on_warnings': True,
}

cnx = mysql.connector.connect(**config)

cnx.close()

```

22.6.4.2. Creating Tables Using Connector/Python

All [DDL](#) (Data Definition Language) statements are executed using a handle structure known as a cursor. The following examples show how to create the tables of the `employees` database. You will need them for the other examples.

In a MySQL server, tables are very long-lived objects, and are often accessed by multiple applications written in different languages. You might typically work with tables that are already set up, rather than creating them within your own application. Avoid setting up and dropping tables over and over again, as that is an expensive operation. The exception is [temporary tables](#), which can be created and dropped quickly within an application.

```

from __future__ import print_function

import mysql.connector
from mysql.connector import errorcode

DB_NAME = 'employees'

TABLES = {}
TABLES['employees'] = (
    "CREATE TABLE `employees` ("
    "  `emp_no` int(11) NOT NULL AUTO_INCREMENT,"
    "  `birth_date` date NOT NULL,"
    "  `first_name` varchar(14) NOT NULL,"
    "  `last_name` varchar(16) NOT NULL,"
    "  `gender` enum('M','F') NOT NULL,"
    "  `hire_date` date NOT NULL,"
    "  PRIMARY KEY (`emp_no`)"
    ") ENGINE=InnoDB")

TABLES['departments'] = (
    "CREATE TABLE `departments` ("
    "  `dept_no` char(4) NOT NULL,"
    "  `dept_name` varchar(40) NOT NULL,"
    "  PRIMARY KEY (`dept_no`), UNIQUE KEY `dept_name` (`dept_name`)"
    ") ENGINE=InnoDB")

TABLES['salaries'] = (
    "CREATE TABLE `salaries` ("
    "  `emp_no` int(11) NOT NULL,"

```

```

" `salary` int(11) NOT NULL,"
" `from_date` date NOT NULL,"
" `to_date` date NOT NULL,"
" PRIMARY KEY (`emp_no`,`from_date`), KEY `emp_no` (`emp_no`),"
" CONSTRAINT `salaries_ibfk_1` FOREIGN KEY (`emp_no`) "
" REFERENCES `employees` (`emp_no`) ON DELETE CASCADE"
") ENGINE=InnoDB"

TABLES['dept_emp'] = (
"CREATE TABLE `dept_emp` ("
" `emp_no` int(11) NOT NULL,"
" `dept_no` char(4) NOT NULL,"
" `from_date` date NOT NULL,"
" `to_date` date NOT NULL,"
" PRIMARY KEY (`emp_no`,`dept_no`), KEY `emp_no` (`emp_no`),"
" KEY `dept_no` (`dept_no`),"
" CONSTRAINT `dept_emp_ibfk_1` FOREIGN KEY (`emp_no`) "
" REFERENCES `employees` (`emp_no`) ON DELETE CASCADE,"
" CONSTRAINT `dept_emp_ibfk_2` FOREIGN KEY (`dept_no`) "
" REFERENCES `departments` (`dept_no`) ON DELETE CASCADE"
") ENGINE=InnoDB"

TABLES['dept_manager'] = (
" CREATE TABLE `dept_manager` ("
" `dept_no` char(4) NOT NULL,"
" `emp_no` int(11) NOT NULL,"
" `from_date` date NOT NULL,"
" `to_date` date NOT NULL,"
" PRIMARY KEY (`emp_no`,`dept_no`),"
" KEY `emp_no` (`emp_no`),"
" KEY `dept_no` (`dept_no`),"
" CONSTRAINT `dept_manager_ibfk_1` FOREIGN KEY (`emp_no`) "
" REFERENCES `employees` (`emp_no`) ON DELETE CASCADE,"
" CONSTRAINT `dept_manager_ibfk_2` FOREIGN KEY (`dept_no`) "
" REFERENCES `departments` (`dept_no`) ON DELETE CASCADE"
") ENGINE=InnoDB"

TABLES['titles'] = (
"CREATE TABLE `titles` ("
" `emp_no` int(11) NOT NULL,"
" `title` varchar(50) NOT NULL,"
" `from_date` date NOT NULL,"
" `to_date` date DEFAULT NULL,"
" PRIMARY KEY (`emp_no`,`title`,`from_date`), KEY `emp_no` (`emp_no`),"
" CONSTRAINT `titles_ibfk_1` FOREIGN KEY (`emp_no`) "
" REFERENCES `employees` (`emp_no`) ON DELETE CASCADE"
") ENGINE=InnoDB"

```

The above code shows how we are storing the `CREATE` statements in a Python dictionary called `TABLES`. We also define the database in a global variable called `DB_NAME`, which allows you to easily use a different schema.

```

cnx = mysql.connector.connect(user='scott')
cursor = cnx.cursor()

```

A single MySQL server can contain multiple [databases](#). Typically, you specify the database to switch to when connecting to the MySQL server. This example does not connect to the database upon connection, so that it can make sure the database exists, and create it if not.

```

def create_database(cursor):
    try:
        cursor.execute(
            "CREATE DATABASE {} DEFAULT CHARACTER SET 'utf8'".format(DB_NAME))
    except mysql.connector.Error as err:
        print("Failed creating database: {}".format(err))
        exit(1)

```

```

try:
    cnx.database = DB_NAME
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_BAD_DB_ERROR:
        create_database(cursor)
        cnx.database = DB_NAME
    else:
        print(err)
        exit(1)

```

We first try to change to a particular database using the `database` property of the connection object `cnx`. If there is an error, we examine the error number to check if the database does not exist. If so, we call the `create_database` function to create it for us.

On any other error, the application exits and displays the error message.

```

for name, ddl in TABLES.iteritems():
    try:
        print("Creating table {}: ".format(name), end='')
        cursor.execute(ddl)
    except mysql.connector.Error as err:
        if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
            print("already exists.")
        else:
            print(err.errmsg)
    else:
        print("OK")

cursor.close()
cnx.close()

```

After we successfully created or changed to the target database, we create the tables by iterating over the items of the `TABLES` dictionary.

We handle the error when the table already exists by simply notifying the user that it was already there. Other errors are printed, but we simply continue creating tables. (We show how to handle the “table already exists” condition for illustration purposes. In a real application, we would typically avoid the error condition entirely by using the `IF NOT EXISTS` clause of the `CREATE TABLE` statement.)

The output would be something like this:

```

Creating table employees: already exists.
Creating table salaries: already exists.
Creating table titles: OK
Creating table departments: already exists.
Creating table dept_manager: already exists.
Creating table dept_emp: already exists.

```

To populate the employees tables, use the dump files of the [Employee Sample Database](#). Note that you only need the data dump files that you will find in an archive named like `employees_db-dump-files-1.0.5.tar.bz2`. After downloading the dump files, do the following from the command line, adding connection options to the `mysql` commands if necessary:

```

shell> tar xzf employees_db-dump-files-1.0.5.tar.bz2
shell> cd employees_db
shell> mysql employees < load_employees.dump
shell> mysql employees < load_titles.dump
shell> mysql employees < load_departments.dump
shell> mysql employees < load_salaries.dump
shell> mysql employees < load_dept_emp.dump
shell> mysql employees < load_dept_manager.dump

```

22.6.4.3. Inserting Data Using Connector/Python

Inserting or updating data is also done using the handler structure known as a cursor. When you use a transactional storage engine such as [InnoDB](#) (which is the default in MySQL 5.5 and later), you must [commit](#) the data after a sequence of [INSERT](#), [DELETE](#), and [UPDATE](#) statements.

In this example we show how to insert new data. The second [INSERT](#) depends on the value of the newly created [primary key](#) of the first. We are also demonstrating how to use extended formats. The task is to add a new employee starting to work tomorrow with a salary set to 50000.



Note

The following example uses tables created in the example [Section 22.6.4.2, “Creating Tables Using Connector/Python”](#). The [AUTO_INCREMENT](#) column option for the primary key of the [employees](#) table is important to ensure reliable, easily searchable data.

```
from __future__ import print_function
from datetime import date, datetime, timedelta
import mysql.connector

cnx = mysql.connector.connect(user='scott', database='employees')
cursor = cnx.cursor()

tomorrow = datetime.now().date() + timedelta(days=1)

add_employee = ("INSERT INTO employees "
               "(first_name, last_name, hire_date, gender, birth_date) "
               "VALUES (%s, %s, %s, %s, %s)")
add_salary = ("INSERT INTO salaries "
              "(emp_no, salary, from_date, to_date) "
              "VALUES (%(emp_no)s, %(salary)s, %(from_date)s, %(to_date)s)")

data_employee = ('Geert', 'Vanderkelen', tomorrow, 'M', date(1977, 6, 14))

# Insert new employee
cursor.execute(add_employee, data_employee)
emp_no = cursor.lastrowid

# Insert salary information
data_salary = {
    'emp_no': emp_no,
    'salary': 50000,
    'from_date': tomorrow,
    'to_date': date(9999, 1, 1),
}
cursor.execute(add_salary, data_salary)

# Make sure data is committed to the database
cnx.commit()

cursor.close()
cnx.close()
```

We first open a connection to the MySQL server and store the [connection object](#) in the variable `cnx`. We then create a new cursor, by default a [MySQLCursor](#) object, using the connection's `cursor()` method.

We could calculate tomorrow by calling a database function, but for clarity we do it in Python using the [datetime](#) module.

Both [INSERT](#) statements are stored in the variables called `add_employee` and `add_salary`. Note that the second [INSERT](#) statement uses extended Python format codes.

The information of the new employee is stored in the tuple `data_employee`. The query to insert the new employee is executed and we retrieve the newly inserted value for the column `emp_no` using the `lastrowid` property of the cursor object.

Next, we insert the new salary for the new employee. We are using the `emp_no` variable in the dictionary holding the data. This dictionary is passed to the `execute()` method of the cursor object.

Since by default Connector/Python turns `autocommit` off, and MySQL 5.5 and later uses transactional `InnoDB` tables by default, it is necessary to commit your changes using the connection's `commit()` method. You could also `roll back` using the `rollback()` method.

22.6.4.4. Querying Data Using Connector/Python

The following example shows how to `query` data using a cursor created using the connection's `cursor()` method. The data returned is formatted and printed on the console.

The task is to select all employees hired in the year 1999 and print their names with their hire date to the console.

```
import datetime
import mysql.connector

cnx = mysql.connector.connect(user='scott', database='employees')
cursor = cnx.cursor()

query = ("SELECT first_name, last_name, hire_date FROM employees "
        "WHERE hire_date BETWEEN %s AND %s")

hire_start = datetime.date(1999, 1, 1)
hire_end = datetime.date(1999, 12, 31)

cursor.execute(query, (hire_start, hire_end))

for (first_name, last_name, hire_date) in cursor:
    print("{} , {} was hired on {:%d %b %Y}".format(
        last_name, first_name, hire_date))

cursor.close()
cnx.close()
```

We first open a connection to the MySQL server and store the `connection object` in the variable `cnx`. We then create a new cursor, by default a `MySQLCursor` object, using the connection's `cursor()` method.

In the preceding example, we store the `SELECT` statement in the variable `query`. Note that we are using unquoted `%s`-markers where dates should have been. Connector/Python converts `hire_start` and `hire_end` from Python types to a data type that MySQL understands and adds the required quotes. In this case, it replaces the first `%s` with `'1999-01-01'`, and the second with `'1999-12-31'`.

We then execute the operation stored in the `query` variable using the `execute()` method. The data used to replace the `%s`-markers in the query is passed as a tuple: `(hire_start, hire_end)`.

After executing the query, the MySQL server is ready to send the data. The result set could be zero rows, one row, or 100 million rows. Depending on the expected volume, you can use different techniques to process this result set. In this example, we use the `cursor` object as an iterator. The first column in the row will be stored in the variable `first_name`, the second in `last_name`, and the third in `hire_date`.

We print the result, formatting the output using Python's built-in `format()` function. Note that `hire_date` was converted automatically by Connector/Python to a Python `datetime.date` object. This means that we can easily format the date in a more human-readable form.

The output should be something like this:

```
..
Wilharm, LiMin was hired on 16 Dec 1999
Wielonsky, Lalit was hired on 16 Dec 1999
Kamble, Dannz was hired on 18 Dec 1999
DuBourdieu, Zhongwei was hired on 19 Dec 1999
Fujisawa, Rosita was hired on 20 Dec 1999
..
```

22.6.5. Connector/Python Tutorials

These tutorials illustrate how to develop Python applications and scripts that connect to a MySQL database server using MySQL Connector/Python.

22.6.5.1. Tutorial: Raise employee's salary using a buffering cursor

The following example script will give a long-overdue raise effective tomorrow to all employees who joined in the year 2000 and are still with the company.

We are using buffered cursors to iterate through the selected employees. This way we do not have to fetch the rows in a new variables, but can instead use the cursor as an iterator.

Note that the script is an example; there are other ways of doing this simple task.

```
from __future__ import print_function

from decimal import Decimal
from datetime import datetime, date, timedelta

import mysql.connector

# Connect with the MySQL Server
cnx = mysql.connector.connect(user='scott', database='employees')

# Get two buffered cursors
curA = cnx.cursor(buffered=True)
curB = cnx.cursor(buffered=True)

# Query to get employees who joined in a period defined by two dates
query = (
    "SELECT s.emp_no, salary, from_date, to_date FROM employees AS e "
    "LEFT JOIN salaries AS s USING (emp_no) "
    "WHERE to_date = DATE('9999-01-01') "
    "AND e.hire_date BETWEEN DATE(%s) AND DATE(%s)")

# UPDATE and INSERT statements for the old and new salary
update_old_salary = (
    "UPDATE salaries SET to_date = %s "
    "WHERE emp_no = %s AND from_date = %s")
insert_new_salary = (
    "INSERT INTO salaries (emp_no, from_date, to_date, salary) "
    "VALUES (%s, %s, %s, %s)")

# Select the employees getting a raise
curA.execute(query, (date(2000, 1, 1), date(2001, 1, 1)))

# Iterate through the result of curA
for (emp_no, salary, from_date, to_date) in curA:

    # Update the old and insert the new salary
    new_salary = int(round(salary * Decimal('1.15')))
    curB.execute(update_old_salary, (tomorrow, emp_no, from_date))
    curB.execute(insert_new_salary,
```

```
(emp_no, tomorrow, date(9999, 1, 1), new_salary))

# Commit the changes
cnx.commit()

cnx.close()
```

22.6.6. Connector/Python Connection Arguments

The following lists the arguments which can be used to initiate a connection with the MySQL server using either:

- Function `mysql.connector.connect()`
- Class `mysql.connector.MySQLConnection()`

Table 22.30. Connection Arguments for Connector/Python

Argument Name	Default	Description
<code>user</code> (<code>username*</code>)		The username used to authenticate with the MySQL Server.
<code>password</code> (<code>passwd*</code>)		The password to authenticate the user with the MySQL Server.
<code>database</code> (<code>db*</code>)		Database name to use when connecting with the MySQL Server.
<code>host</code>	127.0.0.1	Hostname or IP address of the MySQL Server.
<code>port</code>	3306	TCP/IP port of the MySQL Server. Must be an integer.
<code>unix_socket</code>		The location of the Unix socket file.
<code>use_unicode</code>	True	Whether to use Unicode or not.
<code>charset</code>	utf8	Which MySQL character set to use.
<code>collation</code>	utf8_general_ci	Which MySQL collation to use.
<code>autocommit</code>	False	Whether to autocommit transactions.
<code>time_zone</code>		Set the <code>time_zone</code> session variable at connection.
<code>sql_mode</code>		Set the <code>sql_mode</code> session variable at connection.
<code>get_warnings</code>	False	Whether to fetch warnings.
<code>raise_on_warnings</code>	False	Whether to raise an exception on warnings.
<code>connection_timeout</code> (<code>connect_timeout*</code>)		Timeout for the TCP and Unix socket connections.
<code>client_flags</code>		MySQL client flags.
<code>buffered</code>	False	Whether cursor object fetches the result immediately after executing query.
<code>raw</code>	False	Whether MySQL results are returned as-is, rather than converted to Python types.
<code>ssl_ca</code>		File containing the SSL certificate authority.
<code>ssl_cert</code>		File containing the SSL certificate file.
<code>ssl_key</code>		File containing the SSL key.
<code>dsn</code>		Not supported (raises <code>NotSupportedError</code> when used).

* Synonymous argument name, available only for compatibility with other Python MySQL drivers. Oracle recommends not to use these alternative names.

Authentication with MySQL will use `username` and `password`. Note that MySQL Connector/Python does not support the old, insecure password protocols of MySQL versions prior to 4.1.

When the `database` parameter is given, the current database is set to the given value. To later change the database, execute the MySQL `USE` command or set the `database` property of the `MySQLConnection` instance.

By default, Connector/Python tries to connect to a MySQL server running on `localhost` using TCP/IP. The `host` argument defaults to IP address 127.0.0.1 and `port` to 3306. Unix sockets are supported by setting `unix_socket`. Named pipes on the Windows platform are not supported.

Strings coming from MySQL are by default returned as Python Unicode literals. To change this behavior, set `use_unicode` to `False`. You can change the character setting for the client connection through the `charset` argument. To change the character set after connecting to MySQL, set the `charset` property of the `MySQLConnection` instance. This technique is preferred over using the MySQL `SET NAMES` statement directly. Similar to the `charset` property, you can set the `collation` for the current MySQL session.

Transactions are not automatically committed; call the `commit()` method of the `MySQLConnection` instance within your application after doing a set of related insert, update, and delete operations. For data consistency and high throughput for write operations, it is best to leave the `autocommit` configuration option turned off when using `InnoDB` or other transactional tables.

The time zone can be set per connection using the `time_zone` argument. This is useful if the MySQL server is set, for example, to UTC and `TIMESTAMP` values should be returned by MySQL converted to the `PST` time zone.

MySQL supports so called SQL Modes, which will change the behavior of the server globally or per connection. For example, to have warnings raised as errors, set `sql_mode` to `TRADITIONAL`. For more information, see [Section 5.1.7, “Server SQL Modes”](#).

Warnings generated by queries are fetched automatically when `get_warnings` is set to `True`. You can also immediately raise an exception by setting `raise_on_warnings` to `True`. Consider using the MySQL `sql_mode` setting for turning warnings into errors.

To set a timeout value for connections, use `connection_timeout`.

MySQL uses `client flags` to enable or disable features. Using the `client_flags` argument, you have control of what is set. To find out what flags are available, use the following:

```
from mysql.connector.constants import ClientFlag
print '\n'.join(ClientFlag.get_full_info())
```

If `client_flags` is not specified (that is, it is zero), defaults are used for MySQL v4.1 and later. If you specify an integer greater than 0, make sure all flags are set. A better way to set and unset flags is to use a list. For example, to set `FOUND_ROWS`, but disable the default `LONG_FLAG`:

```
flags = [ClientFlag.FOUND_ROWS, -ClientFlag.LONG_FLAG]
mysql.connector.connect(client_flags=flags)
```

By default, MySQL Connector/Python does not buffer or pre-fetch results. This means that after a query is executed, your program is responsible of fetching the data. This avoids using excessive memory when queries return large result sets. If you know that the result set is small enough to handle all at once, fetching the results immediately by setting `buffered` to `True`. It is also possible to set this per cursor (see cursor manual).

MySQL types will be converted automatically to Python types. For example, a `DATETIME` column becomes a `datetime.datetime` object. When conversion should be done differently, for example to get better performance, set `raw` to `True`.

Using SSL connections is possible when your [Python installation supports SSL](#), that is, when it is compiled against the OpenSSL libraries. When you provide the arguments `ssl_ca`, `ssl_key` and `ssl_cert`, the connection switches to SSL. You can use this in combination with the `compressed` argument set to `True`.

`passwd`, `db` and `connect_timeout` are valid for compatibility with other MySQL interfaces and are respectively the same as `password`, `database` and `connection_timeout`. The latter take precedence. Data source name syntax or `dsn` is not used; if specified, it raises a `NotSupportedError` exception.

22.6.7. Connector/Python API Reference

This section contains the public API reference of Connector/Python. Although valid for both Python 2 and Python 3, examples should be considered working for Python 2.7, and Python 3.1 and greater.

The following overview shows the `mysql.connector` package with its modules. Currently, only the most useful modules, classes and functions for end users are documented.

```
mysql.connector
  errorcode
  errors
  connection
  constants
  conversion
  cursor
  dbapi
  locales
    eng
      client_error
  protocol
  utils
```

22.6.7.1. Errors and Exceptions

The `mysql.connector.errors` module defines exception classes for errors and warnings raised by MySQL Connector/Python. Most classes defined in this module are available when you import `mysql.connector`.

The exception classes defined in this module follow mostly the Python Database Specification v2.0 (PEP-249). For some MySQL client or server errors it is not always clear which exception to raise. It is good to discuss whether an error should be reclassified by opening a bug report.

MySQL Server errors are mapped with Python exception based on their SQLState (see [Section C.3, "Server Error Codes and Messages"](#)). The following list shows the `SQLState` classes and the exception Connector/Python will raise. It is, however, possible to redefine which exception is raised for each server error. Note that the default exception is `DatabaseError`.

- `02`: `DataError`
- `07`: `DatabaseError`
- `08`: `OperationalError`
- `0A`: `NotSupportedError`
- `21`: `DataError`

- [22](#): `DataError`
- [23](#): `IntegrityError`
- [24](#): `ProgrammingError`
- [25](#): `ProgrammingError`
- [26](#): `ProgrammingError`
- [27](#): `ProgrammingError`
- [28](#): `ProgrammingError`
- [2A](#): `ProgrammingError`
- [2B](#): `DatabaseError`
- [2C](#): `ProgrammingError`
- [2D](#): `DatabaseError`
- [2E](#): `DatabaseError`
- [33](#): `DatabaseError`
- [34](#): `ProgrammingError`
- [35](#): `ProgrammingError`
- [37](#): `ProgrammingError`
- [3C](#): `ProgrammingError`
- [3D](#): `ProgrammingError`
- [3F](#): `ProgrammingError`
- [40](#): `InternalError`
- [42](#): `ProgrammingError`
- [44](#): `InternalError`
- [HZ](#): `OperationalError`
- [XA](#): `IntegrityError`
- [0K](#): `OperationalError`
- [HY](#): `DatabaseError`

22.6.7.1.1. Module `errorcode`

This module contains both MySQL server and client error codes defined as module attributes with the error number as value. Using error codes instead of error numbers could make reading the source code a bit easier.

```
>>> from mysql.connector import errorcode
```

```
>>> errorcode.ER_BAD_TABLE_ERROR
1051
```

See [Section C.3, “Server Error Codes and Messages”](#) and [Section C.4, “Client Error Codes and Messages”](#).

22.6.7.1.2. Exception `errors.Error`

This exception is the base class for all other exceptions in the `errors` module. It can be used to catch all errors in a single `except` statement.

The following example shows how we could catch syntax errors:

```
import mysql.connector
try:
    cnx = mysql.connector.connect(user='scott', database='employees')
    cursor = cnx.cursor()
    cursor.execute("SELECT * FORM employees") # Syntax error in query
    cnx.close()
except mysql.connector.Error as err:
    print("Something went wrong: {}".format(err))
```

Initializing the exception supports a few optional arguments, namely `msg`, `errno`, `values` and `sqlstate`. All of them are optional and default to `None`. `errors.Error` is internally used by Connector/Python to raise MySQL client and server errors and should not be used by your application to raise exceptions.

The following examples show the result when using no or a combination of the arguments:

```
>>> from mysql.connector.errors import Error
>>> str(Error())
'Unknown error'

>>> str(Error("Oops! There was an error."))
'Oops! There was an error.'

>>> str(Error(errno=2006))
'2006: MySQL server has gone away'

>>> str(Error(errno=2002, values=('/tmp/mysql.sock', 2)))
'2002: Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)'

>>> str(Error(errno=1146, sqlstate='42S02', msg="Table 'test.spam' doesn't exist"))
'1146 (42S02): Table 'test.spam' doesn't exist'
```

The example which uses error number 1146 is used when Connector/Python receives an error packet from the MySQL Server. The information is parsed and passed to the `Error` exception as shown.

Each exception subclassing from `Error` can be initialized using the above mentioned arguments. Additionally, each instance has the attributes `errno`, `msg` and `sqlstate` which can be used in your code.

The following example shows how to handle errors when dropping a table which does not exist (when you do not want to use the `IF EXISTS` clause):

```
import mysql.connector
from mysql.connector import errorcode

cnx = mysql.connector.connect(user='scott', database='test')
try:
    cur.execute("DROP TABLE spam")
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_BAD_TABLE_ERROR:
        print("Creating table spam")
```

```
else:  
    raise
```

`errors.Error` is a subclass of the Python `StandardError`.

22.6.7.1.3. Exception `errors.Warning`

This exception is used for reporting important warnings, however, Connector/Python does not use it. It is included to be compliant with the Python Database Specification v2.0 (PEP-249).

Consider using either more strict [Server SQL Modes](#) or the `raise_on_warnings` connection argument to make Connector/Python raise errors when your queries produce warnings.

`errors.Warning` is a subclass of the Python `StandardError`.

22.6.7.1.4. Exception `errors.InterfaceError`

This exception is raised for errors originating from Connector/Python itself, not related to the MySQL server.

`errors.InterfaceError` is a subclass of `errors.Error`.

22.6.7.1.5. Exception `errors.DatabaseError`

This exception is the default for any MySQL error which does not fit the other exceptions.

`errors.DatabaseError` is a subclass of `errors.Error`.

22.6.7.1.6. Exception `errors.InternalError`

This exception is raised when the MySQL server encounters an internal error, for example, when a deadlock occurred.

`errors.InternalError` is a subclass of `errors.DatabaseError`.

22.6.7.1.7. Exception `errors.OperationalError`

This exception is raised for errors which are related to MySQL's operations. For example, too many connections, a hostname could not be resolved, bad handshake, server is shutting down, communication errors, and so on.

`errors.OperationalError` is a subclass of `errors.DatabaseError`.

22.6.7.1.8. Exception `errors.ProgrammingError`

This exception is raised on programming errors, for example when you have a syntax error in your SQL or a table was not found.

The following example shows how to handle syntax errors:

```
try:  
    cursor.execute("CREATE DESK t1 (id int, PRIMARY KEY (id))")  
except mysql.connector.ProgrammingError as err:  
    if err.errno == errorcode.ER_SYNTAX_ERROR:  
        print("Check your syntax!")  
    else:  
        print("Error: {}".format(err))
```

`errors.ProgrammingError` is a subclass of `errors.DatabaseError`.

22.6.7.1.9. Exception `errors.IntegrityError`

This exception is raised when the relational integrity of the data is affected. For example, a duplicate key was inserted or a foreign key constraint would fail.

The following example shows a duplicate key error raised as `IntegrityError`:

```
cursor.execute("CREATE TABLE t1 (id int, PRIMARY KEY (id))")
try:
    cursor.execute("INSERT INTO t1 (id) VALUES (1)")
    cursor.execute("INSERT INTO t1 (id) VALUES (1)")
except mysql.connector.IntegrityError as err:
    print("Error: {}".format(err))
```

`errors.IntegrityError` is a subclass of `errors.DatabaseError`.

22.6.7.1.10. Exception `errors.DataError`

This exception is raised when there were problems with the data. Examples are a column set to NULL when it can not, out of range values for a column, division by zero, column count does not match value count, and so on.

`errors.DataError` is a subclass of `errors.DatabaseError`.

22.6.7.1.11. Exception `errors.NotSupportedError`

This exception is raised in case some feature was used but not supported by the version of MySQL which returned the error. It is also raised when using functions or statements which are not supported by stored routines.

`errors.NotSupportedError` is a subclass of `errors.DatabaseError`.

22.6.7.1.12. Function `errors.custom_error_exception(error=None, exception=None)`

This function defines custom exceptions for MySQL server errors and returns current customizations.

If `error` is a MySQL Server error number, then you have to pass also the `exception` class. The `error` argument can also be a dictionary in which case the key is the server error number, and value the class of the exception to be raised.

To reset the customizations, simply supply an empty dictionary.

```
import mysql.connector
from mysql.connector import errorcode

# Server error 1028 should raise a DatabaseError
mysql.connector.custom_error_exception(1028, mysql.connector.DatabaseError)

# Or using a dictionary:
mysql.connector.custom_error_exception({
    1028: mysql.connector.DatabaseError,
    1029: mysql.connector.OperationalError,
})

# To reset, pass an empty dictionary:
mysql.connector.custom_error_exception({})
```

22.6.7.2. Class `connection.MySQLConnection`

The `MySQLConnection` class is used to open and manage a connection to a MySQL server. It also used to send commands and SQL queries and read result.

22.6.7.2.1. Constructor `connection.MySQLConnection(**kwargs)`

The `MySQLConnection` constructor initializes the attributes and when at least one argument is passed, it tries to connect with the MySQL server.

For a complete list of arguments, see [Section 22.6.6, “Connector/Python Connection Arguments”](#).

22.6.7.2.2. Method `MySQLConnection.close()`

See [`disconnect\(\)`](#).

Returns a tuple.

22.6.7.2.3. Method `MySQLConnection.config(**kwargs)`

Allows to configure a `MySQLConnection` instance after it was instantiated. See [Section 22.6.6, “Connector/Python Connection Arguments”](#) for a complete list of possible arguments.

You could use the `config()` method to change, for example, the username and call `reconnect()`.

```
cnx = MySQLConnection(user='joe', database='test')
# Connected as 'joe'
cnx.config(user='jane')
cnx.reconnect()
# Now connected as 'jane'
```

22.6.7.2.4. Method `MySQLConnection.connect(**kwargs)`

This method sets up the connection to the MySQL server. If no arguments are given, it uses the already configured or default values. See [Section 22.6.6, “Connector/Python Connection Arguments”](#) for a complete list of possible arguments.

22.6.7.2.5. Method `MySQLConnection.commit()`

This method sends the `COMMIT` command to the MySQL server, committing the current transaction. Since by default, Connector/Python does not auto commit, it is important to call this method after every transaction which updates data for tables using transactional storage engines.

See the [`rollback\(\)`](#) method for rolling back transactions.

```
>>> cursor.execute("INSERT INTO employees (first_name) VALUES (%s)", ('Jane'))
>>> cnx.commit()
```

22.6.7.2.6. Method `MySQLConnection.cursor(buffered=None, raw=None, cursor_class=None)`

This method returns a `MySQLCursor()` object, or a subclass of it depending the passed arguments.

When `buffered` is `True`, the cursor will fetch all rows after the operation is executed. This is useful when queries return small result sets. Setting `raw` will skip the conversion from MySQL data types to Python types when fetching rows. `Raw` is usually used when you want to have more performance and/or you want to do the conversion yourself.

The `cursor_class` argument can be used to pass a class to use for instantiating a new cursor. It has to be a subclass of `cursor.CursorBase`.

The returned object depends on the combination of the `buffered` and `raw` arguments.

- If not buffered and not raw: `cursor.MySQLCursor`
- If buffered and not raw: `cursor.MySQLCursorBuffered`
- If buffered and raw: `cursor.MySQLCursorBufferedRaw`
- If not buffered and raw: `cursor.MySQLCursorRaw`

Returns a `CursorBase` instance.

22.6.7.2.7. Method `MySQLConnection.cmd_change_user(username='', password='', database='', charset=33)`

Changes the user using `username` and `password`. It also causes the specified `database` to become the default (current) database. It is also possible to change the character set using the `charset` argument.

Returns a dictionary containing the OK packet information.

22.6.7.2.8. Method `MySQLConnection.cmd_debug()`

Instructs the server to write some debug information to the log. For this to work, the connected user must have the `SUPER` privilege.

Returns a dictionary containing the OK packet information.

22.6.7.2.9. Method `MySQLConnection.cmd_init_db(database)`

This method makes specified database the default (current) database. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

Returns a dictionary containing the OK packet information.

22.6.7.2.10. Method `MySQLConnection.cmd_ping()`

Checks whether the connection to the server is working.

This method is not to be used directly. Use `ping()` or `is_connected()` instead.

Returns a dictionary containing the OK packet information.

22.6.7.2.11. Method `MySQLConnection.cmd_process_info()`

This method raises the `NotSupportedError` exception. Instead, use the `SHOW PROCESSLIST` statement or query the tables found in the database `INFORMATION_SCHEMA`.

22.6.7.2.12. Method `MySQLConnection.cmd_process_kill(mysql_pid)`

Asks the server to kill the thread specified by `mysql_pid`. Although still available, it's better to use the SQL `KILL` command.

Returns a dictionary containing the OK packet information.

The following two lines do the same:

```
>>> cnx.cmd_process_kill(123)
>>> cnx.cmd_query('KILL 123')
```

22.6.7.2.13. Method `MySQLConnection.cmd_quit()`

This method sends the `QUIT` command to the MySQL server, closing the current connection. Since there is no response from the MySQL, the packet that was sent is returned.

22.6.7.2.14. Method `MySQLConnection.cmd_query(statement)`

This method sends the given `statement` to the MySQL server and returns a result. If you need to send multiple statements, you have to use the `cmd_query_iter()` method.

The returned dictionary contains information depending on what kind of query was executed. If the query is a `SELECT` statement, the result contains information about columns. Other statements return a dictionary containing OK or EOF packet information.

Errors received from the MySQL server are raised as exceptions. An `InterfaceError` is raised when multiple results are found.

Returns a dictionary.

22.6.7.2.15. Method `MySQLConnection.cmd_query_iter(statement)`

Similar to the `cmd_query()` method, but returns a generator object to iterate through results. Use `cmd_query_iter()` when sending multiple statements, and separate the statements with semicolons.

The following example shows how to iterate through the results after sending multiple statements:

```
statement = 'SELECT 1; INSERT INTO t1 VALUES (); SELECT 2'
for result in cnx.cmd_query(statement, iterate=True):
    if 'columns' in result:
        columns = result['columns']
        rows = cnx.get_rows()
    else:
        # do something useful with INSERT result
```

Returns a generator object.

22.6.7.2.16. Method `MySQLConnection.cmd_refresh(options)`

This method flushes tables or caches, or resets replication server information. The connected user must have the `RELOAD` privilege.

The `options` argument should be a bitwise value using constants from the class `constants.RefreshOption`.

See [Section 22.6.7.9, “Class `constants.RefreshOption`”](#) for a list of options.

Example:

```
>>> from mysql.connector import RefreshOption
>>> refresh = RefreshOption.LOG | RefreshOption.THREADS
>>> cnx.cmd_refresh(refresh)
```

22.6.7.2.17. Method `MySQLConnection.cmd_shutdown()`

Asks the database server to shut down. The connected user must have the `SHUTDOWN` privilege.

Returns a dictionary containing the OK packet information.

22.6.7.2.18. Method `MySQLConnection.cmd_statistics()`

Returns a dictionary containing information about the MySQL server including uptime in seconds and the number of running threads, questions, reloads, and open tables.

22.6.7.2.19. Method `MySQLConnection.disconnect()`

This method tries to send the `QUIT` command and close the socket. It does not raise any exceptions.

`MySQLConnection.close()` is a synonymous method name and more commonly used.

22.6.7.2.20. Method `MySQLConnection.get_rows(count=None)`

This method retrieves all or remaining rows of a query result set, returning a tuple containing the rows as sequence and the EOF packet information. The count argument can be used to get a given amount of rows. If count is not specified or is `None`, all rows are retrieved.

The tuple returned by `get_rows()` consists of:

- A list of tuples containing the row data as byte objects, or an empty list when no rows are available.
- EOF packet information as a dictionary containing `status_flag` and `warning_count`.

An `InterfaceError` is raised when all rows have been retrieved.

The `get_rows()` method is used by `MySQLCursor` to fetch rows.

Returns a tuple.

22.6.7.2.21. Method `MySQLConnection.get_row()`

This method retrieves the next row of a query result set, returning a tuple.

The tuple returned by `get_row()` consists of:

- The row as a tuple containing byte objects, or `None` when no more rows are available.
- EOF packet information as a dictionary containing `status_flag` and `warning_count`, or `None` when the row returned is not the last row.

The `get_row()` method is used by `MySQLCursor` to fetch rows.

22.6.7.2.22. Method `MySQLConnection.get_server_info()`

This method returns the MySQL server information verbatim, for example `'5.5.24-log'`, or `None` when not connected.

Returns a string or `None`.

22.6.7.2.23. Method `MySQLConnection.get_server_version()`

This method returns the MySQL server version as a tuple, or `None` when not connected.

Returns a tuple or `None`.

22.6.7.2.24. Method `MySQLConnection.is_connected()`

Reports whether the connection to MySQL Server is available.

This method checks whether the connection to MySQL is available using the `ping()` method, but unlike `ping()`, `is_connected()` returns `True` when the connection is available, `False` otherwise.

Returns `True` or `False`.

22.6.7.2.25. Method `MySQLConnection.isset_client_flag(flag)`

This method returns `True` if the client flag was set, `False` otherwise.

Returns `True` or `False`.

22.6.7.2.26. Method `MySQLConnection.ping(attempts=1, delay=0)`

Check whether the connection to the MySQL server is still available.

When `reconnect` is set to `True`, one or more attempts are made to try to reconnect to the MySQL server using the `reconnect()` method. Use the `delay` argument (seconds) if you want to wait between each retry.

When the connection is not available, an `InterfaceError` is raised. Use the `is_connected()` method to check the connection without raising an error.

Raises `InterfaceError` on errors.

22.6.7.2.27. Method `MySQLConnection.reconnect(attempts=1, delay=0)`

Attempt to reconnect with the MySQL server.

The argument `attempts` specifies the number of times a reconnect is tried. The `delay` argument is the number of seconds to wait between each retry.

You might set the number of attempts higher and use a longer delay when you expect the MySQL server to be down for maintenance, or when you expect the network to be temporarily unavailable.

22.6.7.2.28. Method `MySQLConnection.rollback()`

This method sends the `ROLLBACK` command to the MySQL server, undoing all data changes from the current `transaction`. Since by default, Connector/Python does not auto commit, it is possible to cancel transactions when using transactional storage engines such as `InnoDB`.

See the `commit()` method for `committing` transactions.

```
>>> cursor.execute("INSERT INTO employees (first_name) VALUES (%s)", ('Jane'))
>>> cnx.rollback()
```

22.6.7.2.29. Method `MySQLConnection.set_charset_collation(charset=None, collation=None)`

This method sets the character set and collation to be used for the current connection. The `charset` argument can be either the name of a character set, or the numerical equivalent as defined in `constants.CharacterSet`.

When `collation` is `None`, the default will be looked up and used.

The `charset` argument then be either:

In the following example, we set the character set to `latin1` and the collation will be set to the default `latin1_swedish_ci`:

```
>>> cnx = mysql.connector.connect(user='scott')
>>> cnx.set_charset('latin1')
```

Specify a specific collation as follows:

```
>>> cnx = mysql.connector.connect(user='scott')
>>> cnx.set_charset('latin1', 'latin1_general_ci')
```

22.6.7.2.30. Method `MySQLConnection.set_client_flags(flags)`

This method sets the client flags which are used when connecting with the MySQL server and returns the new value. The `flags` argument can be either an integer or a sequence of valid client flag values (see [Section 22.6.7.5, “Class constants.ClientFlag”](#)).

If `flags` is a sequence, each item in the sequence will set the flag when the value is positive or unset it when negative. For example, to unset `LONG_FLAG` and set the `FOUND_ROWS` flags:

```
>>> from mysql.connector.constants import ClientFlag
>>> cnx.set_client_flags([ClientFlag.FOUND_ROWS, -ClientFlag.LONG_FLAG])
>>> cnx.reconnect()
```

Note that client flags are only set or used when connecting with the MySQL server. It is therefore necessary to reconnect after making changes.

Returns an integer.

22.6.7.2.31. Property `MySQLConnection.autocommit`

This property is used to toggle the auto commit feature of MySQL and retrieve the current state. When the value evaluates to True, auto commit will be turned, otherwise it is turned off.

Note that auto commit is disabled by default when connecting through Connector/Python. This can be toggled using the [connection parameter `autocommit`](#).

When the auto commit is turned off, you have to [commit](#) transactions when using transactional storage engines such as InnoDB or NDBCluster.

```
>>> cnx.autocommit
False
>>> cnx.autocommit = True
>>> cnx.autocommit
True
```

Returns True or False.

22.6.7.2.32. Property `MySQLConnection.charset_name`

This property returns which character set is used for the connection whether it is connected or not.

Returns a string.

22.6.7.2.33. Property `MySQLConnection.collation_name`

This property returns which collation is used for the connection whether it is connected or not.

Returns a string.

22.6.7.2.34. Property `MySQLConnection.connection_id`

This property returns the connection ID (thread ID or session ID) for the current connection or None when not connected.

Returns a integer or None.

22.6.7.2.35. Property `MySQLConnection.database`

This property is used to set current (active) database executing the `USE` command. The property can also be used to retrieve the current database name.

```
>>> cnx.database = 'test'
>>> cnx.database = 'mysql'
>>> cnx.database
u'mysql'
```

Returns a string.

22.6.7.2.36. Property `MySQLConnection.get_warnings`

This property is used to toggle whether warnings should be fetched automatically or not. It accepts True or False (default).

Fetching warnings automatically could be useful when debugging queries. Cursors will make warnings available through the method `MySQLCursor.fetchwarnings()`.

```
>>> cnx.get_warnings = True
>>> cursor.execute('SELECT "a"+1')
>>> cursor.fetchall()
[(1.0,)]
>>> cursor.fetchwarnings()
[(u'Warning', 1292, u'Truncated incorrect DOUBLE value: 'a')]
```

Returns True or False.

22.6.7.2.37. Property `MySQLConnection.raise_on_warnings`

This property is used to toggle whether warnings should raise exceptions or not. It accepts True or False (default).

Toggleing `raise_on_warnings` will also toggle `get_warnings` since warnings need to be fetched so they can be raised as exceptions.

Note that you might always want to check setting SQL Mode if you would like to have the MySQL server directly report warnings as errors. It is also good to use transactional engines so transactions can be rolled back when catching the exception.

Result sets needs to be fetched completely before any exception can be raised. The following example shows the execution of a query which produces a warning

```
>>> cnx.raise_on_warnings = True
>>> cursor.execute('SELECT "a"+1')
>>> cursor.fetchall()
..
mysql.connector.errors.DataError: 1292: Truncated incorrect DOUBLE value: 'a'
```

Returns True or False.

22.6.7.2.38. Property `MySQLConnection.server_host`

This read-only property returns the hostname or IP address used for connecting with the MySQL server.

Returns a string.

22.6.7.2.39. Property `MySQLConnection.server_port`

This read-only property returns the TCP/IP port used for connecting with the MySQL server.

Returns an integer.

22.6.7.2.40. Property `MySQLConnection.sql_mode`

This property is used to retrieve and set the SQL Modes for the current. The value should be list of different modes separated by comma (","), or a sequence of modes, preferably using the constants.SQLMode class.

To unset all modes, pass an empty string or an empty sequence.

```
>>> cnx.sql_mode = 'TRADITIONAL,NO_ENGINE_SUBSTITUTION'
>>> cnx.sql_mode.split(',')
[u'STRICT_TRANS_TABLES', u'STRICT_ALL_TABLES', u'NO_ZERO_IN_DATE',
u'NO_ZERO_DATE', u'ERROR_FOR_DIVISION_BY_ZERO', u'TRADITIONAL',
u'NO_AUTO_CREATE_USER', u'NO_ENGINE_SUBSTITUTION']

>>> from mysql.connector.constants import SQLMode
>>> cnx.sql_mode = [ SQLMode.NO_ZERO_DATE, SQLMode.REAL_AS_FLOAT]
>>> cnx.sql_mode

u'REAL_AS_FLOAT,NO_ZERO_DATE'
```

Returns a string.

22.6.7.2.41. Property `MySQLConnection.time_zone`

This property is used to set the time zone session variable for the current connection and retrieve it.

```
>>> cnx.time_zone = '+00:00'
>>> cur.execute('SELECT NOW()') ; cur.fetchone()
(datetime.datetime(2012, 6, 15, 11, 24, 36),)
>>> cnx.time_zone = '-09:00'
>>> cur.execute('SELECT NOW()') ; cur.fetchone()
(datetime.datetime(2012, 6, 15, 2, 24, 44),)
>>> cnx.time_zone
u'-09:00'
```

Returns a string.

22.6.7.2.42. Property `MySQLConnection.unix_socket`

This read-only property returns the UNIX socket user for connecting with the MySQL server.

Returns a string.

22.6.7.2.43. Property `MySQLConnection.user`

This read-only property returns the username used for connecting with the MySQL server.

Returns a string.

22.6.7.3. Class `cursor.MySQLCursor`

The `MySQLCursor` class is used to instantiate object which can execute operation such as SQL queries. They interact with the MySQL server using a `MySQLConnection` object.

22.6.7.3.1. Constructor `cursor.MySQLCursor`

The constructor initializes the instance with the optional `connection`, which should be an instance of `MySQLConnection`.

In most cases, the `MySQLConnection` method `cursor()` is used to instantiate a `MySQLCursor` object.

22.6.7.3.2. Method `MySQLCursor.callproc(procname, args=())`

This method calls a stored procedure with the given name. The `args` sequence of parameters must contain one entry for each argument that the routine expects. The result is returned as modified copy of the input sequence. Input parameters are left untouched, output and input/output parameters replaced with possibly new values.

Result set provided by the stored procedure are automatically fetched and stored as `MySQLBufferedCursor` instances. See `stored_results()` for more information.

The following example shows how to execute a stored procedure which takes two parameters, multiplies the values and returns the product:

```
# Definition of the multiply stored procedure:
# CREATE PROCEDURE multiply(IN pFac1 INT, IN pFac2 INT, OUT pProd INT)
# BEGIN
#   SET pProd := pFac1 * pFac2;
# END

>>> args = (5, 5, 0) # 0 is to hold value of the OUT parameter pProd
>>> cursor.callproc('multiply', args)
('5', '5', 25L)
```

22.6.7.3.3. Method `MySQLCursor.close()`

This method will close the MySQL cursor, resetting all results and removing the connection.

Use `close()` every time you are done using the cursor.

22.6.7.3.4. Method `MySQLCursor.execute(operation, params=None, multi=False)`

This method prepare the given database `operation` (query or command). The parameters found in the tuple or dictionary `params` will be bound to the variables in the operation. Variables are specified using `%s` markers or named markers `%(name)s`.

For example, insert information about a new employee and selecting again the data of this person:

```
insert = (
"INSERT INTO employees (emp_no, first_name, last_name, hire_date) "
"VALUES (%s, %s, %s, %s)"
data = (2, 'Jane', 'Doe', datetime.date(2012, 3, 23))
cursor.execute(insert, data)

select = "SELECT * FROM employees WHERE emp_no = %(emp_no)s"
cursor.execute(select, { 'emp_no': 2 })
```

Note that the data is converted from Python object to something MySQL understand. In the above example, the `datetime.date()` instance is converted to `'2012-03-23'` in the above example.

When `multi` is set to `True`, `execute()` will be able to execute multiple statements. It will return an iterator which makes it possible to go through all results for each statement. Note that using parameters is not working well in this case, and it's usually a good idea to execute each statement on its own.

In the following example we select and insert data in one operation and display the result:

```
operation = 'SELECT 1; INSERT INTO t1 VALUES (); SELECT 2'
for result in cursor.execute(operation):
    if result.with_rows:
        print("Statement '{}' has following rows:".format(
            result.statement))
        print(result.fetchall())
    else:
        print("Affected row(s) by query '{}' was {}".format(
            result.statement, result.rowcount))
```

If the connection was configured to fetch warnings, warnings generated by the operation will be available through the method `MySQLCursor.fetchwarnings()`.

Returns an iterator when `multi` is `True`.

22.6.7.3.5. Method `MySQLCursor.executemany(operation, seq_params)`

This method prepares a database operation (query or command) and then execute it against all parameter sequences or mappings found in the sequence `seq_of_params`.

The `executemany()` is simply iterating through the sequence of parameters calling the `execute()` method. Inserting data, however, is optimized by batching them using the multiple rows syntax.

In the following example we are inserting 3 records:

```
data = [
    ('Jane', date(2005, 2, 12)),
    ('Joe', date(2006, 5, 23)),
    ('John', date(2010, 10, 3)),
]
stmt = "INSERT INTO employees (first_name, hire_date) VALUES (%s, %s)"
cursor.executemany(stmt, data)
```

In the above example, the INSERT statement sent to MySQL would be as follows: `INSERT INTO employees (first_name, hire_date) VALUES ('Jane', '2005-02-12'), ('Joe', '2006-05-23'), ('John', '2010-10-03')`.

Note that it is not possible to execute multiple statements using the `executemany()` method. Doing so will raise an `InternalError` exception.

22.6.7.3.6. Method `MySQLCursor.fetchall()`

The method fetches all or remaining rows of a query result set, returning a list of tuples. An empty list is returned when no rows are (anymore) available.

The following examples shows how to retrieve the first 2 rows of a result set, and then retrieve the remaining rows:

```
>>> cursor.execute("SELECT * FROM employees ORDER BY emp_no")
>>> head_rows = cursor.fetchmany(size=2)
>>> remaining_rows = cursor.fetchall()
```

Note that you have to fetch all rows before being able to execute new queries using the same connection.

Returns a list of tuples or empty list when no rows available.

22.6.7.3.7. Method `MySQLCursor.fetchmany(size=1)`

This method fetches the next set of rows of a query results, returning a list of tuples. An empty list is returned when no more rows are available.

The number of rows returned can be specified using the `size` argument, which defaults to one. Fewer rows might be returned, when there are not more rows available than specified by the argument.

Note that you have to fetch all rows before being able to execute new queries using the same connection.

Returns a list of tuples or empty list when no rows available.

22.6.7.3.8. Method `MySQLCursor.fetchone()`

This method retrieves the next row of a query result set, returning a single sequence, or `None` when no more data is available. The returned tuple consists of data returned by the MySQL server converted to Python objects.

The `fetchone()` method is used by `fetchmany()` and `fetchall()`. It is also used when using the `MySQLCursor` instance as an iterator.

The following examples show how to iterate through the result of a query using `fetchone()`:

```
# Using a while-loop
cursor.execute("SELECT * FROM employees")
row = cursor.fetchone()
while row is not None:
    print(row)
    row = cursor.fetchone()

# Using the cursor as iterator
cursor.execute("SELECT * FROM employees")
for row in cursor:
    print(row)
```

Note that you have to fetch all rows before being able to execute new queries using the same connection.

Returns a tuple or `None`.

22.6.7.3.9. Method `MySQLCursor.fetchwarnings()`

This method returns a list of tuples containing warnings generated by previously executed statement. Use the connection's `get_warnings` property to toggle whether warnings has to be fetched.

The following example shows a `SELECT` statement which generated a warning:

```
>>> cnx.get_warnings = True
>>> cursor.execute('SELECT "a"+1')
>>> cursor.fetchall()
[(1.0,)]
>>> cursor.fetchwarnings()
[(u'Warning', 1292, u'Truncated incorrect DOUBLE value: 'a')]
```

It is also possible to raise errors when warnings are found. See the `MySQLConnection` property `raise_on_warnings`.

Returns a list of tuples.

22.6.7.3.10. Method `MySQLCursor.stored_results()`

This method returns an list iterator object which can be used to go through result sets provided by stored procedures after calling them using the `callproc()` method.

In the following example we execute a stored procedure which will provide two result sets. We use `stored_results()` to retrieve them:

```
>>> cursor.callproc('sp1')
()
>>> for result in cursor.stored_results():
...     print result.fetchall()
...
[(1,)]
[(2,)]
```

Note that the result sets stay available until you executed another operation or call another stored procedure.

Returns a listiterator.

22.6.7.3.11. Property `MySQLCursor.column_names`

This read-only property returns the column names of a result set as sequence of (unicode) strings.

The following example shows how you can create a dictionary out of a tuple containing data with keys using `column_names`:

```
cursor.execute("SELECT last_name, first_name, hire_date "
              "FROM employees WHERE emp_no = %s", (123,))
row = dict(zip(cursor.column_names, cursor.fetchone()))
print("{last_name}, {first_name}: {hire_date}".format(row))
```

Returns a tuple.

22.6.7.3.12. Property `MySQLCursor.statement`

This read-only property returns the last executed statement. In case multiple statements were executed, it will show the actual statement.

The `statement` property might be useful for debugging and showing what was send to the MySQL server.

Returns a string.

22.6.7.3.13. Property `MySQLCursor.with_rows`

This read-only property will return True when the result of the executed operation provides rows.

The `with_rows` property is useful when executing multiple statements and you need to fetch rows. In the following example we only report the affected rows by the `UPDATE` statement:

```
import mysql.connector
cnx = mysql.connector.connect(user='scott', database='test')
cursor = cnx.cursor()
operation = 'SELECT 1; UPDATE t1 SET c1 = 2; SELECT 2'
for result in cursor.execute(operation, multi=True):
    if result.with_rows:
        result.fetchall()
    else:
        print("Updated row(s): {}".format(result.rowcount))
```

22.6.7.4. Class `cursor.MySQLCursorBuffered`

This class is inheriting from `cursor.MySQLCursor` and if needed automatically retrieves rows after an operation has been executed.

`MySQLCursorBuffered` can be useful in situations where two queries, with small result sets, need to be combined or computed with each other.

You can either use the `buffered` argument when using the connection's `cursor()` method, or you can use the [buffered connection option](#) to make all created cursors by default buffering.

```
import mysql.connector

cnx = mysql.connector.connect()

# Only this particular cursor will be buffering results
cursor.cursor(buffered=True)

# All cursors by default buffering
cnx = mysql.connector.connect(buffered=True)
```

See [Section 22.6.5.1, "Tutorial: Raise employee's salary using a buffering cursor"](#) for a practical use case.

22.6.7.5. Class `constants.ClientFlag`

This class provides constants defining MySQL client flags which can be used upon connection to configure the session. The `ClientFlag` class is available when importing `mysql.connector`.

```
>>> import mysql.connector
>>> mysql.connector.constants.ClientFlag.FOUND_ROWS
2
```

See [Section 22.6.7.2.30, "Method `MySQLConnection.set_client_flags\(flags\)`"](#) and the [connection argument `client_flag`](#).

Note that the `ClientFlag` class can not be instantiated.

22.6.7.6. Class `constants.FieldType`

This class provides all supported MySQL field or data types. They can be useful when dealing with raw data or defining your own converters. The field type is stored with every cursor in the description for each column.

The following example shows how you can print the name of the data types for each of the columns in the result set.

```
from __future__ import print_function
import mysql.connector
from mysql.connector import FieldType

cnx = mysql.connector.connect(user='scott', database='test')
cursor = cnx.cursor()

cursor.execute(
    "SELECT DATE(NOW()) AS `c1`, TIME(NOW()) AS `c2`, "
    "NOW() AS `c3`, 'a string' AS `c4`, 42 AS `c5`")
rows = cursor.fetchall()

for desc in cursor.description:
    colname = desc[0]
    coltype = desc[1]
```

```
print("Column {} has type {}".format(
    colname, FieldType.get_info(coltype)))

cursor.close()
cnx.close()
```

Note that the `FieldType` class can not be instantiated.

22.6.7.7. Class `constants.SQLMode`

This class provides all known MySQL [Server SQL Modes](#). It is mostly used when setting the SQL modes at connection time using the connection's property `sql_mode`. See [Section 22.6.7.2.40](#), “[Property `MySQLConnection.sql_mode`](#)”.

Note that the `SQLMode` class can not be instantiated.

22.6.7.8. Class `constants.CharacterSet`

This class provides all known MySQL characters sets and their default collations. See [Section 22.6.7.2.29](#), “[Method `MySQLConnection.set_charset_collation\(charset=None, collation=None\)`](#)” for examples.

Note that the `CharacterSet` class can not be instantiated.

22.6.7.9. Class `constants.RefreshOption`

- `RefreshOption.GRANT`

Refresh the grant tables, like `FLUSH PRIVILEGES`.

- `RefreshOption.LOG`

Flush the logs, like `FLUSH LOGS`.

- `RefreshOption.TABLES`

Flush the table cache, like `FLUSH TABLES`.

- `RefreshOption.HOSTS`

Flush the host cache, like `FLUSH HOSTS`.

- `RefreshOption.STATUS`

Reset status variables, like `FLUSH STATUS`.

- `RefreshOption.THREADS`

Flush the thread cache.

- `RefreshOption.SLAVE`

On a slave replication server, reset the master server information and restart the slave, like `RESET SLAVE`.

- `RefreshOption.MASTER`

On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like `RESET MASTER`.

22.6.8. Connector/Python FAQ

Frequently asked questions.

22.7. libmysqld, the Embedded MySQL Server Library

The embedded MySQL server library makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and more simple management for embedded applications.

The embedded server library is based on the client/server version of MySQL, which is written in C/C++. Consequently, the embedded server also is written in C/C++. There is no embedded server available in other languages.

The API is identical for the embedded MySQL version and the client/server version. To change an old threaded application to use the embedded library, you normally only have to add calls to the following functions.

Function	When to Call
<code>mysql_library_init()</code>	Should be called before any other MySQL function is called, preferably early in the <code>main()</code> function.
<code>mysql_library_end()</code>	Should be called before your program exits.
<code>mysql_thread_init()</code>	Should be called in each thread you create that accesses MySQL.
<code>mysql_thread_end()</code>	Should be called before calling <code>pthread_exit()</code>

Then you must link your code with `libmysqld.a` instead of `libmysqlclient.a`. To ensure binary compatibility between your application and the server library, be sure to compile your application against headers for the same series of MySQL that was used to compile the server library. For example, if `libmysqld` was compiled against MySQL 4.1 headers, do not compile your application against MySQL 5.1 headers, or vice versa.

The `mysql_library_xxx()` functions are also included in `libmysqlclient.a` to enable you to change between the embedded and the client/server version by just linking your application with the right library. See [Section 22.8.3.40](#), “`mysql_library_init()`”.

One difference between the embedded server and the standalone server is that for the embedded server, authentication for connections is disabled by default. To use authentication for the embedded server, define the `HAVE_EMBEDDED_PRIVILEGE_CONTROL` [156] compiler flag when you invoke `CMake` to configure your MySQL distribution. See [Section 2.10.4](#), “MySQL Source-Configuration Options”.

22.7.1. Compiling Programs with `libmysqld`

In precompiled binary MySQL distributions that include `libmysqld`, the embedded server library, MySQL builds the library using the appropriate vendor compiler if there is one.

To get a `libmysqld` library if you build MySQL from source yourself, you should configure MySQL with the `-DWITH_EMBEDDED_SERVER=1` [155] option. See [Section 2.10.4](#), “MySQL Source-Configuration Options”.

When you link your program with `libmysqld`, you must also include the system-specific `pthread` libraries and some libraries that the MySQL server uses. You can get the full list of libraries by executing `mysql_config --libmysqld-libs`.

The correct flags for compiling and linking a threaded program must be used, even if you do not directly call any thread functions in your code.

To compile a C program to include the necessary files to embed the MySQL server library into an executable version of a program, the compiler will need to know where to find various files and need instructions on how to compile the program. The following example shows how a program could be compiled from the command line, assuming that you are using `gcc`, use the GNU C compiler:

```
gcc mysql_test.c -o mysql_test -lz \  
`/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Immediately following the `gcc` command is the name of the C program source file. After it, the `-o` option is given to indicate that the file name that follows is the name that the compiler is to give to the output file, the compiled program. The next line of code tells the compiler to obtain the location of the include files and libraries and other settings for the system on which it is compiled. Because of a problem with `mysql_config`, the option `-lz` (for compression) is added here. The `mysql_config` command is contained in backticks, not single quotation marks.

On some non-`gcc` platforms, the embedded library depends on C++ runtime libraries and linking against the embedded library might result in missing-symbol errors. To solve this, link using a C++ compiler or explicitly list the required libraries on the link command line.

22.7.2. Restrictions When Using the Embedded MySQL Server

The embedded server has the following limitations:

- No user-defined functions (UDFs).
- No stack trace on core dump.
- You cannot set this up as a master or a slave (no replication).
- Very large result sets may be unusable on low memory systems.
- You cannot connect to an embedded server from an outside process with sockets or TCP/IP. However, you can connect to an intermediate application, which in turn can connect to an embedded server on the behalf of a remote client or outside process.
- `InnoDB` is not reentrant in the embedded server and cannot be used for multiple connections, either successively or simultaneously.
- The Event Scheduler is not available. Because of this, the `event_scheduler` [498] system variable is disabled.

Some of these limitations can be changed by editing the `mysql_embed.h` include file and recompiling MySQL.

22.7.3. Options with the Embedded Server

Any options that may be given with the `mysqld` server daemon, may be used with an embedded server library. Server options may be given in an array as an argument to the `mysql_library_init()`, which initializes the server. They also may be given in an option file like `my.cnf`. To specify an option file for a C program, use the `--defaults-file` [248] option as one of the elements of the second argument of the `mysql_library_init()` function. See Section 22.8.3.40, “`mysql_library_init()`”, for more information on the `mysql_library_init()` function.

Using option files can make it easier to switch between a client/server application and one where MySQL is embedded. Put common options under the `[server]` group. These are read by both MySQL versions. Client/server-specific options should go under the `[mysqld]` section. Put options specific to the embedded MySQL server library in the `[embedded]` section. Options specific to applications go under section labeled `[ApplicationName_SERVER]`. See [Section 4.2.3.3, “Using Option Files”](#).

22.7.4. Embedded Server Examples

These two example programs should work without any changes on a Linux or FreeBSD system. For other operating systems, minor changes are needed, mostly with file paths. These examples are designed to give enough details for you to understand the problem, without the clutter that is a necessary part of a real application. The first example is very straightforward. The second example is a little more advanced with some error checking. The first is followed by a command-line entry for compiling the program. The second is followed by a GNUmake file that may be used for compiling instead.

Example 1

`test1_libmysqld.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = \
    { "mysql_test", "--defaults-file=my.cnf", NULL };
int num_elements = (sizeof(server_options) / sizeof(char *)) - 1;

static char *server_groups[] = { "libmysqld_server",
    "libmysqld_client", NULL };

int main(void)
{
    mysql_library_init(num_elements, server_options, server_groups);
    mysql = mysql_init(NULL);
    mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
    mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);

    mysql_real_connect(mysql, NULL, NULL, NULL, "database1", 0, NULL, 0);

    mysql_query(mysql, "SELECT column1, column2 FROM table1");

    results = mysql_store_result(mysql);

    while((record = mysql_fetch_row(results))) {
        printf("%s - %s\n", record[0], record[1]);
    }

    mysql_free_result(results);
    mysql_close(mysql);
    mysql_library_end();

    return 0;
}
```

Here is the command line for compiling the above program:

```
gcc test1_libmysqld.c -o test1_libmysqld -lz \
    `/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Example 2

To try the example, create an `test2_libmysqld` directory at the same level as the MySQL source directory. Save the `test2_libmysqld.c` source and the `GNUmakefile` in the directory, and run GNU `make` from inside the `test2_libmysqld` directory.

`test2_libmysqld.c`

```

/*
 * A simple example client, using the embedded MySQL server library
 */

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
    "test2_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
    MYSQL *one, *two;

    /* mysql_library_init() must be called before any other mysql
     * functions.
     *
     * You can use mysql_library_init(0, NULL, NULL), and it
     * initializes the server using groups = {
     *   "server", "embedded", NULL
     * }.
     *
     * In your $HOME/.my.cnf file, you probably want to put:
     [test2_libmysqld_SERVER]
     language = /path/to/source/of/mysql/sql/share/english
     *
     * You could, of course, modify argc and argv before passing
     * them to this function. Or you could create new ones in any
     * way you like. But all of the arguments in argv (except for
     * argv[0], which is the program name) should be valid options
     * for the MySQL server.
     *
     * If you link this client against the normal mysqlclient
     * library, this function is just a stub that does nothing.
     */
    mysql_library_init(argc, argv, (char **)server_groups);

    one = db_connect("test");
    two = db_connect(NULL);

    db_do_query(one, "SHOW TABLE STATUS");
    db_do_query(two, "SHOW DATABASES");

    mysql_close(two);
    mysql_close(one);

    /* This must be called after all other mysql functions */
    mysql_library_end();

```

```

    exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);
    (void)putc('\n', stderr);
    if (db)
        db_disconnect(db);
    exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
    MYSQL *db = mysql_init(NULL);
    if (!db)
        die(db, "mysql_init failed: no memory");
    /*
     * Notice that the client and server use separate group names.
     * This is critical, because the server does not accept the
     * client's options, and vice versa.
     */
    mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_libmysqld_CLIENT");
    if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
        die(db, "mysql_real_connect failed: %s", mysql_error(db));

    return db;
}

void
db_disconnect(MYSQL *db)
{
    mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
    if (mysql_query(db, query) != 0)
        goto err;

    if (mysql_field_count(db) > 0)
    {
        MYSQL_RES *res;
        MYSQL_ROW row, end_row;
        int num_fields;

        if (!(res = mysql_store_result(db)))
            goto err;
        num_fields = mysql_num_fields(res);
        while ((row = mysql_fetch_row(res)))
        {
            (void)fputs(">> ", stdout);
            for (end_row = row + num_fields; row < end_row; ++row)
                (void)printf("%s\t", row ? (char*)*row : "NULL");
            (void)putc('\n', stdout);
        }
        (void)putc('\n', stdout);
        mysql_free_result(res);
    }
    else
        (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));
}

```

```

    return;

err:
    die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}

```

GNUMakefile

```

# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc     := $(HOME)/mysql-5.5/include
#lib     := $(HOME)/mysql-5.5/libmysqld

CC       := gcc
CPPFLAGS := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS  := -g -W -Wall
LDFLAGS := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS  = -L$(lib) -lmysqld -lz -lm -ldl -lcrypt

ifneq (, $(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
    rm -f $(targets) $(objects) *.core

```

22.8. MySQL C API

The C API code is distributed with MySQL. It is included in the `mysqlclient` library and enables C programs to access a database.

Many of the clients in the MySQL source distribution are written in C. If you are looking for examples that demonstrate how to use the C API, take a look at these clients. You can find these in the `client` directory in the MySQL source distribution.

Most of the other client APIs (all except Connector/J and Connector/Net) use the `mysqlclient` library to communicate with the MySQL server. This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs, because they are referenced from the library. See [Chapter 4, MySQL Programs](#), for a list of these variables.

The client has a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (the maximum is 16MB). Because buffer sizes are increased only as demand warrants, simply increasing the default maximum limit does not in itself cause more resources to be used. This size check is mostly a check for erroneous statements and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each thread's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16MB of data, you must have a communication buffer limit of at least 16MB (in both server and client). The client's default maximum is 16MB, but the default maximum in the server is 1MB. You can increase this by changing the value of the `max_allowed_packet` [519] parameter when the server is started. See [Section 8.11.2, "Tuning Server Parameters"](#).

The MySQL server shrinks each communication buffer to `net_buffer_length` [531] bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

For programming with threads, see [Section 22.8.17.2, "How to Write a Threaded Client"](#). For creating a standalone application which includes the "server" and "client" in the same program (and does not communicate with an external MySQL server), see [Section 22.7, "libmysql, the Embedded MySQL Server Library"](#).

**Note**

If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example from `libmysqlclient.so.15` to `libmysqlclient.so.16`).

22.8.1. C API Data Structures

This section describes C API data structures other than those used for prepared statements. For information about the latter, see [Section 22.8.5, "C API Prepared Statement Data Structures"](#).

- `MYSQL`

This structure represents a handle to one database connection. It is used for almost all MySQL functions. Do not try to make a copy of a `MYSQL` structure. There is no guarantee that such a copy will be usable.

- `MYSQL_RES`

This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

- `MYSQL_ROW`

This is a type-safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

- `MYSQL_FIELD`

This structure contains metadata: information about a field, such as the field's name, type, and size. Its members are described in more detail later in this section. You may obtain the `MYSQL_FIELD` structures

for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

- `MYSQL_FIELD_OFFSET`

This is a type-safe representation of an offset into a MySQL field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row, beginning at zero.

- `my_ulonglong`

The type used for the number of rows and for `mysql_affected_rows()`, `mysql_num_rows()`, and `mysql_insert_id()`. This type provides a range of 0 to 1.84e19.

On some systems, attempting to print a value of type `my_ulonglong` does not work. To print such a value, convert it to `unsigned long` and use a `%lu` print format. Example:

```
printf ("Number of rows: %lu\n",
        (unsigned long) mysql_num_rows(result));
```

- `my_bool`

A boolean type, for values that are true (nonzero) or false (zero).

The `MYSQL_FIELD` structure contains the members described in the following list. The definitions apply primarily for columns of result sets such as those produced by `SELECT` statements. As of MySQL 5.5.3, `MYSQL_FIELD` structures are also used to provide metadata for `OUT` and `INOUT` parameters returned from stored procedures executed using prepared `CALL` statements. For such parameters, some of the structure members have a meaning different from the meaning for column values.

- `char * name`

The name of the field, as a null-terminated string. If the field was given an alias with an `AS` clause, the value of `name` is the alias. For a procedure parameter, the parameter name.

- `char * org_name`

The name of the field, as a null-terminated string. Aliases are ignored. For a procedure parameter, the parameter name.

- `char * table`

The name of the table containing this field, if it isn't a calculated field. For calculated fields, the `table` value is an empty string. If the column is selected from a view, `table` names the view. If the table or view was given an alias with an `AS` clause, the value of `table` is the alias. For a `UNION`, the value is the empty string. For a procedure parameter, the procedure name.

- `char * org_table`

The name of the table, as a null-terminated string. Aliases are ignored. If the column is selected from a view, `org_table` names the underlying table. For a `UNION`, the value is the empty string. For a procedure parameter, the procedure name.

- `char * db`

The name of the database that the field comes from, as a null-terminated string. If the field is a calculated field, `db` is an empty string. For a `UNION`, the value is the empty string. For a procedure parameter, the name of the database containing the procedure.

- `char * catalog`

The catalog name. This value is always "def".

- `char * def`

The default value of this field, as a null-terminated string. This is set only if you use `mysql_list_fields()`.

- `unsigned long length`

The width of the field. This corresponds to the display length, in bytes.

The server determines the `length` value before it generates the result set, so this is the minimum length required for a data type capable of holding the largest possible value from the result column, without knowing in advance the actual values that will be produced by the query for the result set.

- `unsigned long max_length`

The maximum width of the field for the result set (the length in bytes of the longest field value for the rows actually in the result set). If you use `mysql_store_result()` or `mysql_list_fields()`, this contains the maximum length for the field. If you use `mysql_use_result()`, the value of this variable is zero.

The value of `max_length` is the length of the string representation of the values in the result set. For example, if you retrieve a `FLOAT` column and the "widest" value is `-12.345`, `max_length` is 7 (the length of `'-12.345'`).

If you are using prepared statements, `max_length` is not set by default because for the binary protocol the lengths of the values depend on the types of the values in the result set. (See [Section 22.8.5, "C API Prepared Statement Data Structures"](#).) If you want the `max_length` values anyway, enable the `STMT_ATTR_UPDATE_MAX_LENGTH` option with `mysql_stmt_attr_set()` and the lengths will be set when you call `mysql_stmt_store_result()`. (See [Section 22.8.7.3, "mysql_stmt_attr_set\(\)"](#), and [Section 22.8.7.28, "mysql_stmt_store_result\(\)"](#).)

- `unsigned int name_length`

The length of `name`.

- `unsigned int org_name_length`

The length of `org_name`.

- `unsigned int table_length`

The length of `table`.

- `unsigned int org_table_length`

The length of `org_table`.

- `unsigned int db_length`

The length of `db`.

- `unsigned int catalog_length`

The length of `catalog`.

- `unsigned int def_length`

The length of `def`.

- `unsigned int flags`

Bit-flags that describe the field. The `flags` value may have zero or more of the bits set that are shown in the following table.

Flag Value	Flag Description
<code>NOT_NULL_FLAG</code>	Field can't be <code>NULL</code>
<code>PRI_KEY_FLAG</code>	Field is part of a primary key
<code>UNIQUE_KEY_FLAG</code>	Field is part of a unique key
<code>MULTIPLE_KEY_FLAG</code>	Field is part of a nonunique key
<code>UNSIGNED_FLAG</code>	Field has the <code>UNSIGNED</code> attribute
<code>ZEROFILL_FLAG</code>	Field has the <code>ZEROFILL</code> attribute
<code>BINARY_FLAG</code>	Field has the <code>BINARY</code> attribute
<code>AUTO_INCREMENT_FLAG</code>	Field has the <code>AUTO_INCREMENT</code> attribute
<code>ENUM_FLAG</code>	Field is an <code>ENUM</code> (deprecated)
<code>SET_FLAG</code>	Field is a <code>SET</code> (deprecated)
<code>BLOB_FLAG</code>	Field is a <code>BLOB</code> or <code>TEXT</code> (deprecated)
<code>TIMESTAMP_FLAG</code>	Field is a <code>TIMESTAMP</code> (deprecated)
<code>NUM_FLAG</code>	Field is numeric; see additional notes following table
<code>NO_DEFAULT_VALUE_FLAG</code>	Field has no default value; see additional notes following table

Some of these flags indicate data type information and are superseded by or used in conjunction with the `MYSQL_TYPE_xxx` value in the `field->type` member described later:

- To check for `BLOB` or `TIMESTAMP` values, check whether `type` is `MYSQL_TYPE_BLOB` or `MYSQL_TYPE_TIMESTAMP`. (The `BLOB_FLAG` and `TIMESTAMP_FLAG` flags are unneeded.)
- `ENUM` and `SET` values are returned as strings. For these, check that the `type` value is `MYSQL_TYPE_STRING` and that the `ENUM_FLAG` or `SET_FLAG` flag is set in the `flags` value.

`NUM_FLAG` indicates that a column is numeric. This includes columns with a type of `MYSQL_TYPE_DECIMAL`, `MYSQL_TYPE_NEWDECIMAL`, `MYSQL_TYPE_TINY`, `MYSQL_TYPE_SHORT`, `MYSQL_TYPE_LONG`, `MYSQL_TYPE_FLOAT`, `MYSQL_TYPE_DOUBLE`, `MYSQL_TYPE_NULL`, `MYSQL_TYPE_LONGLONG`, `MYSQL_TYPE_INT24`, and `MYSQL_TYPE_YEAR`.

`NO_DEFAULT_VALUE_FLAG` indicates that a column has no `DEFAULT` clause in its definition. This does not apply to `NULL` columns (because such columns have a default of `NULL`), or to `AUTO_INCREMENT` columns (which have an implied default value).

The following example illustrates a typical use of the `flags` value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

You may use the convenience macros shown in the following table to determine the boolean status of the `flags` value.

Flag Status	Description
<code>IS_NOT_NULL(flags)</code>	True if this field is defined as <code>NOT NULL</code>
<code>IS_PRI_KEY(flags)</code>	True if this field is a primary key
<code>IS_BLOB(flags)</code>	True if this field is a <code>BLOB</code> or <code>TEXT</code> (deprecated; test <code>field->type</code> instead)

- `unsigned int decimals`

The number of decimals for numeric fields.

- `unsigned int charsetnr`

An ID number that indicates the character set/collation pair for the field.

To distinguish between binary and nonbinary data for string data types, check whether the `charsetnr` value is 63. If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

`charsetnr` values are the same as those displayed in the `Id` column of the `SHOW COLLATION` statement or the `ID` column of the `INFORMATION_SCHEMA.COLLATIONS` table. You can use those information sources to see which character set and collation specific `charsetnr` values indicate:

```
mysql> SHOW COLLATION WHERE Id = 63;
+-----+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| binary    | binary  | 63 | Yes     | Yes     | 1       |
+-----+-----+-----+-----+-----+-----+

mysql> SELECT COLLATION_NAME, CHARACTER_SET_NAME
-> FROM INFORMATION_SCHEMA.COLLATIONS WHERE ID = 33;
+-----+-----+
| COLLATION_NAME | CHARACTER_SET_NAME |
+-----+-----+
| utf8_general_ci | utf8                |
+-----+-----+
```

- `enum enum_field_types type`

The type of the field. The `type` value may be one of the `MYSQL_TYPE_` symbols shown in the following table.

Type Value	Type Description
<code>MYSQL_TYPE_TINY</code>	<code>TINYINT</code> field
<code>MYSQL_TYPE_SHORT</code>	<code>SMALLINT</code> field
<code>MYSQL_TYPE_LONG</code>	<code>INTEGER</code> field
<code>MYSQL_TYPE_INT24</code>	<code>MEDIUMINT</code> field
<code>MYSQL_TYPE_LONGLONG</code>	<code>BIGINT</code> field
<code>MYSQL_TYPE_DECIMAL</code>	<code>DECIMAL</code> or <code>NUMERIC</code> field
<code>MYSQL_TYPE_NEWDECIMAL</code>	Precision math <code>DECIMAL</code> or <code>NUMERIC</code>
<code>MYSQL_TYPE_FLOAT</code>	<code>FLOAT</code> field

Type Value	Type Description
<code>MYSQL_TYPE_DOUBLE</code>	<code>DOUBLE</code> or <code>REAL</code> field
<code>MYSQL_TYPE_BIT</code>	<code>BIT</code> field
<code>MYSQL_TYPE_TIMESTAMP</code>	<code>TIMESTAMP</code> field
<code>MYSQL_TYPE_DATE</code>	<code>DATE</code> field
<code>MYSQL_TYPE_TIME</code>	<code>TIME</code> field
<code>MYSQL_TYPE_DATETIME</code>	<code>DATETIME</code> field
<code>MYSQL_TYPE_YEAR</code>	<code>YEAR</code> field
<code>MYSQL_TYPE_STRING</code>	<code>CHAR</code> or <code>BINARY</code> field
<code>MYSQL_TYPE_VAR_STRING</code>	<code>VARCHAR</code> or <code>VARBINARY</code> field
<code>MYSQL_TYPE_BLOB</code>	<code>BLOB</code> or <code>TEXT</code> field (use <code>max_length</code> to determine the maximum length)
<code>MYSQL_TYPE_SET</code>	<code>SET</code> field
<code>MYSQL_TYPE_ENUM</code>	<code>ENUM</code> field
<code>MYSQL_TYPE_GEOMETRY</code>	Spatial field
<code>MYSQL_TYPE_NULL</code>	<code>NULL</code> -type field

You can use the `IS_NUM()` macro to test whether a field has a numeric type. Pass the `type` value to `IS_NUM()` and it evaluates to `TRUE` if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

`ENUM` and `SET` values are returned as strings. For these, check that the `type` value is `MYSQL_TYPE_STRING` and that the `ENUM_FLAG` or `SET_FLAG` flag is set in the `flags` value.

22.8.2. C API Function Overview

The functions available in the C API are summarized here and described in greater detail in a later section. See [Section 22.8.3, “C API Function Descriptions”](#).

Table 22.31. C API Function Names and Descriptions

Function	Description
<code>my_init()</code>	Initialize global variables, and thread handler in thread-safe programs
<code>mysql_affected_rows()</code>	Returns the number of rows changed/deleted/inserted by the last <code>UPDATE</code> , <code>DELETE</code> , or <code>INSERT</code> query
<code>mysql_autocommit()</code>	Toggles autocommit mode on/off
<code>mysql_change_user()</code>	Changes user and database on an open connection
<code>mysql_character_set_name()</code>	Return default character set name for current connection
<code>mysql_client_find_plugin()</code>	Return pointer to plugin
<code>mysql_client_register_plugin()</code>	Register a plugin
<code>mysql_close()</code>	Closes a server connection
<code>mysql_commit()</code>	Commits the transaction
<code>mysql_connect()</code>	Connects to a MySQL server (this function is deprecated; use <code>mysql_real_connect()</code> instead)

Function	Description
<code>mysql_create_db()</code>	Creates a database (this function is deprecated; use the SQL statement <code>CREATE DATABASE</code> instead)
<code>mysql_data_seek()</code>	Seeks to an arbitrary row number in a query result set
<code>mysql_debug()</code>	Does a <code>DEBUG_PUSH</code> with the given string
<code>mysql_drop_db()</code>	Drops a database (this function is deprecated; use the SQL statement <code>DROP DATABASE</code> instead)
<code>mysql_dump_debug_info()</code>	Makes the server write debug information to the log
<code>mysql_eof()</code>	Determines whether the last row of a result set has been read (this function is deprecated; <code>mysql_errno()</code> or <code>mysql_error()</code> may be used instead)
<code>mysql_errno()</code>	Returns the error number for the most recently invoked MySQL function
<code>mysql_error()</code>	Returns the error message for the most recently invoked MySQL function
<code>mysql_escape_string()</code>	Escapes special characters in a string for use in an SQL statement
<code>mysql_fetch_field()</code>	Returns the type of the next table field
<code>mysql_fetch_field_direct()</code>	Returns the type of a table field, given a field number
<code>mysql_fetch_fields()</code>	Returns an array of all field structures
<code>mysql_fetch_lengths()</code>	Returns the lengths of all columns in the current row
<code>mysql_fetch_row()</code>	Fetches the next row from the result set
<code>mysql_field_count()</code>	Returns the number of result columns for the most recent statement
<code>mysql_field_seek()</code>	Puts the column cursor on a specified column
<code>mysql_field_tell()</code>	Returns the position of the field cursor used for the last <code>mysql_fetch_field()</code>
<code>mysql_free_result()</code>	Frees memory used by a result set
<code>mysql_get_character_set_info()</code>	Return information about default character set
<code>mysql_get_client_info()</code>	Returns client version information as a string
<code>mysql_get_client_version()</code>	Returns client version information as an integer
<code>mysql_get_host_info()</code>	Returns a string describing the connection
<code>mysql_get_proto_info()</code>	Returns the protocol version used by the connection
<code>mysql_get_server_info()</code>	Returns the server version number
<code>mysql_get_server_version()</code>	Returns version number of server as an integer
<code>mysql_get_ssl_cipher()</code>	Return current SSL cipher
<code>mysql_hex_string()</code>	Encode string in hexadecimal format
<code>mysql_info()</code>	Returns information about the most recently executed query
<code>mysql_init()</code>	Gets or initializes a <code>MYSQL</code> structure
<code>mysql_insert_id()</code>	Returns the ID generated for an <code>AUTO_INCREMENT</code> column by the previous query
<code>mysql_kill()</code>	Kills a given thread
<code>mysql_library_end()</code>	Finalize the MySQL C API library
<code>mysql_library_init()</code>	Initialize the MySQL C API library

Function	Description
<code>mysql_list_dbs()</code>	Returns database names matching a simple regular expression
<code>mysql_list_fields()</code>	Returns field names matching a simple regular expression
<code>mysql_list_processes()</code>	Returns a list of the current server threads
<code>mysql_list_tables()</code>	Returns table names matching a simple regular expression
<code>mysql_load_plugin()</code>	Load a plugin
<code>mysql_load_plugin_v()</code>	Load a plugin
<code>mysql_more_results()</code>	Checks whether any more results exist
<code>mysql_next_result()</code>	Returns/initiates the next result in multiple-result executions
<code>mysql_num_fields()</code>	Returns the number of columns in a result set
<code>mysql_num_rows()</code>	Returns the number of rows in a result set
<code>mysql_options()</code>	Sets connect options for <code>mysql_real_connect()</code>
<code>mysql_ping()</code>	Checks whether the connection to the server is working, reconnecting as necessary
<code>mysql_plugin_options()</code>	Set a plugin option
<code>mysql_query()</code>	Executes an SQL query specified as a null-terminated string
<code>mysql_real_connect()</code>	Connects to a MySQL server
<code>mysql_real_escape_string()</code>	Escapes special characters in a string for use in an SQL statement, taking into account the current character set of the connection
<code>mysql_real_query()</code>	Executes an SQL query specified as a counted string
<code>mysql_refresh()</code>	Flush or reset tables and caches
<code>mysql_reload()</code>	Tells the server to reload the grant tables
<code>mysql_rollback()</code>	Rolls back the transaction
<code>mysql_row_seek()</code>	Seeks to a row offset in a result set, using value returned from <code>mysql_row_tell()</code>
<code>mysql_row_tell()</code>	Returns the row cursor position
<code>mysql_select_db()</code>	Selects a database
<code>mysql_server_end()</code>	Finalize the MySQL C API library
<code>mysql_server_init()</code>	Initialize the MySQL C API library
<code>mysql_set_character_set()</code>	Set default character set for current connection
<code>mysql_set_local_infile_defaults()</code>	Set the <code>LOAD DATA LOCAL INFILE</code> handler callbacks to their default values
<code>mysql_set_local_infile_handler()</code>	Install application-specific <code>LOAD DATA LOCAL INFILE</code> handler callbacks
<code>mysql_set_server_option()</code>	Sets an option for the connection (like <code>multi-statements</code>)
<code>mysql_sqlstate()</code>	Returns the SQLSTATE error code for the last error
<code>mysql_shutdown()</code>	Shuts down the database server
<code>mysql_ssl_set()</code>	Prepare to establish SSL connection to server
<code>mysql_stat()</code>	Returns the server status as a string
<code>mysql_store_result()</code>	Retrieves a complete result set to the client

Function	Description
<code>mysql_thread_end()</code>	Finalize thread handler
<code>mysql_thread_id()</code>	Returns the current thread ID
<code>mysql_thread_init()</code>	Initialize thread handler
<code>mysql_thread_safe()</code>	Returns 1 if the clients are compiled as thread-safe
<code>mysql_use_result()</code>	Initiates a row-by-row result set retrieval
<code>mysql_warning_count()</code>	Returns the warning count for the previous SQL statement

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL library by calling `mysql_library_init()`. This function exists in both the `mysqlclient` C client library and the `mysqld` embedded server library, so it is used whether you build a regular client program by linking with the `-libmysqlclient` flag, or an embedded server application by linking with the `-libmysqld` flag.
2. Initialize a connection handler by calling `mysql_init()` and connect to the server by calling `mysql_real_connect()`.
3. Issue SQL statements and process their results. (The following discussion provides more information about how to do this.)
4. Close the connection to the MySQL server by calling `mysql_close()`.
5. End use of the MySQL library by calling `mysql_library_end()`.

The purpose of calling `mysql_library_init()` and `mysql_library_end()` is to provide proper initialization and finalization of the MySQL library. For applications that are linked with the client library, they provide improved memory management. If you don't call `mysql_library_end()`, a block of memory remains allocated. (This does not increase the amount of memory used by the application, but some memory leak detectors will complain about it.) For applications that are linked with the embedded server, these calls start and stop the server.

In a nonmulti-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. This should be done prior to any other client library call.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the host name, user name, and password). Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. You can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior. When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL statements to the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null-terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-`SELECT` query (for example, `INSERT`, `UPDATE`, `DELETE`), you can find out how many rows were changed (affected) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`-like in that they return rows. These include `SHOW`, `DESCRIBE`, and `EXPLAIN`. Treat these statements the same way as `SELECT` statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have previously been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about the size of the data in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Choose the approach that is most appropriate for each client application. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that because the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set because it maintains only one row at a time (and because there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you don't have random access to rows within the result set (you can only access rows sequentially), and you don't know how many rows are in the result set until you have retrieved them all. Furthermore, you *must* retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to statements (retrieving rows only as necessary) without knowing whether the statement is a `SELECT`. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the statement was a `SELECT` and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether a result was actually to be expected. If `mysql_field_count()` returns zero, the statement returned no data (indicating that it was an `INSERT`, `UPDATE`, `DELETE`, and so forth), and was not expected to return rows. If `mysql_field_count()` is nonzero, the statement should have returned rows, but didn't. This indicates that the statement was a `SELECT` that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` enable you to obtain information about the fields that make up the result set (the number of fields, their names and types, and so forth). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, MySQL provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the

most recently invoked function that can succeed or fail, enabling you to determine when an error occurred and what it was.

22.8.3. C API Function Descriptions

In the descriptions here, a parameter or return value of `NULL` means `NULL` in the sense of the C programming language, not a MySQL `NULL` value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-`NULL` value to indicate success or a `NULL` value to indicate an error, and functions returning an integer return zero to indicate success or nonzero to indicate an error. Note that “nonzero” means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                /* correct */
    ... error ...

if (result < 0)            /* incorrect */
    ... error ...

if (result == -1)         /* incorrect */
    ... error ...
```

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

22.8.3.1. `mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Description

`mysql_affected_rows()` may be called immediately after executing a statement with `mysql_query()` or `mysql_real_query()`. It returns the number of rows changed, deleted, or inserted by the last statement if it was an `UPDATE`, `DELETE`, or `INSERT`. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

For `UPDATE` statements, the affected-rows value by default is the number of rows actually changed. If you specify the `CLIENT_FOUND_ROWS` flag to `mysql_real_connect()` when connecting to `mysqld`, the affected-rows value is the number of rows “found”; that is, matched by the `WHERE` clause.

For `REPLACE` statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For `INSERT ... ON DUPLICATE KEY UPDATE` statements, the affected-rows value is 1 if the row is inserted as a new row and 2 if an existing row is updated.

Following a `CALL` statement for a stored procedure, `mysql_affected_rows()` returns the value that it would return for the last statement executed within the procedure, or 0 if that statement would return -1. Within the procedure, you can use `ROW_COUNT()` [1196] at the SQL level to obtain the affected-rows value for individual statements.

As of MySQL 5.5.5, `mysql_affected_rows()` returns a meaningful value for a wider range of statements. For details, see the description for `ROW_COUNT()` [1196] in Section 12.14, “Information Functions”.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an `UPDATE` statement, no rows matched the `WHERE` clause in the query or that no query has yet been executed. -1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`.

Because `mysql_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

Errors

None.

Example

```
char *stmt = "UPDATE products SET cost=cost*1.25
             WHERE group=10";
mysql_query(&mysql,stmt);
printf("%ld products updated",
       (long) mysql_affected_rows(&mysql));
```

22.8.3.2. `mysql_autocommit()`

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

Description

Sets autocommit mode on if `mode` is 1, off if `mode` is 0.

Return Values

Zero if successful. Nonzero if an error occurred.

Errors

None.

22.8.3.3. `mysql_change_user()`

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password,
                           const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_change_user()` fails if the connected user cannot be authenticated or doesn't have permission to use the database. In this case, the user and database are not changed.

The `db` parameter may be set to `NULL` if you don't want to have a default database.

This command resets the state as if one had done a new connect. (See [Section 22.8.12, "Controlling Automatic Reconnection Behavior"](#).) It always performs a `ROLLBACK` of any active transactions, closes and drops all temporary tables, and unlocks all locked tables. Session system variables are reset to the values

of the corresponding global system variables. Prepared statements are released and [HANDLER](#) variables are closed. Locks acquired with [GET_LOCK\(\)](#) [1199] are released. These effects occur even if the user didn't change.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

The same that you can get from [mysql_real_connect\(\)](#).

- [CR_COMMANDS_OUT_OF_SYNC](#) [4030]

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR](#) [4030]

The MySQL server has gone away.

- [CR_SERVER_LOST](#) [4030]

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR](#) [4029]

An unknown error occurred.

- [ER_UNKNOWN_COM_ERROR](#) [3982]

The MySQL server doesn't implement this command (probably an old server).

- [ER_ACCESS_DENIED_ERROR](#) [3982]

The user or password was wrong.

- [ER_BAD_DB_ERROR](#) [3982]

The database didn't exist.

- [ER_DBACCESS_DENIED_ERROR](#) [3982]

The user did not have access rights to the database.

- [ER_WRONG_DB_NAME](#) [3986]

The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
    fprintf(stderr, "Failed to change user. Error: %s\n",
            mysql_error(&mysql));
}
```

22.8.3.4. [mysql_character_set_name\(\)](#)

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Description

Returns the default character set name for the current connection.

Return Values

The default character set name

Errors

None.

22.8.3.5. `mysql_close()`

```
void mysql_close(MYSQL *mysql)
```

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_connect()`.

Return Values

None.

Errors

None.

22.8.3.6. `mysql_commit()`

```
my_bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

The action of this function is subject to the value of the `completion_type` [489] system variable. In particular, if the value of `completion_type` [489] is `RELEASE` (or 2), the server performs a release after terminating a transaction and closes the client connection. Call `mysql_close()` from the client program to close the connection from the client side.

Return Values

Zero if successful. Nonzero if an error occurred.

Errors

None.

22.8.3.7. `mysql_connect()`

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd)
```

Description

This function is deprecated. Use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()` with the difference that the connection parameter may be `NULL`. In this case, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you can't retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

Return Values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

22.8.3.8. `mysql_create_db()`

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Description

Creates the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `CREATE DATABASE` statement instead.

Return Values

Zero if the database was created successfully. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
    fprintf(stderr, "Failed to create new database. Error: %s\n",
            mysql_error(&mysql));
}
```

```
}
```

22.8.3.9. `mysql_data_seek()`

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a query result set. The `offset` value is a row number. Specify a value in the range from 0 to `mysql_num_rows(result)-1`.

This function requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

None.

Errors

None.

22.8.3.10. `mysql_debug()`

```
void mysql_debug(const char *debug)
```

Description

Does a `DEBUG_PUSH` with the given string. `mysql_debug()` uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See [MySQL Internals: Porting to Other Systems](#).

Return Values

None.

Errors

None.

Example

The call shown here causes the client library to generate a trace file in `/tmp/client.trace` on the client machine:

```
mysql_debug("d:t:0,/tmp/client.trace");
```

22.8.3.11. `mysql_drop_db()`

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `DROP DATABASE` statement instead.

Return Values

Zero if the database was dropped successfully. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

Example

```
if(mysql_drop_db(&mysql, "my_database"))  
    fprintf(stderr, "Failed to drop the database: Error: %s\n",  
            mysql_error(&mysql));
```

22.8.3.12. `mysql_dump_debug_info()`

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write some debug information to the log. For this to work, the connected user must have the [SUPER \[697\]](#) privilege.

Return Values

Zero if the command was successful. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

22.8.3.13. `mysql_eof()`

```
my_bool mysql_eof(MYSQL_RES *result)
```

Description

This function is deprecated. `mysql_errno()` or `mysql_error()` may be used instead.

`mysql_eof()` determines whether the last row of a result set has been read.

If you acquire a result set from a successful call to `mysql_store_result()`, the client receives the entire set in one operation. In this case, a `NULL` return from `mysql_fetch_row()` always means the end of the result set has been reached and it is unnecessary to call `mysql_eof()`. When used with `mysql_store_result()`, `mysql_eof()` always returns true.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL` return value from `mysql_fetch_row()` does not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a nonzero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard MySQL error functions `mysql_errno()` and `mysql_error()`. Because those error functions provide the same information, their use is preferred over `mysql_eof()`, which is deprecated. (In fact, they provide more information, because `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return Values

Zero if no error occurred. Nonzero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql, "SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result)) // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard MySQL error functions:

```
mysql_query(&mysql, "SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql)) // mysql_fetch_row() failed due to an error
{
```

```
fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

22.8.3.14. `mysql_errno()`

```
unsigned int mysql_errno(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at [Appendix C, Errors, Error Codes, and Common Problems](#).

Note that some functions like `mysql_fetch_row()` don't set `mysql_errno()` if they succeed.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_errno()` if they succeed.

MySQL-specific error numbers returned by `mysql_errno()` differ from SQLSTATE values returned by `mysql_sqlstate()`. For example, the `mysql` client program displays errors using the following format, where 1146 is the `mysql_errno()` value and '42S02' is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Return Values

An error code value for the last `mysql_xxx()` call, if it failed. zero means no error occurred.

Errors

None.

22.8.3.15. `mysql_error()`

```
const char *mysql_error(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_error()` returns a null-terminated string containing the error message for the most recently invoked API function that failed. If a function didn't fail, the return value of `mysql_error()` may be the previous error or an empty string to indicate no error.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_error()` if they succeed.

For functions that reset `mysql_error()`, the following two tests are equivalent:

```
if(*mysql_error(&mysql))
{
    // an error occurred
}

if(mysql_error(&mysql)[0])
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages. See [Section 10.2, “Setting the Error Message Language”](#).

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

22.8.3.16. `mysql_escape_string()`

Use `mysql_real_escape_string()` instead!

This function is identical to `mysql_real_escape_string()` except that `mysql_real_escape_string()` takes a connection handler as its first argument and escapes the string according to the current character set. `mysql_escape_string()` does not take a connection argument and does not respect the current character set.

22.8.3.17. `mysql_fetch_field()`

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, MySQL returns the default blob length (8KB) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8KB size is chosen because MySQL doesn't know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

Return Values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left.

Errors

None.

Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

```
}
```

22.8.3.18. `mysql_fetch_field_direct()`

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)
```

Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. Use this function to retrieve the definition for an arbitrary column. Specify a value for `fieldnr` in the range from 0 to `mysql_num_fields(result)-1`.

Return Values

The `MYSQL_FIELD` structure for the specified column.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

22.8.3.19. `mysql_fetch_fields()`

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

Return Values

An array of `MYSQL_FIELD` structures for all columns of a result set.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
```

```
printf("Field %u is %s\n", i, fields[i].name);
}
```

22.8.3.20. `mysql_fetch_lengths()`

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you **must** use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

Return Values

An array of unsigned long integers representing the size of each column (not including any terminating null characters). `NULL` if an error occurred.

Errors

`mysql_fetch_lengths()` is valid only for the current row of the result set. It returns `NULL` if you call it before calling `mysql_fetch_row()` or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("Column %u is %lu bytes in length.\n",
              i, lengths[i]);
    }
}
```

22.8.3.21. `mysql_fetch_row()`

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

Description

Retrieves the next row of a result set. When used after `mysql_store_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve. When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by `mysql_num_fields(result)`. If `row` holds the return value from a call to `mysql_fetch_row()`, pointers to the values are accessed as `row[0]` to `row[mysql_num_fields(result)-1]`. `NULL` values in the row are indicated by `NULL` pointers.

The lengths of the field values in the row may be obtained by calling `mysql_fetch_lengths()`. Empty fields and fields containing `NULL` both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is `NULL`, the field is `NULL`; otherwise, the field is empty.

Return Values

A `MYSQL_ROW` structure for the next row. `NULL` if there are no more rows to retrieve or if an error occurred.

Errors

Note that error is not reset between calls to `mysql_fetch_row()`

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
    unsigned long *lengths;
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("[%.*s] ", (int) lengths[i],
                row[i] ? row[i] : "NULL");
    }
    printf("\n");
}
```

22.8.3.22. `mysql_field_count()`

```
unsigned int mysql_field_count(MYSQL *mysql)
```

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See [Section 22.8.11.1, “Why `mysql_store_result\(\)` Sometimes Returns `NULL` After `mysql_query\(\)` Returns Success”](#).

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```

MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql, query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}

```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether the statement was a `SELECT`.

22.8.3.23. `mysql_field_seek()`

```

MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET
offset)

```

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` retrieves the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

Return Values

The previous value of the field cursor.

Errors

None.

22.8.3.24. `mysql_field_tell()`

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

Return Values

The current offset of the field cursor.

Errors

None.

22.8.3.25. `mysql_free_result()`

```
void mysql_free_result(MYSQL_RES *result)
```

Description

Frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, and so forth. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

Do not attempt to access a result set after freeing it.

Return Values

None.

Errors

None.

22.8.3.26. `mysql_get_character_set_info()`

```
void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)
```

Description

This function provides information about the default client character set. The default character set may be changed with the `mysql_set_character_set()` function.

Example

This example shows the fields that are available in the `MY_CHARSET_INFO` structure:

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set+collation number: %d\n", cs.number);
    printf("character set name: %s\n", cs.name);
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

```
}
```

22.8.3.27. `mysql_get_client_info()`

```
const char *mysql_get_client_info(void)
```

Description

Returns a string that represents the client library version.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

22.8.3.28. `mysql_get_client_version()`

```
unsigned long mysql_get_client_version(void)
```

Description

Returns an integer that represents the client library version. The value has the format `XYZZ` where `X` is the major version, `YY` is the release level, and `ZZ` is the version number within the release level. For example, a value of `40102` represents a client library version of `4.1.2`.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

22.8.3.29. `mysql_get_host_info()`

```
const char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server host name.

Return Values

A character string representing the server host name and the connection type.

Errors

None.

22.8.3.30. `mysql_get_proto_info()`

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Description

Returns the protocol version used by current connection.

Return Values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

22.8.3.31. `mysql_get_server_info()`

```
const char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the server version number.

Return Values

A character string that represents the server version number.

Errors

None.

22.8.3.32. `mysql_get_server_version()`

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns the version number of the server as an integer.

Return Values

A number that represents the MySQL server version in this format:

```
major_version*10000 + minor_version *100 + sub_version
```

For example, 5.1.5 is returned as 50105.

This function is useful in client programs for quickly determining whether some version-specific server capability exists.

Errors

None.

22.8.3.33. `mysql_get_ssl_cipher()`

```
const char *mysql_get_ssl_cipher(MYSQL *mysql)
```

Description

`mysql_get_ssl_cipher()` returns the SSL cipher used for the given connection to the server. `mysql` is the connection handler returned from `mysql_init()`.

Return Values

A string naming the SSL cipher used for the connection, or `NULL` if no cipher is being used.

22.8.3.34. `mysql_hex_string()`

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long
length)
```

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See [Section 9.1.1, “String Literals”](#).

The string in `from` is encoded to hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in `to` and a terminating null byte is appended.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_hex_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

The return value can be placed into an SQL statement using either `0xvalue` or `X'value'` format. However, the return value does not include the `0x` or `X'...`. The caller must supply whichever of those is desired.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"What is this",12);
end = strmov(end,"0x");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
*end++ = '\0';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\n",
        mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

22.8.3.35. `mysql_info()`

```
const char *mysql_info(MYSQL *mysql)
```

Description

Retrieves a string providing information about the most recently executed statement, but only for the statements listed here. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of statement, as described here. The numbers are illustrative only; the string contains values appropriate for the statement.

- `INSERT INTO ... SELECT ...`

String format: `Records: 100 Duplicates: 0 Warnings: 0`

- `INSERT INTO ... VALUES (...),(...),(...)...`

String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `LOAD DATA INFILE ...`

String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`

- `ALTER TABLE`

String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `UPDATE`

String format: `Rows matched: 40 Changed: 40 Warnings: 0`

Note that `mysql_info()` returns a non-NULL value for `INSERT ... VALUES` only for the multiple-row form of the statement (that is, only if multiple value lists are specified).

Return Values

A character string representing additional information about the most recently executed statement. `NULL` if no information is available for the statement.

Errors

None.

22.8.3.36. `mysql_init()`

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a `MYSQL` object suitable for `mysql_real_connect()`. If `mysql` is a `NULL` pointer, the function allocates, initializes, and returns a new object. Otherwise, the object is initialized and the address of the object is returned. If `mysql_init()` allocates a new object, it is freed when `mysql_close()` is called to close the connection.

Return Values

An initialized `MYSQL*` handle. `NULL` if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, `NULL` is returned.

22.8.3.37. `mysql_insert_id()`

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` or `UPDATE` statement. Use this function after you have performed an `INSERT` statement into a table that

contains an `AUTO_INCREMENT` field, or have used `INSERT` or `UPDATE` to set a column value with `LAST_INSERT_ID(expr)` [1194].

The return value of `mysql_insert_id()` is always zero unless explicitly updated under one of the following conditions:

- `INSERT` statements that store a value into an `AUTO_INCREMENT` column. This is true whether the value is automatically generated by storing the special values `NULL` or `0` into the column, or is an explicit nonspecial value.
- In the case of a multiple-row `INSERT` statement, `mysql_insert_id()` returns the first automatically generated `AUTO_INCREMENT` value that was successfully inserted.

If no rows are successfully inserted, `mysql_insert_id()` returns 0.

- If an `INSERT ... SELECT` statement is executed, and no automatically generated value is successfully inserted, `mysql_insert_id()` returns the ID of the last inserted row.
- If an `INSERT ... SELECT` statement uses `LAST_INSERT_ID(expr)` [1194], `mysql_insert_id()` returns *expr*.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by inserting `LAST_INSERT_ID(expr)` [1194] into any column or by updating any column to `LAST_INSERT_ID(expr)` [1194].
- If the previous statement returned an error, the value of `mysql_insert_id()` is undefined.

The return value of `mysql_insert_id()` can be simplified to the following sequence:

1. If there is an `AUTO_INCREMENT` column, and an automatically generated value was successfully inserted, return the first such value.
2. If `LAST_INSERT_ID(expr)` [1194] occurred in the statement, return *expr*, even if there was an `AUTO_INCREMENT` column in the affected table.
3. The return value varies depending on the statement used. When called after an `INSERT` statement:
 - If there is an `AUTO_INCREMENT` column in the table, and there were some explicit values for this column that were successfully inserted into the table, return the last of the explicit values.

When called after an `INSERT ... ON DUPLICATE KEY UPDATE` statement:

- If there is an `AUTO_INCREMENT` column in the table and there were some explicit successfully inserted values or some updated values, return the last of the inserted or updated values.

`mysql_insert_id()` returns 0 if the previous statement does not use an `AUTO_INCREMENT` value. If you need to save the value for later, be sure to call `mysql_insert_id()` immediately after the statement that generates the value.

The value of `mysql_insert_id()` is affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

The `LAST_INSERT_ID()` [1194] SQL function will contain the value of the first automatically generated value that was successfully inserted. `LAST_INSERT_ID()` [1194] is not reset between statements because the value of that function is maintained in the server. Another difference from `mysql_insert_id()` is that `LAST_INSERT_ID()` [1194] is not updated if you set an `AUTO_INCREMENT` column to a specific nonspecial value. See [Section 12.14, "Information Functions"](#).

`mysql_insert_id()` returns 0 following a `CALL` statement for a stored procedure that generates an `AUTO_INCREMENT` value because in this case `mysql_insert_id()` applies to `CALL` and not the statement within the procedure. Within the procedure, you can use `LAST_INSERT_ID()` [1194] at the SQL level to obtain the `AUTO_INCREMENT` value.

The reason for the differences between `LAST_INSERT_ID()` [1194] and `mysql_insert_id()` is that `LAST_INSERT_ID()` [1194] is made easy to use in scripts while `mysql_insert_id()` tries to provide more exact information about what happens to the `AUTO_INCREMENT` column.

Return Values

Described in the preceding discussion.

Errors

None.

22.8.3.38. `mysql_kill()`

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by `pid`.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `KILL` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [4030]
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR` [4030]
The MySQL server has gone away.
- `CR_SERVER_LOST` [4030]
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR` [4029]
An unknown error occurred.

22.8.3.39. `mysql_library_end()`

```
void mysql_library_end(void)
```

Description

This function finalizes the MySQL library. Call it when you are done using the library (for example, after disconnecting from the server). The action taken by the call depends on whether your application is linked

to the MySQL client library or the MySQL embedded server library. For a client program linked against the `libmysqlclient` library by using the `-libmysqlclient` flag, `mysql_library_end()` performs some memory management to clean up. For an embedded server application linked against the `libmysqld` library by using the `-libmysqld` flag, `mysql_library_end()` shuts down the embedded server and then cleans up.

For usage information, see [Section 22.8.2, “C API Function Overview”](#), and [Section 22.8.3.40, “mysql_library_init\(\)”](#).

22.8.3.40. `mysql_library_init()`

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

Call this function to initialize the MySQL library before you call any other MySQL function, whether your application is a regular client program or uses the embedded server. If the application uses the embedded server, this call starts the server and initializes any subsystems (`mysys`, `InnoDB`, and so forth) that the server uses.

After your application is done using the MySQL library, call `mysql_library_end()` to clean up. See [Section 22.8.3.39, “mysql_library_end\(\)”](#).

The choice of whether the application operates as a regular client or uses the embedded server depends on whether you use the `libmysqlclient` or `libmysqld` library at link time to produce the final executable. For additional information, see [Section 22.8.2, “C API Function Overview”](#).

In a nonmulti-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. Do this prior to any other client library call.

The `argc` and `argv` arguments are analogous to the arguments to `main()`, and enable passing of options to the embedded server. For convenience, `argc` may be 0 (zero) if there are no command-line arguments for the server. This is the usual case for applications intended for use only as regular (nonembedded) clients, and the call typically is written as `mysql_library_init(0, NULL, NULL)`.

```
#include <mysql.h>
#include <stdlib.h>

int main(void) {
    if (mysql_library_init(0, NULL, NULL)) {
        fprintf(stderr, "could not initialize MySQL library\n");
        exit(1);
    }

    /* Use any MySQL API functions here */

    mysql_library_end();

    return EXIT_SUCCESS;
}
```

When arguments are to be passed (`argc` is greater than 0), the first element of `argv` is ignored (it typically contains the program name). `mysql_library_init()` makes a copy of the arguments so it is safe to destroy `argv` or `groups` after the call.

For embedded applications, if you want to connect to an external server without starting the embedded server, you have to specify a negative value for `argc`.

The `groups` argument is an array of strings that indicate the groups in option files from which to read options. See [Section 4.2.3.3, “Using Option Files”](#). Make the final entry in the array `NULL`. For convenience, if the `groups` argument itself is `NULL`, the `[server]` and `[embedded]` groups are used by default.

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
    "this_program",          /* this string is not used */
    "--datadir=",
    "--key_buffer_size=32M"
};
static char *server_groups[] = {
    "embedded",
    "server",
    "this_program_SERVER",
    (char *)NULL
};

int main(void) {
    if (mysql_library_init(sizeof(server_args) / sizeof(char *),
                          server_args, server_groups)) {
        fprintf(stderr, "could not initialize MySQL library\n");
        exit(1);
    }

    /* Use any MySQL API functions here */

    mysql_library_end();

    return EXIT_SUCCESS;
}
```

Return Values

Zero if successful. Nonzero if an error occurred.

22.8.3.41. `mysql_list_dbs()`

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters “%” or “_”, or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW DATABASES [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [4030]

Commands were executed in an improper order.

- [CR_OUT_OF_MEMORY \[4030\]](#)

Out of memory.

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.3.42. `mysql_list_fields()`

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

Description

Returns a result set consisting of field names in the given table that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters “%” or “_”, or may be a `NULL` pointer to match all fields. Calling `mysql_list_fields()` is similar to executing the query `SHOW COLUMNS FROM tbl_name [LIKE wild]`.

It is preferable to use `SHOW COLUMNS FROM tbl_name` instead of `mysql_list_fields()`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.3.43. `mysql_list_processes()`

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by `mysqladmin processlist` or a `SHOW PROCESSLIST` query.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [4030]
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR` [4030]
The MySQL server has gone away.
- `CR_SERVER_LOST` [4030]
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR` [4029]
An unknown error occurred.

22.8.3.44. `mysql_list_tables()`

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters “%” or “_”, or may be a `NULL` pointer to match all tables. Calling `mysql_list_tables()` is similar to executing the query `SHOW TABLES [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [4030]
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR` [4030]
The MySQL server has gone away.
- `CR_SERVER_LOST` [4030]

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.3.45. `mysql_more_results()`

```
my_bool mysql_more_results(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you execute `CALL` statements, which can return multiple result sets.

`mysql_more_results()` true if more results exist from the currently executed statement, in which case the application must call `mysql_next_result()` to fetch the results.

Return Values

`TRUE` (1) if more results exist. `FALSE` (0) if no more results exist.

In most cases, you can call `mysql_next_result()` instead to test whether more results exist and initiate retrieval if so.

See [Section 22.8.13, “C API Support for Multiple Statement Execution”](#), and [Section 22.8.3.46, “`mysql_next_result\(\)`”](#).

Errors

None.

22.8.3.46. `mysql_next_result()`

```
int mysql_next_result(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you use `CALL` statements to execute stored procedures, which can return multiple result sets.

`mysql_next_result()` reads the next statement result and returns a status to indicate whether more results exist. If `mysql_next_result()` returns an error, there are no more results.

Before each call to `mysql_next_result()`, you must call `mysql_free_result()` for the current statement if it is a statement that returned a result set (rather than just a result status).

After calling `mysql_next_result()` the state of the connection is as if you had called `mysql_real_query()` or `mysql_query()` for the next statement. This means that you can call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If your program uses `CALL` statements to execute stored procedures, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. Because `CALL` can return multiple results, process them using a loop that calls `mysql_next_result()` to determine whether there are more results.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). As of MySQL 5.5.3, `CLIENT_MULTI_RESULTS` is enabled by default.

It is also possible to test whether there are more results by calling `mysql_more_results()`. However, this function does not change the connection state, so if it returns true, you must still call `mysql_next_result()` to advance to the next result.

For an example that shows how to use `mysql_next_result()`, see [Section 22.8.13, “C API Support for Multiple Statement Execution”](#).

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results
>0	An error occurred

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)

Commands were executed in an improper order. For example, if you didn't call `mysql_use_result()` for a previous result set.

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.3.47. `mysql_num_fields()`

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

To pass a `MYSQL*` argument instead, use `unsigned int mysql_field_count(MYSQL *mysql)`.

Description

Returns the number of columns in a result set.

Note that you can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if `mysql_store_result()` or `mysql_use_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See [Section 22.8.11.1, “Why `mysql_store_result\(\)` Sometimes Returns `NULL` After `mysql_query\(\)` Returns Success”](#).

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```

MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql, query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}

```

An alternative (if you know that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check whether `mysql_field_count(&mysql)` returns 0. This happens only if something went wrong.

22.8.3.48. `mysql_num_rows()`

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` does not return the correct value until all the rows in the result set have been retrieved.

`mysql_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

22.8.3.49. `mysql_options()`

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const void *arg)
```

Description

Can be used to set extra connect options and affect behavior for a connection. This function may be called multiple times to set several options.

Call `mysql_options()` after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, specify a pointer to the value of the integer as the `arg` argument.

The following list describes the possible options, their effect, and how `arg` is used for each option. Several of the options apply only when the application is linked against the `libmysqld` embedded server library and are unused for applications linked against the `libmysql` client library. For option descriptions that indicate `arg` is unused, its value is irrelevant; it is conventional to pass 0.

- `MYSQL_DEFAULT_AUTH` (argument type: `char *`)

The name of the authentication plugin to use. This option was added in MySQL 5.5.7.

- `MYSQL_ENABLE_CLEARTEXT_PLUGIN` (argument type: `my_bool *`)

Enable the `mysql_clear_password` cleartext authentication plugin. (See [Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”](#).) This option was added in MySQL 5.5.27.

- `MYSQL_INIT_COMMAND` (argument type: `char *`)

SQL statement to execute when connecting to the MySQL server. Automatically re-executed if reconnection occurs.

- `MYSQL_OPT_COMPRESS` (argument: not used)

Use the compressed client/server protocol.

- `MYSQL_OPT_CONNECT_TIMEOUT` (argument type: `unsigned int *`)

Connect timeout in seconds.

- `MYSQL_OPT_GUESS_CONNECTION` (argument: not used)

For an application linked against the `libmysqld` embedded server library, this enables the library to guess whether to use the embedded server or a remote server. “Guess” means that if

the host name is set and is not `localhost`, it uses a remote server. This behavior is the default. `MYSQL_OPT_USE_EMBEDDED_CONNECTION` and `MYSQL_OPT_USE_REMOTE_CONNECTION` can be used to override it. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_LOCAL_INFILE` (argument type: optional pointer to `unsigned int`)

If no pointer is given or if pointer points to an `unsigned int` that has a nonzero value, the `LOAD LOCAL INFILE` statement is enabled.

- `MYSQL_OPT_NAMED_PIPE` (argument: not used)

Use named pipes to connect to a MySQL server on Windows, if the server permits named-pipe connections.

- `MYSQL_OPT_PROTOCOL` (argument type: `unsigned int *`)

Type of protocol to use. Specify one of the enum values of `mysql_protocol_type` defined in `mysql.h`.

- `MYSQL_OPT_READ_TIMEOUT` (argument type: `unsigned int *`)

The timeout in seconds for attempts to read from the server. Each attempt uses this timeout value and there are retries if necessary, so the total effective timeout value is three times the option value. You can set the value so that a lost connection can be detected earlier than the TCP/IP `Close_Wait_Timeout` value of 10 minutes.

- `MYSQL_OPT_RECONNECT` (argument type: `my_bool *`)

Enable or disable automatic reconnection to the server if the connection is found to have been lost. Reconnect is off by default; this option provides a way to set reconnection behavior explicitly.

- `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` (argument type: `my_bool *`)

Enable or disable verification of the server's Common Name value in its certificate against the host name used when connecting to the server. The connection is rejected if there is a mismatch. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default.

- `MYSQL_OPT_USE_EMBEDDED_CONNECTION` (argument: not used)

For an application linked against the `libmysqld` embedded server library, this forces the use of the embedded server for the connection. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_USE_REMOTE_CONNECTION` (argument: not used)

For an application linked against the `libmysqld` embedded server library, this forces the use of a remote server for the connection. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_USE_RESULT` (argument: not used)

This option is unused.

- `MYSQL_OPT_WRITE_TIMEOUT` (argument type: `unsigned int *`)

The timeout in seconds for attempts to write to the server. Each attempt uses this timeout value and there are `net_retry_count` [532] retries if necessary, so the total effective timeout value is `net_retry_count` [532] times the option value.

- `MYSQL_PLUGIN_DIR` (argument type: `char *`)
The directory in which to look for client plugins. This option was added in MySQL 5.5.7.
- `MYSQL_READ_DEFAULT_FILE` (argument type: `char *`)
Read options from the named option file instead of from `my.cnf`.
- `MYSQL_READ_DEFAULT_GROUP` (argument type: `char *`)
Read options from the named group from `my.cnf` or the file specified with `MYSQL_READ_DEFAULT_FILE`.
- `MYSQL_REPORT_DATA_TRUNCATION` (argument type: `my_bool *`)
Enable or disable reporting of data truncation errors for prepared statements using the `error` member of `MYSQL_BIND` structures. (Default: enabled.)
- `MYSQL_SECURE_AUTH` (argument type: `my_bool *`)
Whether to connect to a server that does not support the password hashing used in MySQL 4.1.1 and later.
- `MYSQL_SET_CHARSET_DIR` (argument type: `char *`)
The path name to the directory that contains character set definition files.
- `MYSQL_SET_CHARSET_NAME` (argument type: `char *`)
The name of the character set to use as the default character set. The argument can be `MYSQL_AUTODETECT_CHARSET_NAME` to cause the character set to be autodetected based on the operating system setting (see [Section 10.1.4](#), “[Connection Character Sets and Collations](#)”).
- `MYSQL_SET_CLIENT_IP` (argument type: `char *`)
For an application linked against the `libmysqld` embedded server library (when `libmysqld` is compiled with authentication support), this means that the user is considered to have connected from the specified IP address (specified as a string) for authentication purposes. This option is ignored for applications linked against the `libmysqlclient` client library.
- `MYSQL_SHARED_MEMORY_BASE_NAME` (argument type: `char *`)
The name of the shared-memory object for communication to the server on Windows, if the server supports shared-memory connections. Specify the same value as the `--shared-memory-base-name [460]` option used for the `mysqld` server you want to connect to.

The `client` group is always read if you use `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP`.

The specified group in the option file may contain the following options.

Option	Description
<code>character-sets-dir=path</code>	The directory where character sets are installed.
<code>compress</code>	Use the compressed client/server protocol.
<code>connect-timeout=seconds</code>	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.

Option	Description
<code>database=db_name</code>	Connect to this database if no database was specified in the connect command.
<code>debug</code>	Debug options.
<code>default-character-set=charset_name</code>	The default character set to use.
<code>disable-local-infile</code>	Disable use of <code>LOAD DATA LOCAL</code> .
<code>enable-cleartext-plugin</code>	Enable the <code>mysql_clear_password</code> cleartext authentication plugin. Added in MySQL 5.5.27.
<code>host=host_name</code>	Default host name.
<code>init-command=stmt</code>	Statement to execute when connecting to MySQL server. Automatically re-executed if reconnection occurs.
<code>interactive-timeout=seconds</code>	Same as specifying <code>CLIENT_INTERACTIVE</code> to <code>mysql_real_connect()</code> . See Section 22.8.3.52 , “ <code>mysql_real_connect()</code> ”.
<code>local-infile[={0 1}]</code>	If no argument or nonzero argument, enable use of <code>LOAD DATA LOCAL</code> ; otherwise disable.
<code>max_allowed_packet=bytes</code>	Maximum size of packet that client can read from server.
<code>multi-queries, multi-results</code>	Enable multiple result sets from multiple-statement executions or stored procedures.
<code>multi-statements</code>	Enable the client to send multiple statements in a single string (separated by “;”).
<code>password=password</code>	Default password.
<code>pipe</code>	Use named pipes to connect to a MySQL server on Windows.
<code>port=port_num</code>	Default port number.
<code>protocol={TCP SOCKET PIPE MEMORY}</code>	The protocol to use when connecting to the server.
<code>return-found-rows</code>	Tell <code>mysql_info()</code> to return found rows instead of updated rows when using <code>UPDATE</code> .
<code>shared-memory-base-name=name</code>	Shared-memory name to use to connect to server.
<code>socket=path</code>	Default socket file.
<code>ssl-ca=file_name</code>	Certificate Authority file.
<code>ssl-capath=path</code>	Certificate Authority directory.
<code>ssl-cert=file_name</code>	Certificate file.
<code>ssl-cipher=cipher_list</code>	Permissible SSL ciphers.
<code>ssl-key=file_name</code>	Key file.
<code>timeout=seconds</code>	Like <code>connect-timeout</code> .
<code>user</code>	Default user.

`timeout` has been replaced by `connect-timeout`, but `timeout` is still supported in MySQL 5.5 for backward compatibility.

For more information about option files, see [Section 4.2.3.3](#), “Using Option Files”.

Return Values

Zero for success. Nonzero if you specify an unknown option.

Example

The following `mysql_options()` calls request the use of compression in the client/server protocol, cause options to be read from the `[odbc]` group of option files, and disable transaction autocommit mode:

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
mysql_options(&mysql,MYSQL_INIT_COMMAND,"SET autocommit=0");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr,"Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}
```

This code requests that the client use the compressed client/server protocol and read the additional options from the `odbc` section in the `my.cnf` file.

22.8.3.50. `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If the connection has gone down and auto-reconnect is enabled an attempt to reconnect is made. If the connection is down and auto-reconnect is disabled, `mysql_ping()` returns an error.

Auto-reconnect is disabled by default. To enable it, call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option. For details, see [Section 22.8.3.49, “mysql_options\(\)”](#).

`mysql_ping()` can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

If `mysql_ping()` does cause a reconnect, there is no explicit indication of it. To determine whether a reconnect occurs, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier has changed.

If reconnect occurs, some characteristics of the connection will have been reset. For details about these characteristics, see [Section 22.8.12, “Controlling Automatic Reconnection Behavior”](#).

Return Values

Zero if the connection to the server is active. Nonzero if an error occurred. A nonzero return does not indicate whether the MySQL server itself is down; the connection might be broken for other reasons such as network problems.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.3.51. `mysql_query()`

```
int mysql_query(MYSQL *mysql, const char *stmt_str)
```

Description

Executes the SQL statement pointed to by the null-terminated string `stmt_str`. Normally, the string must consist of a single SQL statement without a terminating semicolon (“;”) or `\g`. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See [Section 22.8.13, “C API Support for Multiple Statement Execution”](#).

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the “\0” character, which `mysql_query()` interprets as the end of the statement string.)

If you want to know whether the statement should return a result set, you can use `mysql_field_count()` to check for this. See [Section 22.8.3.22, “mysql_field_count\(\)”](#).

Return Values

Zero if the statement was successful. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.3.52. `mysql_real_connect()`

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,  
const char *passwd, const char *db, unsigned int port, const char *unix_socket,  
unsigned long client_flag)
```

Description

`mysql_real_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_real_connect()` must complete successfully before you can execute any other API functions that require a valid `MYSQL` connection handle structure.

The parameters are specified as follows:

- For the first parameter, specify the address of an existing `MYSQL` structure. Before calling `mysql_real_connect()`, call `mysql_init()` to initialize the `MYSQL` structure. You can change a lot of connect options with the `mysql_options()` call. See [Section 22.8.3.49](#), “`mysql_options()`”.
- The value of `host` may be either a host name or an IP address. If `host` is `NULL` or the string “`localhost`”, a connection to the local host is assumed. For Windows, the client connects using a shared-memory connection, if the server has shared-memory connections enabled. Otherwise, TCP/IP is used. For Unix, the client connects using a Unix socket file. For local connections, you can also influence the type of connection to use with the `MYSQL_OPT_PROTOCOL` or `MYSQL_OPT_NAMED_PIPE` options to `mysql_options()`. The type of connection must be supported by the server. For a `host` value of “.” on Windows, the client connects using a named pipe, if the server has named-pipe connections enabled. If named-pipe connections are not enabled, an error occurs.
- The `user` parameter contains the user's MySQL login ID. If `user` is `NULL` or the empty string “”, the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current user name must be specified explicitly. See the Connector/ODBC section of [Chapter 22, Connectors and APIs](#).
- The `passwd` parameter contains the password for `user`. If `passwd` is `NULL`, only entries in the `user` table for the user that have a blank (empty) password field are checked for a match. This enables the database administrator to set up the MySQL privilege system in such a way that users get different privileges depending on whether they have specified a password.



Note

Do not attempt to encrypt the password before calling `mysql_real_connect()`; password encryption is handled automatically by the client API.

- The `user` and `passwd` parameters use whatever character set has been configured for the `MYSQL` object. By default, this is `latin1`, but can be changed by calling `mysql_options(mysql, MYSQL_SET_CHARSET_NAME, "charset_name")` prior to connecting.
- `db` is the database name. If `db` is not `NULL`, the connection sets the default database to this value.
- If `port` is not 0, the value is used as the port number for the TCP/IP connection. Note that the `host` parameter determines the type of the connection.
- If `unix_socket` is not `NULL`, the string specifies the socket or named pipe to use. Note that the `host` parameter determines the type of the connection.
- The value of `client_flag` is usually 0, but can be set to a combination of the following flags to enable certain features.

Flag Name	Flag Description
<code>CLIENT_COMPRESS</code>	Use compression protocol.
<code>CLIENT_FOUND_ROWS</code>	Return the number of found (matched) rows, not the number of changed rows.
<code>CLIENT_IGNORE_SIGPIPE</code>	Prevents the client library from installing a <code>SIGPIPE</code> signal handler. This can be used to avoid conflicts with a handler that the application has already installed.
<code>CLIENT_IGNORE_SPACE</code>	Permit spaces after function names. Makes all functions names reserved words.

Flag Name	Flag Description
<code>CLIENT_INTERACTIVE</code>	Permit <code>interactive_timeout</code> [506] seconds (instead of <code>wait_timeout</code> [572] seconds) of inactivity before closing the connection. The client's session <code>wait_timeout</code> [572] variable is set to the value of the session <code>interactive_timeout</code> [506] variable.
<code>CLIENT_LOCAL_FILES</code>	Enable <code>LOAD DATA LOCAL</code> handling.
<code>CLIENT_MULTI_RESULTS</code>	Tell the server that the client can handle multiple result sets from multiple-statement executions or stored procedures. This flag is automatically enabled if <code>CLIENT_MULTI_STATEMENTS</code> is enabled. See the note following this table for more information about this flag.
<code>CLIENT_MULTI_STATEMENTS</code>	Tell the server that the client may send multiple statements in a single string (separated by “;”). If this flag is not set, multiple-statement execution is disabled. See the note following this table for more information about this flag.
<code>CLIENT_NO_SCHEMA</code>	Do not permit the <code>db_name.tbl_name.col_name</code> syntax. This is for ODBC. It causes the parser to generate an error if you use that syntax, which is useful for trapping bugs in some ODBC programs.
<code>CLIENT_ODBC</code>	Unused.
<code>CLIENT_SSL</code>	Use SSL (encrypted protocol). Do not set this option within an application program; it is set internally in the client library. Instead, use <code>mysql_ssl_set()</code> before calling <code>mysql_real_connect()</code> .
<code>CLIENT_REMEMBER_OPTIONS</code>	Remember options specified by calls to <code>mysql_options()</code> . Without this option, if <code>mysql_real_connect()</code> fails, you must repeat the <code>mysql_options()</code> calls before trying to connect again. With this option, the <code>mysql_options()</code> calls need not be repeated.

If your program uses `CALL` statements to execute stored procedures, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. Because `CALL` can return multiple results, process them using a loop that calls `mysql_next_result()` to determine whether there are more results.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). As of MySQL 5.5.3, `CLIENT_MULTI_RESULTS` is enabled by default.

If you enable `CLIENT_MULTI_STATEMENTS` or `CLIENT_MULTI_RESULTS`, you should process the result for every call to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 22.8.13, “C API Support for Multiple Statement Execution”](#).

For some parameters, it is possible to have the value taken from an option file rather than from an explicit value in the `mysql_real_connect()` call. To do this, call `mysql_options()` with the `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP` option before calling `mysql_real_connect()`. Then, in the `mysql_real_connect()` call, specify the “no-value” value for each parameter to be read from an option file:

- For `host`, specify a value of `NULL` or the empty string (“”).

- For `user`, specify a value of `NULL` or the empty string.
- For `passwd`, specify a value of `NULL`. (For the password, a value of the empty string in the `mysql_real_connect()` call cannot be overridden in an option file, because the empty string indicates explicitly that the MySQL account must have an empty password.)
- For `db`, specify a value of `NULL` or the empty string.
- For `port`, specify a value of 0.
- For `unix_socket`, specify a value of `NULL`.

If no value is found in an option file for a parameter, its default value is used as indicated in the descriptions given earlier in this section.

Return Values

A `MYSQL*` connection handle if the connection was successful, `NULL` if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter.

Errors

- `CR_CONN_HOST_ERROR` [4030]
Failed to connect to the MySQL server.
- `CR_CONNECTION_ERROR` [4029]
Failed to connect to the local MySQL server.
- `CR_IPSOCK_ERROR` [4030]
Failed to create an IP socket.
- `CR_OUT_OF_MEMORY` [4030]
Out of memory.
- `CR_SOCKET_CREATE_ERROR` [4029]
Failed to create a Unix socket.
- `CR_UNKNOWN_HOST` [4030]
Failed to find the IP address for the host name.
- `CR_VERSION_ERROR` [4030]
A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version.
- `CR_NAMEDPIPEOPEN_ERROR` [4030]
Failed to create a named pipe on Windows.
- `CR_NAMEDPIPEWAIT_ERROR` [4030]
Failed to wait for a named pipe on Windows.

- [CR_NAMEDPIPESETSTATE_ERROR \[4031\]](#)
Failed to get a pipe handler on Windows.
- [CR_SERVER_LOST \[4030\]](#)
If [connect_timeout \[492\]](#) > 0 and it took longer than [connect_timeout \[492\]](#) seconds to connect to the server or if the server died while executing the `init-command`.
- [CR_ALREADY_CONNECTED \[4033\]](#)
The `MYSQL` connection handle is already connected.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}
```

By using `mysql_options()` the MySQL library reads the `[client]` and `[your_prog_name]` sections in the `my.cnf` file which ensures that your program works, even if someone has set up MySQL in some nonstandard way.

Note that upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. You can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior.

22.8.3.53. `mysql_real_escape_string()`

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char
*from, unsigned long length)
```

Note that `mysql` must be a valid, open connection. This is needed because the escaping depends on the character set in use by the server.

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See [Section 9.1.1, “String Literals”](#).

The string in `from` is encoded to an escaped SQL string, taking into account the current character set of the connection. The result is placed in `to` and a terminating null byte is appended. Characters encoded are “\”, “'”, “\”, `NUL` (ASCII 0), “\n”, “\r”, and Control+Z. Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. `mysql_real_escape_string()` quotes the other characters to make them easier to read in log files. For comparison, see the quoting rules for literal strings and the `QUOTE()` [\[1107\]](#) SQL function in [Section 9.1.1, “String Literals”](#), and [Section 12.5, “String Functions”](#).

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. (In the worst case, each character may need to be encoded as using two bytes, and you need room for the terminating null byte.) When `mysql_real_escape_string()` returns, the

contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

If you need to change the character set of the connection, use the `mysql_set_character_set()` function rather than executing a `SET NAMES` (or `SET CHARACTER SET`) statement.

`mysql_set_character_set()` works like `SET NAMES` but also affects the character set used by `mysql_real_escape_string()`, which `SET NAMES` does not.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\\';
end += mysql_real_escape_string(&mysql, end,"What is this",12);
*end++ = '\\';
*end++ = ',';
*end++ = '\\';
end += mysql_real_escape_string(&mysql, end,"binary data: \\0\\r\\n",16);
*end++ = '\\';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\\n",
            mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

22.8.3.54. `mysql_real_query()`

```
int mysql_real_query(MYSQL *mysql, const char *stmt_str, unsigned long length)
```

Description

Executes the SQL statement pointed to by `stmt_str`, a string `length` bytes long. Normally, the string must consist of a single SQL statement without a terminating semicolon (“;”) or `\g`. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See [Section 22.8.13, “C API Support for Multiple Statement Execution”](#).

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the “\0” character, which `mysql_query()` interprets as the end of the statement string.) In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the statement string.

If you want to know whether the statement should return a result set, you can use `mysql_field_count()` to check for this. See [Section 22.8.3.22, “mysql_field_count\(\)”](#).

Return Values

Zero if the statement was successful. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

22.8.3.55. `mysql_refresh()`

```
int mysql_refresh(MYSQL *mysql, unsigned int options)
```

Description

This function flushes tables or caches, or resets replication server information. The connected user must have the [RELOAD \[696\]](#) privilege.

The `options` argument is a bit mask composed from any combination of the following values. Multiple values can be OR'ed together to perform multiple operations with a single call.

- [REFRESH_GRANT](#)
Refresh the grant tables, like [FLUSH PRIVILEGES](#).
- [REFRESH_LOG](#)
Flush the logs, like [FLUSH LOGS](#).
- [REFRESH_TABLES](#)
Flush the table cache, like [FLUSH TABLES](#).
- [REFRESH_HOSTS](#)
Flush the host cache, like [FLUSH HOSTS](#).
- [REFRESH_STATUS](#)
Reset status variables, like [FLUSH STATUS](#).
- [REFRESH_THREADS](#)
Flush the thread cache.
- [REFRESH_SLAVE](#)
On a slave replication server, reset the master server information and restart the slave, like [RESET SLAVE](#).
- [REFRESH_MASTER](#)

On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like `RESET MASTER`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

22.8.3.56. `mysql_reload()`

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the MySQL server to reload the grant tables. The connected user must have the `RELOAD [696]` privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `FLUSH PRIVILEGES` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

22.8.3.57. `mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

The action of this function is subject to the value of the `completion_type` [489] system variable. In particular, if the value of `completion_type` [489] is `RELEASE` (or 2), the server performs a release after terminating a transaction and closes the client connection. Call `mysql_close()` from the client program to close the connection from the client side.

Return Values

Zero if successful. Nonzero if an error occurred.

Errors

None.

22.8.3.58. `mysql_row_seek()`

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. The `offset` value is a row offset, typically a value returned from `mysql_row_tell()` or from `mysql_row_seek()`. This value is not a row number; to seek to a row within a result set by number, use `mysql_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

22.8.3.59. `mysql_row_tell()`

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

Use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

22.8.3.60. `mysql_select_db()`

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

22.8.3.61. `mysql_set_character_set()`

```
int mysql_set_character_set(MYSQL *mysql, const char *csname)
```

Description

This function is used to set the default character set for the current connection. The string `csname` specifies a valid character set name. The connection collation becomes the default collation of the character set. This function works like the `SET NAMES` statement, but also sets the value of `mysql->charset`, and thus affects the character set used by `mysql_real_escape_string()`

Return Values

Zero for success. Nonzero if an error occurred.

Example

```
MYSQL mysql;
```

```
mysql_init(&mysql);
if (!mysql_real_connect(&mysql, "host", "user", "passwd", "database", 0, NULL, 0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}

if (!mysql_set_character_set(&mysql, "utf8"))
{
    printf("New client character set: %s\n",
            mysql_character_set_name(&mysql));
}
```

22.8.3.62. `mysql_set_local_infile_default()`

```
void mysql_set_local_infile_default(MYSQL *mysql);
```

Description

Sets the `LOAD LOCAL DATA INFILE` handler callback functions to the defaults used internally by the C client library. The library calls this function automatically if `mysql_set_local_infile_handler()` has not been called or does not supply valid functions for each of its callbacks.

Return Values

None.

Errors

None.

22.8.3.63. `mysql_set_local_infile_handler()`

```
void mysql_set_local_infile_handler(MYSQL *mysql, int (*local_infile_init)(void
**, const char *, void *), int (*local_infile_read)(void *, char *, unsigned
int), void (*local_infile_end)(void *), int (*local_infile_error)(void *,
char*, unsigned int), void *userdata);
```

Description

This function installs callbacks to be used during the execution of `LOAD DATA LOCAL INFILE` statements. It enables application programs to exert control over local (client-side) data file reading. The arguments are the connection handler, a set of pointers to callback functions, and a pointer to a data area that the callbacks can use to share information.

To use `mysql_set_local_infile_handler()`, you must write the following callback functions:

```
int
local_infile_init(void **ptr, const char *filename, void *userdata);
```

The initialization function. This is called once to do any setup necessary, open the data file, allocate data structures, and so forth. The first `void**` argument is a pointer to a pointer. You can set the pointer (that is, `*ptr`) to a value that will be passed to each of the other callbacks (as a `void*`). The callbacks can use this pointed-to value to maintain state information. The `userdata` argument is the same value that is passed to `mysql_set_local_infile_handler()`.

The initialization function should return zero for success, nonzero for an error.

```
int
local_infile_read(void *ptr, char *buf, unsigned int buf_len);
```

The data-reading function. This is called repeatedly to read the data file. `buf` points to the buffer where the read data is stored, and `buf_len` is the maximum number of bytes that the callback can read and store in the buffer. (It can read fewer bytes, but should not read more.)

The return value is the number of bytes read, or zero when no more data could be read (this indicates EOF). Return a value less than zero if an error occurs.

```
void
local_infile_end(void *ptr)
```

The termination function. This is called once after `local_infile_read()` has returned zero (EOF) or an error. Within this function, deallocate any memory allocated by `local_infile_init()` and perform any other cleanup necessary. It is invoked even if the initialization function returns an error.

```
int
local_infile_error(void *ptr,
                  char *error_msg,
                  unsigned int error_msg_len);
```

The error-handling function. This is called to get a textual error message to return to the user in case any of your other functions returns an error. `error_msg` points to the buffer into which the message is written, and `error_msg_len` is the length of the buffer. Write the message as a null-terminated string, at most `error_msg_len-1` bytes long.

The return value is the error number.

Typically, the other callbacks store the error message in the data structure pointed to by `ptr`, so that `local_infile_error()` can copy the message from there into `error_msg`.

After calling `mysql_set_local_infile_handler()` in your C code and passing pointers to your callback functions, you can then issue a `LOAD DATA LOCAL INFILE` statement (for example, by using `mysql_query()`). The client library automatically invokes your callbacks. The file name specified in `LOAD DATA LOCAL INFILE` will be passed as the second parameter to the `local_infile_init()` callback.

Return Values

None.

Errors

None.

22.8.3.64. `mysql_set_server_option()`

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)
```

Description

Enables or disables an option for the connection. `option` can have one of the following values.

Option	Description
<code>MYSQL_OPTION_MULTI_STATEMENTS_ON</code>	Enable multiple-statement support
<code>MYSQL_OPTION_MULTI_STATEMENTS_OFF</code>	Disable multiple-statement support

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 22.8.13, “C API Support for Multiple Statement Execution”](#).

Enabling multiple-statement support with `MYSQL_OPTION_MULTI_STATEMENTS_ON` does not have quite the same effect as enabling it by passing the `CLIENT_MULTI_STATEMENTS` flag to `mysql_real_connect()`: `CLIENT_MULTI_STATEMENTS` also enables `CLIENT_MULTI_RESULTS`. If you are using the `CALL` SQL statement in your programs, multiple-result support must be enabled; this means that `MYSQL_OPTION_MULTI_STATEMENTS_ON` by itself is insufficient to permit the use of `CALL`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query.

- [ER_UNKNOWN_COM_ERROR \[3982\]](#)

The server didn't support `mysql_set_server_option()` (which is the case that the server is older than 4.1.1) or the server didn't support the option one tried to set.

22.8.3.65. `mysql_shutdown()`

```
int mysql_shutdown(MYSQL *mysql, enum mysql_enum_shutdown_level shutdown_level)
```

Description

Asks the database server to shut down. The connected user must have the [SHUTDOWN \[697\]](#) privilege. MySQL 5.5 servers support only one type of shutdown; `shutdown_level` must be equal to `SHUTDOWN_DEFAULT`. Additional shutdown levels are planned to make it possible to choose the desired level. Dynamically linked executables which have been compiled with older versions of the `libmysqlclient` headers and call `mysql_shutdown()` need to be used with the old `libmysqlclient` dynamic library.

The shutdown process is described in [Section 5.1.12, "The Shutdown Process"](#).

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.3.66. `mysql_sqlstate()`

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the SQLSTATE error code for the most recently executed SQL statement. The error code consists of five characters. '00000' means “no error.” The values are specified by ANSI SQL and ODBC. For a list of possible values, see [Appendix C, Errors, Error Codes, and Common Problems](#).

SQLSTATE values returned by `mysql_sqlstate()` differ from MySQL-specific error numbers returned by `mysql_errno()`. For example, the `mysql` client program displays errors using the following format, where 1146 is the `mysql_errno()` value and '42S02' is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Not all MySQL error numbers are mapped to SQLSTATE error codes. The value 'HY000' (general error) is used for unmapped error numbers.

If you call `mysql_sqlstate()` after `mysql_real_connect()` fails, `mysql_sqlstate()` might not return a useful value. For example, this happens if a host is blocked by the server and the connection is closed without any SQLSTATE value being sent to the client.

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See [Section 22.8.3.14](#), “`mysql_errno()`”, [Section 22.8.3.15](#), “`mysql_error()`”, and [Section 22.8.7.27](#), “`mysql_stmt_sqlstate()`”.

22.8.3.67. `mysql_ssl_set()`

```
my_bool mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const
char *ca, const char *capath, const char *cipher)
```

Description

`mysql_ssl_set()` is used for establishing secure connections using SSL. It must be called before `mysql_real_connect()`.

`mysql_ssl_set()` does nothing unless SSL support is enabled in the client library.

`mysql` is the connection handler returned from `mysql_init()`. The other parameters are specified as follows:

- `key` is the path name to the key file.

- `cert` is the path name to the certificate file.
- `ca` is the path name to the certificate authority file.
- `capath` is the path name to a directory that contains trusted SSL CA certificates in PEM format.
- `cipher` is a list of permissible ciphers to use for SSL encryption.

Any unused SSL parameters may be given as `NULL`.

Return Values

This function always returns 0. If SSL setup is incorrect, `mysql_real_connect()` returns an error when you attempt to connect.

22.8.3.68. `mysql_stat()`

```
const char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the `mysqladmin status` command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Return Values

A character string describing the server status. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC [4030]`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR [4030]`
The MySQL server has gone away.
- `CR_SERVER_LOST [4030]`
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR [4029]`
An unknown error occurred.

22.8.3.69. `mysql_store_result()`

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

You don't have to call `mysql_store_result()` or `mysql_use_result()` for other statements, but it does not do any harm or cause any notable performance degradation if you call `mysql_store_result()` in all cases. You can detect whether the statement has a result set by checking whether `mysql_store_result()` returns a nonzero value (more about this later).

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 22.8.13, "C API Support for Multiple Statement Execution"](#).

If you want to know whether a statement should return a result set, you can use `mysql_field_count()` to check for this. See [Section 22.8.3.22, "mysql_field_count\(\)"](#).

`mysql_store_result()` reads the entire result of a query to the client, allocates a `MYSQL_RES` structure, and places the result into this structure.

`mysql_store_result()` returns a null pointer if the statement didn't return a result set (for example, if it was an `INSERT` statement).

`mysql_store_result()` also returns a null pointer if reading of the result set failed. You can check whether an error occurred by checking whether `mysql_error()` returns a nonempty string, `mysql_errno()` returns nonzero, or `mysql_field_count()` returns zero.

An empty result set is returned if there are no rows returned. (An empty result set differs from a null pointer as a return value.)

After you have called `mysql_store_result()` and gotten back a result that isn't a null pointer, you can call `mysql_num_rows()` to find out how many rows are in the result set.

You can call `mysql_fetch_row()` to fetch rows from the result set, or `mysql_row_seek()` and `mysql_row_tell()` to obtain or set the current row position within the result set.

See [Section 22.8.11.1, "Why mysql_store_result\(\) Sometimes Returns NULL After mysql_query\(\) Returns Success"](#).

Return Values

A `MYSQL_RES` result structure with the results. `NULL` (0) if an error occurred.

Errors

`mysql_store_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC` [4030]

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY` [4030]

Out of memory.

- `CR_SERVER_GONE_ERROR` [4030]

The MySQL server has gone away.

- `CR_SERVER_LOST` [4030]

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.3.70. `mysql_thread_id()`

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID changes. This means you should not get the thread ID and store it for later. You should get it when you need it.



Note

This function does not work correctly if thread IDs become larger than 32 bits, which can occur on some systems. To avoid problems with `mysql_thread_id()`, do not use it. To get the connection ID, execute a `SELECT CONNECTION_ID()` query and retrieve the result.

Return Values

The thread ID of the current connection.

Errors

None.

22.8.3.71. `mysql_use_result()`

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client allocates memory only for the current row and a communication buffer that may grow up to `max_allowed_packet [519]` bytes.

On the other hand, you shouldn't use `mysql_use_result()` if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This ties up the server and prevent other threads from updating any tables from which the data is being fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise, the un fetched rows are returned as part of the result set for your next query. The C API gives the error `Commands out of sync; you can't run this command now` if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()`, or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other queries until `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` accurately returns the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

When using the `libmysqld` embedded server, the memory benefits are essentially lost because memory usage incrementally increases with each row retrieved until `mysql_free_result()` is called.

Return Values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

Errors

`mysql_use_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC` [4030]
Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY` [4030]
Out of memory.
- `CR_SERVER_GONE_ERROR` [4030]
The MySQL server has gone away.
- `CR_SERVER_LOST` [4030]
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR` [4029]
An unknown error occurred.

22.8.3.72. `mysql_warning_count()`

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of errors, warnings, and notes generated during execution of the previous SQL statement.

Return Values

The warning count.

Errors

None.

22.8.4. C API Prepared Statements

The MySQL client/server protocol provides for the use of prepared statements. This capability uses the `MYSQL_STMT` statement handler data structure returned by the `mysql_stmt_init()` initialization function. Prepared execution is an efficient way to execute a statement more than once. The statement is first parsed to prepare it for execution. Then it is executed one or more times at a later time, using the statement handle returned by the initialization function.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Prepared statements might not provide a performance increase in some situations. For best results, test your application both with prepared and nonprepared statements and choose whichever yields best performance.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

The following SQL statements can be used as prepared statements:

```
CALL
CREATE TABLE
DELETE
DO
INSERT
REPLACE
SELECT
SET
UPDATE
ANALYZE TABLE
OPTIMIZE TABLE
REPAIR TABLE
CACHE INDEX
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
{CREATE | RENAME | DROP} DATABASE
{CREATE | RENAME | DROP} USER
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
      | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
REVOKE
KILL
LOAD INDEX INTO CACHE
RESET {MASTER | SLAVE | QUERY CACHE}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
INSTALL PLUGIN
UNINSTALL PLUGIN
```

Other statements are not yet supported in MySQL 5.5.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic reparation of the statement when it is next executed. For more information, see [Section 13.5.4, “Automatic Prepared Statement Reparation”](#).

22.8.5. C API Prepared Statement Data Structures

Prepared statements use several data structures:

- To obtain a statement handle, pass a `MYSQL` connection handler to `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT` data structure. This structure is used for further operations with the statement. To specify the statement to prepare, pass the `MYSQL_STMT` pointer and the statement string to `mysql_stmt_prepare()`.
- To provide input parameters for a prepared statement, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_param()`. To receive output column values, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_result()`.
- The `MYSQL_TIME` structure is used to transfer temporal data in both directions.

The following discussion describes the prepared statement data types in detail. For examples that show how to use them, see [Section 22.8.7.10](#), “`mysql_stmt_execute()`”, and [Section 22.8.7.11](#), “`mysql_stmt_fetch()`”.

- `MYSQL_STMT`

This structure is a handle for a prepared statement. A handle is created by calling `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT`. The handle is used for all subsequent operations with the statement until you close it with `mysql_stmt_close()`, at which point the handle becomes invalid.

The `MYSQL_STMT` structure has no members intended for application use. Applications should not try to copy a `MYSQL_STMT` structure. There is no guarantee that such a copy will be usable.

Multiple statement handles can be associated with a single connection. The limit on the number of handles depends on the available system resources.

- `MYSQL_BIND`

This structure is used both for statement input (data values sent to the server) and output (result values returned from the server):

- For input, use `MYSQL_BIND` structures with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`.
- For output, use `MYSQL_BIND` structures with `mysql_stmt_bind_result()` to bind buffers to result set columns, for use in fetching rows with `mysql_stmt_fetch()`.

To use a `MYSQL_BIND` structure, zero its contents to initialize it, then set its members appropriately. For example, to declare and initialize an array of three `MYSQL_BIND` structures, use this code:

```
MYSQL_BIND bind[3];
memset(bind, 0, sizeof(bind));
```

The `MYSQL_BIND` structure contains the following members for use by application programs. For several of the members, the manner of use depends on whether the structure is used for input or output.

- `enum enum_field_types buffer_type`

The type of the buffer. This member indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored. For permissible `buffer_type` values, see [Section 22.8.5.1](#), “C API Prepared Statement Type Codes”.

- `void *buffer`

A pointer to the buffer to be used for data transfer. This is the address of a C language variable.

For input, `buffer` is a pointer to the variable in which you store the data value for a statement parameter. When you call `mysql_stmt_execute()`, MySQL use the value stored in the variable in place of the corresponding parameter marker in the statement (specified with `?` in the statement string).

For output, `buffer` is a pointer to the variable in which to return a result set column value. When you call `mysql_stmt_fetch()`, MySQL stores a column value from the current row of the result set in this variable. You can access the value when the call returns.

To minimize the need for MySQL to perform type conversions between C language values on the client side and SQL values on the server side, use C variables that have types similar to those of the corresponding SQL values:

- For numeric data types, `buffer` should point to a variable of the proper numeric C type. For integer variables (which can be `char` for single-byte values or an integer type for larger values), you should also indicate whether the variable has the `unsigned` attribute by setting the `is_unsigned` member, described later.
- For character (nonbinary) and binary string data types, `buffer` should point to a character buffer.
- For date and time data types, `buffer` should point to a `MYSQL_TIME` structure.

For guidelines about mapping between C types and SQL types and notes about type conversions, see [Section 22.8.5.1, “C API Prepared Statement Type Codes”](#), and [Section 22.8.5.2, “C API Prepared Statement Type Conversions”](#).

- `unsigned long buffer_length`

The actual size of `*buffer` in bytes. This indicates the maximum amount of data that can be stored in the buffer. For character and binary C data, the `buffer_length` value specifies the length of `*buffer` when used with `mysql_stmt_bind_param()` to specify input values, or the maximum number of output data bytes that can be fetched into the buffer when used with `mysql_stmt_bind_result()`.

- `unsigned long *length`

A pointer to an `unsigned long` variable that indicates the actual number of bytes of data stored in `*buffer`. `length` is used for character or binary C data.

For input parameter data binding, set `*length` to indicate the actual length of the parameter value stored in `*buffer`. This is used by `mysql_stmt_execute()`.

For output value binding, MySQL sets `*length` when you call `mysql_stmt_fetch()`. The `mysql_stmt_fetch()` return value determines how to interpret the length:

- If the return value is 0, `*length` indicates the actual length of the parameter value.
- If the return value is `MYSQL_DATA_TRUNCATED`, `*length` indicates the nontruncated length of the parameter value. In this case, the minimum of `*length` and `buffer_length` indicates the actual length of the value.

`length` is ignored for numeric and temporal data types because the `buffer_type` value determines the length of the data value.

If you must determine the length of a returned value before fetching it, see [Section 22.8.7.11](#), “`mysql_stmt_fetch()`”, for some strategies.

- `my_bool *is_null`

This member points to a `my_bool` variable that is true if a value is `NULL`, false if it is not `NULL`. For input, set `*is_null` to true to indicate that you are passing a `NULL` value as a statement parameter.

`is_null` is a *pointer* to a boolean scalar, not a boolean scalar, to provide flexibility in how you specify `NULL` values:

- If your data values are always `NULL`, use `MYSQL_TYPE_NULL` as the `buffer_type` value when you bind the column. The other `MYSQL_BIND` members, including `is_null`, do not matter.
- If your data values are always `NOT NULL`, set `is_null = (my_bool*) 0`, and set the other members appropriately for the variable you are binding.
- In all other cases, set the other members appropriately and set `is_null` to the address of a `my_bool` variable. Set that variable's value to true or false appropriately between executions to indicate whether the corresponding data value is `NULL` or `NOT NULL`, respectively.

For output, when you fetch a row, MySQL sets the value pointed to by `is_null` to true or false according to whether the result set column value returned from the statement is or is not `NULL`.

- `my_bool is_unsigned`

This member applies for C variables with data types that can be `unsigned` (`char`, `short int`, `int`, `long long int`). Set `is_unsigned` to true if the variable pointed to by `buffer` is `unsigned` and false otherwise. For example, if you bind a `signed char` variable to `buffer`, specify a type code of `MYSQL_TYPE_TINY` and set `is_unsigned` to false. If you bind an `unsigned char` instead, the type code is the same but `is_unsigned` should be true. (For `char`, it is not defined whether it is signed or unsigned, so it is best to be explicit about signedness by using `signed char` or `unsigned char`.)

`is_unsigned` applies only to the C language variable on the client side. It indicates nothing about the signedness of the corresponding SQL value on the server side. For example, if you use an `int` variable to supply a value for a `BIGINT UNSIGNED` column, `is_unsigned` should be false because `int` is a signed type. If you use an `unsigned int` variable to supply a value for a `BIGINT` column, `is_unsigned` should be true because `unsigned int` is an unsigned type. MySQL performs the proper conversion between signed and unsigned values in both directions, although a warning occurs if truncation results.

- `my_bool *error`

For output, set this member to point to a `my_bool` variable to have truncation information for the parameter stored there after a row fetching operation. When truncation reporting is enabled, `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED` and `*error` is true in the `MYSQL_BIND` structures for parameters in which truncation occurred. Truncation indicates loss of sign or significant digits, or that a string was too long to fit in a column. Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.

- `MYSQL_TIME`

This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. Set the `buffer` member to point to a `MYSQL_TIME` structure, and set the

`buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types (`MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATE`, `MYSQL_TYPE_DATETIME`, `MYSQL_TYPE_TIMESTAMP`).

The `MYSQL_TIME` structure contains the members listed in the following table.

Member	Description
<code>unsigned int year</code>	The year
<code>unsigned int month</code>	The month of the year
<code>unsigned int day</code>	The day of the month
<code>unsigned int hour</code>	The hour of the day
<code>unsigned int minute</code>	The minute of the hour
<code>unsigned int second</code>	The second of the minute
<code>my_bool neg</code>	A boolean flag indicating whether the time is negative
<code>unsigned long second_part</code>	The fractional part of the second in microseconds; currently unused

Only those parts of a `MYSQL_TIME` structure that apply to a given type of temporal value are used. The `year`, `month`, and `day` elements are used for `DATE`, `DATETIME`, and `TIMESTAMP` values. The `hour`, `minute`, and `second` elements are used for `TIME`, `DATETIME`, and `TIMESTAMP` values. See [Section 22.8.15, “C API Prepared Statement Handling of Date and Time Values”](#).

22.8.5.1. C API Prepared Statement Type Codes

The `buffer_type` member of `MYSQL_BIND` structures indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored.

The following table shows the permissible values for the `buffer_type` member of `MYSQL_BIND` structures for input values sent to the server. The table shows the C variable types that you can use, the corresponding type codes, and the SQL data types for which the supplied value can be used without conversion. Choose the `buffer_type` value according to the data type of the C language variable that you are binding. For the integer types, you should also set the `is_unsigned` member to indicate whether the variable is signed or unsigned.

Input Variable C Type	<code>buffer_type</code> Value	SQL Type of Destination Value
<code>signed char</code>	<code>MYSQL_TYPE_TINY</code>	<code>TINYINT</code>
<code>short int</code>	<code>MYSQL_TYPE_SHORT</code>	<code>SMALLINT</code>
<code>int</code>	<code>MYSQL_TYPE_LONG</code>	<code>INT</code>
<code>long long int</code>	<code>MYSQL_TYPE_LONGLONG</code>	<code>BIGINT</code>
<code>float</code>	<code>MYSQL_TYPE_FLOAT</code>	<code>FLOAT</code>
<code>double</code>	<code>MYSQL_TYPE_DOUBLE</code>	<code>DOUBLE</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_TIME</code>	<code>TIME</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_DATE</code>	<code>DATE</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_DATETIME</code>	<code>DATETIME</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_TIMESTAMP</code>	<code>TIMESTAMP</code>
<code>char[]</code>	<code>MYSQL_TYPE_STRING</code>	<code>TEXT</code> , <code>CHAR</code> , <code>VARCHAR</code>

Input Variable C Type	buffer_type Value	SQL Type of Destination Value
char[]	MYSQL_TYPE_BLOB	BLOB, BINARY, VARBINARY
	MYSQL_TYPE_NULL	NULL

Use `MYSQL_TYPE_NULL` as indicated in the description for the `is_null` member in [Section 22.8.5, “C API Prepared Statement Data Structures”](#).

For input string data, use `MYSQL_TYPE_STRING` or `MYSQL_TYPE_BLOB` depending on whether the value is a character (nonbinary) or binary string:

- `MYSQL_TYPE_STRING` indicates character input string data. The value is assumed to be in the character set indicated by the `character_set_client` [486] system variable. If the server stores the value into a column with a different character set, it converts the value to that character set.
- `MYSQL_TYPE_BLOB` indicates binary input string data. The value is treated as having the `binary` character set. That is, it is treated as a byte string and no conversion occurs.

The following table shows the permissible values for the `buffer_type` member of `MYSQL_BIND` structures for output values received from the server. The table shows the SQL types of received values, the corresponding type codes that such values have in result set metadata, and the recommended C language data types to bind to the `MYSQL_BIND` structure to receive the SQL values without conversion. Choose the `buffer_type` value according to the data type of the C language variable that you are binding. For the integer types, you should also set the `is_unsigned` member to indicate whether the variable is signed or unsigned.

SQL Type of Received Value	buffer_type Value	Output Variable C Type
TINYINT	MYSQL_TYPE_TINY	signed char
SMALLINT	MYSQL_TYPE_SHORT	short int
MEDIUMINT	MYSQL_TYPE_INT24	int
INT	MYSQL_TYPE_LONG	int
BIGINT	MYSQL_TYPE_LONGLONG	long long int
FLOAT	MYSQL_TYPE_FLOAT	float
DOUBLE	MYSQL_TYPE_DOUBLE	double
DECIMAL	MYSQL_TYPE_NEWDECIMAL	char[]
YEAR	MYSQL_TYPE_SHORT	short int
TIME	MYSQL_TYPE_TIME	MYSQL_TIME
DATE	MYSQL_TYPE_DATE	MYSQL_TIME
DATETIME	MYSQL_TYPE_DATETIME	MYSQL_TIME
TIMESTAMP	MYSQL_TYPE_TIMESTAMP	MYSQL_TIME
CHAR, BINARY	MYSQL_TYPE_STRING	char[]
VARCHAR, VARBINARY	MYSQL_TYPE_VAR_STRING	char[]
TINYBLOB, TINYTEXT	MYSQL_TYPE_TINY_BLOB	char[]
BLOB, TEXT	MYSQL_TYPE_BLOB	char[]
MEDIUMBLOB, MEDIUMTEXT	MYSQL_TYPE_MEDIUM_BLOB	char[]
LONGBLOB, LONGTEXT	MYSQL_TYPE_LONG_BLOB	char[]
BIT	MYSQL_TYPE_BIT	char[]

22.8.5.2. C API Prepared Statement Type Conversions

Prepared statements transmit data between the client and server using C language variables on the client side that correspond to SQL values on the server side. If there is a mismatch between the C variable type on the client side and the corresponding SQL value type on the server side, MySQL performs implicit type conversions in both directions.

MySQL knows the type code for the SQL value on the server side. The `buffer_type` value in the `MYSQL_BIND` structure indicates the type code of the C variable that holds the value on the client side. The two codes together tell MySQL what conversion must be performed, if any. Here are some examples:

- If you use `MYSQL_TYPE_LONG` with an `int` variable to pass an integer value to the server that is to be stored into a `FLOAT` column, MySQL converts the value to floating-point format before storing it.
- If you fetch an SQL `MEDIUMINT` column value, but specify a `buffer_type` value of `MYSQL_TYPE_LONGLONG` and use a C variable of type `long long int` as the destination buffer, MySQL converts the `MEDIUMINT` value (which requires less than 8 bytes) for storage into the `long long int` (an 8-byte variable).
- If you fetch a numeric column with a value of 255 into a `char[4]` character array and specify a `buffer_type` value of `MYSQL_TYPE_STRING`, the resulting value in the array is a 4-byte string `'255\0'`.
- MySQL returns `DECIMAL` values as the string representation of the original server-side value, which is why the corresponding C type is `char[]`. For example, `12.345` is returned to the client as `'12.345'`. If you specify `MYSQL_TYPE_NEWDECIMAL` and bind a string buffer to the `MYSQL_BIND` structure, `mysql_stmt_fetch()` stores the value in the buffer as a string without conversion. If instead you specify a numeric variable and type code, `mysql_stmt_fetch()` converts the string-format `DECIMAL` value to numeric form.
- For the `MYSQL_TYPE_BIT` type code, `BIT` values are returned into a string buffer, which is why the corresponding C type is `char[]`. The value represents a bit string that requires interpretation on the client side. To return the value as a type that is easier to deal with, you can cause the value to be cast to integer using either of the following types of expressions:

```
SELECT bit_col + 0 FROM t
SELECT CAST(bit_col AS UNSIGNED) FROM t
```

To retrieve the value, bind an integer variable large enough to hold the value and specify the appropriate corresponding integer type code.

Before binding variables to the `MYSQL_BIND` structures that are to be used for fetching column values, you can check the type codes for each column of the result set. This might be desirable if you want to determine which variable types would be best to use to avoid type conversions. To get the type codes, call `mysql_stmt_result_metadata()` after executing the prepared statement with `mysql_stmt_execute()`. The metadata provides access to the type codes for the result set as described in [Section 22.8.7.23, “mysql_stmt_result_metadata\(\)”](#), and [Section 22.8.1, “C API Data Structures”](#).

To determine whether output string values in a result set returned from the server contain binary or nonbinary data, check whether the `charsetnr` value of the result set metadata is 63 (see [Section 22.8.1, “C API Data Structures”](#)). If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

If you cause the `max_length` member of the `MYSQL_FIELD` column metadata structures to be set (by calling `mysql_stmt_attr_set()`), be aware that the `max_length` values for the result set

indicate the lengths of the longest string representation of the result values, not the lengths of the binary representation. That is, `max_length` does not necessarily correspond to the size of the buffers needed to fetch the values with the binary protocol used for prepared statements. Choose the size of the buffers according to the types of the variables into which you fetch the values. For example, a `TINYINT` column containing the value -128 might have a `max_length` value of 4. But the binary representation of any `TINYINT` value requires only 1 byte for storage, so you can supply a `signed char` variable in which to store the value and set `is_unsigned` to indicate that values are signed.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic reparation of the statement when it is next executed. For more information, see [Section 13.5.4, “Automatic Prepared Statement Reparation”](#).

22.8.6. C API Prepared Statement Function Overview

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See [Section 22.8.7, “C API Prepared Statement Function Descriptions”](#).

Function	Description
<code>mysql_stmt_affected_rows()</code>	Returns the number of rows changed, deleted, or inserted by prepared <code>UPDATE</code> , <code>DELETE</code> , or <code>INSERT</code> statement
<code>mysql_stmt_attr_get()</code>	Gets value of an attribute for a prepared statement
<code>mysql_stmt_attr_set()</code>	Sets an attribute for a prepared statement
<code>mysql_stmt_bind_param()</code>	Associates application data buffers with the parameter markers in a prepared SQL statement
<code>mysql_stmt_bind_result()</code>	Associates application data buffers with columns in a result set
<code>mysql_stmt_close()</code>	Frees memory used by a prepared statement
<code>mysql_stmt_data_seek()</code>	Seeks to an arbitrary row number in a statement result set
<code>mysql_stmt_errno()</code>	Returns the error number for the last statement execution
<code>mysql_stmt_error()</code>	Returns the error message for the last statement execution
<code>mysql_stmt_execute()</code>	Executes a prepared statement
<code>mysql_stmt_fetch()</code>	Fetches the next row of data from a result set and returns data for all bound columns
<code>mysql_stmt_fetch_column()</code>	Fetch data for one column of the current row of a result set
<code>mysql_stmt_field_count()</code>	Returns the number of result columns for the most recent statement
<code>mysql_stmt_free_result()</code>	Free the resources allocated to a statement handle
<code>mysql_stmt_init()</code>	Allocates memory for a <code>MYSQL_STMT</code> structure and initializes it
<code>mysql_stmt_insert_id()</code>	Returns the ID generated for an <code>AUTO_INCREMENT</code> column by a prepared statement
<code>mysql_stmt_next_result()</code>	Returns/initiates the next result in a multiple-result execution
<code>mysql_stmt_num_rows()</code>	Returns the row count from a buffered statement result set
<code>mysql_stmt_param_count()</code>	Returns the number of parameters in a prepared statement
<code>mysql_stmt_param_metadata()</code>	(Return parameter metadata in the form of a result set) Currently, this function does nothing
<code>mysql_stmt_prepare()</code>	Prepares an SQL statement string for execution
<code>mysql_stmt_reset()</code>	Resets the statement buffers in the server

Function	Description
<code>mysql_stmt_result_metadata()</code>	Returns prepared statement metadata in the form of a result set
<code>mysql_stmt_row_seek()</code>	Seeks to a row offset in a statement result set, using value returned from <code>mysql_stmt_row_tell()</code>
<code>mysql_stmt_row_tell()</code>	Returns the statement row cursor position
<code>mysql_stmt_send_long_data()</code>	Sends long data in chunks to server
<code>mysql_stmt_sqlstate()</code>	Returns the SQLSTATE error code for the last statement execution
<code>mysql_stmt_store_result()</code>	Retrieves a complete result set to the client

Call `mysql_stmt_init()` to create a statement handle, then `mysql_stmt_prepare()` to prepare the statement string, `mysql_stmt_bind_param()` to supply the parameter data, and `mysql_stmt_execute()` to execute the statement. You can repeat the `mysql_stmt_execute()` by changing parameter values in the respective buffers supplied through `mysql_stmt_bind_param()`.

You can send text or binary data in chunks to server using `mysql_stmt_send_long_data()`. See [Section 22.8.7.26](#), “`mysql_stmt_send_long_data()`”.

If the statement is a `SELECT` or any other statement that produces a result set, `mysql_stmt_prepare()` also returns the result set metadata information in the form of a `MYSQL_RES` result set through `mysql_stmt_result_metadata()`.

You can supply the result buffers using `mysql_stmt_bind_result()`, so that the `mysql_stmt_fetch()` automatically returns data to these buffers. This is row-by-row fetching.

When statement execution has been completed, close the statement handle using `mysql_stmt_close()` so that all resources associated with it can be freed.

If you obtained a `SELECT` statement's result set metadata by calling `mysql_stmt_result_metadata()`, you should also free the metadata using `mysql_free_result()`.

Execution Steps

To prepare and execute a statement, an application follows these steps:

1. Create a prepared statement handle with `mysql_stmt_init()`. To prepare the statement on the server, call `mysql_stmt_prepare()` and pass it a string containing the SQL statement.
2. If the statement will produce a result set, call `mysql_stmt_result_metadata()` to obtain the result set metadata. This metadata is itself in the form of result set, albeit a separate one from the one that contains the rows returned by the query. The metadata result set indicates how many columns are in the result and contains information about each column.
3. Set the values of any parameters using `mysql_stmt_bind_param()`. All parameters must be set. Otherwise, statement execution returns an error or produces unexpected results.
4. Call `mysql_stmt_execute()` to execute the statement.
5. If the statement produces a result set, bind the data buffers to use for retrieving the row values by calling `mysql_stmt_bind_result()`.
6. Fetch the data into the buffers row by row by calling `mysql_stmt_fetch()` repeatedly until no more rows are found.
7. Repeat steps 3 through 6 as necessary, by changing the parameter values and re-executing the statement.

When `mysql_stmt_prepare()` is called, the MySQL client/server protocol performs these actions:

- The server parses the statement and sends the okay status back to the client by assigning a statement ID. It also sends total number of parameters, a column count, and its metadata if it is a result set oriented statement. All syntax and semantics of the statement are checked by the server during this call.
- The client uses this statement ID for the further operations, so that the server can identify the statement from among its pool of statements.

When `mysql_stmt_execute()` is called, the MySQL client/server protocol performs these actions:

- The client uses the statement handle and sends the parameter data to the server.
- The server identifies the statement using the ID provided by the client, replaces the parameter markers with the newly supplied data, and executes the statement. If the statement produces a result set, the server sends the data back to the client. Otherwise, it sends an okay status and the number of rows changed, deleted, or inserted.

When `mysql_stmt_fetch()` is called, the MySQL client/server protocol performs these actions:

- The client reads the data from the current row of the result set and places it into the application data buffers by doing the necessary conversions. If the application buffer type is same as that of the field type returned from the server, the conversions are straightforward.

If an error occurs, you can get the statement error number, error message, and SQLSTATE code using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

Prepared Statement Logging

For prepared statements that are executed with the `mysql_stmt_prepare()` and `mysql_stmt_execute()` C API functions, the server writes `Prepare` and `Execute` lines to the general query log so that you can tell when statements are prepared and executed.

Suppose that you prepare and execute a statement as follows:

1. Call `mysql_stmt_prepare()` to prepare the statement string `"SELECT ?"`.
2. Call `mysql_stmt_bind_param()` to bind the value `3` to the parameter in the prepared statement.
3. Call `mysql_stmt_execute()` to execute the prepared statement.

As a result of the preceding calls, the server writes the following lines to the general query log:

```
Prepare [1] SELECT ?
Execute [1] SELECT 3
```

Each `Prepare` and `Execute` line in the log is tagged with a `[N]` statement identifier so that you can keep track of which prepared statement is being logged. `N` is a positive integer. If there are multiple prepared statements active simultaneously for the client, `N` may be greater than 1. Each `Execute` lines shows a prepared statement after substitution of data values for `?` parameters.

22.8.7. C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions described in detail in the following sections.

All functions that operate with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.

To create a `MYSQL_STMT` handle, use the `mysql_stmt_init()` function.

22.8.7.1. `mysql_stmt_affected_rows()`

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_affected_rows()` may be called immediately after executing a statement with `mysql_stmt_execute()`. It is like `mysql_affected_rows()` but for prepared statements. For a description of what the affected-rows value returned by this function means, See [Section 22.8.3.1](#), “`mysql_affected_rows()`”.

Errors

None.

Example

See the Example in [Section 22.8.7.10](#), “`mysql_stmt_execute()`”.

22.8.7.2. `mysql_stmt_attr_get()`

```
my_bool mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
void *arg)
```

Description

Can be used to get the current value for a statement attribute.

The `option` argument is the option that you want to get; the `arg` should point to a variable that should contain the option value. If the option is an integer, `arg` should point to the value of the integer.

See [Section 22.8.7.3](#), “`mysql_stmt_attr_set()`”, for a list of options and option types.

Return Values

Zero if successful. Nonzero if `option` is unknown.

Errors

None.

22.8.7.3. `mysql_stmt_attr_set()`

```
my_bool mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
const void *arg)
```

Description

Can be used to affect behavior for a prepared statement. This function may be called multiple times to set several options.

The `option` argument is the option that you want to set. The `arg` argument is the value for the option. `arg` should point to a variable that is set to the desired attribute value. The variable type is as indicated in the following table.

The following table shows the possible `option` values.

Option	Argument Type	Function
<code>STMT_ATTR_UPDATE_MAX_LENGTH</code>	<code>my_bool *</code>	If set to 1, causes <code>mysql_stmt_store_result()</code> to

Option	Argument Type	Function
		update the metadata <code>MYSQL_FIELD->max_length</code> value.
<code>STMT_ATTR_CURSOR_TYPE</code>	unsigned long *	Type of cursor to open for statement when <code>mysql_stmt_execute()</code> is invoked. <code>*arg</code> can be <code>CURSOR_TYPE_NO_CURSOR</code> (the default) or <code>CURSOR_TYPE_READ_ONLY</code> .
<code>STMT_ATTR_PREFETCH_ROWS</code>	unsigned long *	Number of rows to fetch from server at a time when using a cursor. <code>*arg</code> can be in the range from 1 to the maximum value of <code>unsigned long</code> . The default is 1.

If you use the `STMT_ATTR_CURSOR_TYPE` option with `CURSOR_TYPE_READ_ONLY`, a cursor is opened for the statement when you invoke `mysql_stmt_execute()`. If there is already an open cursor from a previous `mysql_stmt_execute()` call, it closes the cursor before opening a new one. `mysql_stmt_reset()` also closes any open cursor before preparing the statement for re-execution. `mysql_stmt_free_result()` closes any open cursor.

If you open a cursor for a prepared statement, `mysql_stmt_store_result()` is unnecessary, because that function causes the result set to be buffered on the client side.

Return Values

Zero if successful. Nonzero if `option` is unknown.

Errors

None.

Example

The following example opens a cursor for a prepared statement and sets the number of rows to fetch at a time to 5:

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type);
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                        (void*) &prefetch_rows);
/* ... check return value ... */
```

22.8.7.4. `mysql_stmt_bind_param()`

```
my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_param()` is used to bind input data for the parameter markers in the SQL statement that was passed to `mysql_stmt_prepare()`. It uses `MYSQL_BIND` structures to supply the data. `bind`

is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each “?” parameter marker that is present in the query.

Suppose that you prepare the following statement:

```
INSERT INTO mytbl VALUES(?,?,?)
```

When you bind the parameters, the array of `MYSQL_BIND` structures must contain three elements, and can be declared like this:

```
MYSQL_BIND bind[3];
```

[Section 22.8.5, “C API Prepared Statement Data Structures”](#), describes the members of each `MYSQL_BIND` element and how they should be set to provide input values.

Return Values

Zero if the bind operation was successful. Nonzero if an error occurred.

Errors

- [CR_UNSUPPORTED_PARAM_TYPE \[4032\]](#)

The conversion is not supported. Possibly the `buffer_type` value is invalid or is not one of the supported types.

- [CR_OUT_OF_MEMORY \[4030\]](#)

Out of memory.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

Example

See the Example in [Section 22.8.7.10, “mysql_stmt_execute\(\)”](#).

22.8.7.5. `mysql_stmt_bind_result()`

```
my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_result()` is used to associate (that is, bind) output columns in the result set to data buffers and length buffers. When `mysql_stmt_fetch()` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified buffers.

All columns must be bound to buffers prior to calling `mysql_stmt_fetch()`. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each column of the result set. If you do not bind columns to `MYSQL_BIND` structures, `mysql_stmt_fetch()` simply ignores the data fetch. The buffers should be large enough to hold the data values, because the protocol doesn't return data values in chunks.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysql_stmt_fetch()` is called. Suppose that an application binds the columns in a result set and calls `mysql_stmt_fetch()`. The client/server protocol returns data in the bound buffers. Then suppose that the application binds the columns to a different set of buffers. The protocol places data into the newly bound buffers when the next call to `mysql_stmt_fetch()` occurs.

To bind a column, an application calls `mysql_stmt_bind_result()` and passes the type, address, and length of the output buffer into which the value should be stored. [Section 22.8.5, “C API Prepared Statement Data Structures”](#), describes the members of each `MYSQL_BIND` element and how they should be set to receive output values.

Return Values

Zero if the bind operation was successful. Nonzero if an error occurred.

Errors

- [CR_UNSUPPORTED_PARAM_TYPE \[4032\]](#)

The conversion is not supported. Possibly the `buffer_type` value is invalid or is not one of the supported types.

- [CR_OUT_OF_MEMORY \[4030\]](#)

Out of memory.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

Example

See the Example in [Section 22.8.7.11, “mysql_stmt_fetch\(\)”](#).

22.8.7.6. `mysql_stmt_close()`

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

Description

Closes the prepared statement. `mysql_stmt_close()` also deallocates the statement handle pointed to by `stmt`.

If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Return Values

Zero if the statement was freed successfully. Nonzero if an error occurred.

Errors

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

Example

See the Example in [Section 22.8.7.10, “mysql_stmt_execute\(\)”](#).

22.8.7.7. `mysql_stmt_data_seek()`

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a statement result set. The `offset` value is a row number and should be in the range from 0 to `mysql_stmt_num_rows(stmt)-1`.

This function requires that the statement result set structure contains the entire result of the last executed query, so `mysql_stmt_data_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

None.

Errors

None.

22.8.7.8. `mysql_stmt_errno()`

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_errno()` returns the error code for the most recently invoked statement API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at [Appendix C, Errors, Error Codes, and Common Problems](#).

Return Values

An error code value. Zero if no error occurred.

Errors

None.

22.8.7.9. `mysql_stmt_error()`

```
const char *mysql_stmt_error(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_error()` returns a null-terminated string containing the error message for the most recently invoked statement API function that can succeed or fail. An empty string (" ") is returned if no error occurred. This means the following two tests are equivalent:

```
if(*mysql_stmt_errno(stmt))
{
    // an error occurred
}

if (mysql_stmt_error(stmt)[0])
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

22.8.7.10. `mysql_stmt_execute()`

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_execute()` executes the prepared query associated with the statement handle. The currently bound parameter marker values are sent to server during this call, and the server replaces the markers with this newly supplied data.

Statement processing following `mysql_stmt_execute()` depends on the type of statement:

- For an `UPDATE`, `DELETE`, or `INSERT`, the number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`.
- For a statement such as `SELECT` that generates a result set, you must call `mysql_stmt_fetch()` to fetch the data prior to calling any other functions that result in query processing. For more information on how to fetch the results, refer to [Section 22.8.7.11, “mysql_stmt_fetch\(\)”](#).

Do not following invocation of `mysql_stmt_execute()` with a call to `mysql_store_result()` or `mysql_use_result()`. Those functions are not intended for processing results from prepared statements.

For statements that generate a result set, you can request that `mysql_stmt_execute()` open a cursor for the statement by calling `mysql_stmt_attr_set()` before executing the statement. If you execute a statement multiple times, `mysql_stmt_execute()` closes any open cursor before opening a new one.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic reparation of the statement when it is next executed. For more information, see [Section 13.5.4, “Automatic Prepared Statement Reparation”](#).

Return Values

Zero if execution was successful. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [4030]
Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY` [4030]
Out of memory.
- `CR_SERVER_GONE_ERROR` [4030]
The MySQL server has gone away.
- `CR_SERVER_LOST` [4030]
The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

Example

The following example demonstrates how to create and populate a table using `mysql_stmt_init()`, `mysql_stmt_prepare()`, `mysql_stmt_param_count()`, `mysql_stmt_bind_param()`, `mysql_stmt_execute()`, and `mysql_stmt_affected_rows()`. The `mysql` variable is assumed to be a valid connection handle. For an example that shows how to retrieve data, see [Section 22.8.7.11](#), "`mysql_stmt_fetch()`".

```
#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(coll INT,\
                                col2 VARCHAR(40),\
                                col3 SMALLINT,\
                                col4 TIMESTAMP)"

#define INSERT_SAMPLE "INSERT INTO \
                        test_table(coll,col2,col3) \
                        VALUES(?,?,?)"

MYSQL_STMT      *stmt;
MYSQL_BIND      bind[3];
my_ulonglong    affected_rows;
int             param_count;
short           small_data;
int             int_data;
char            str_data[STRING_SIZE];
unsigned long   str_length;
my_bool         is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
    fprintf(stderr, " DROP TABLE failed\n");
    fprintf(stderr, " %s\n", mysql_error(mysql));
    exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
    fprintf(stderr, " CREATE TABLE failed\n");
    fprintf(stderr, " %s\n", mysql_error(mysql));
    exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/* sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
    fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
```

```
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
    fprintf(stderr, " invalid parameter count returned by MySQL\n");
    exit(0);
}

/* Bind the data for all 3 parameters */

memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need
   to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
    fprintf(stderr, " mysql_stmt_bind_param() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Specify the data values for the first row */
int_data= 10; /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get the number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
        (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, " invalid affected rows by MySQL\n");
    exit(0);
}
```

```

}

/* Specify data values for second row,
   then re-execute the statement */
int_data= 1000;
strncpy(str_data, "
        The most popular Open Source database",
        STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;      /* smallint */
is_null= 0;          /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
        (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, " invalid affected rows by MySQL\n");
    exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
    fprintf(stderr, " failed while closing the statement\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

```



Note

For complete examples on the use of prepared statement functions, refer to the file [tests/mysql_client_test.c](#). This file can be obtained from a MySQL source distribution or from the Bazaar source repository.

22.8.7.11. [mysql_stmt_fetch\(\)](#)

```
int mysql_stmt_fetch(MYSQL_STMT *stmt)
```

Description

[mysql_stmt_fetch\(\)](#) returns the next row in the result set. It can be called only while the result set exists; that is, after a call to [mysql_stmt_execute\(\)](#) for a statement such as `SELECT` that produces a result set.

[mysql_stmt_fetch\(\)](#) returns row data using the buffers bound by [mysql_stmt_bind_result\(\)](#). It returns the data in those buffers for all the columns in the current row set and the lengths are returned to the `length` pointer. All columns must be bound by the application before it calls [mysql_stmt_fetch\(\)](#).

By default, result sets are fetched unbuffered a row at a time from the server. To buffer the entire result set on the client, call [mysql_stmt_store_result\(\)](#) after binding the data buffers and before calling [mysql_stmt_fetch\(\)](#).

If a fetched data value is a `NULL` value, the `*is_null` value of the corresponding `MYSQL_BIND` structure contains `TRUE` (1). Otherwise, the data and its length are returned in the `*buffer` and `*length` elements based on the buffer type specified by the application. Each numeric and temporal type has a fixed length, as listed in the following table. The length of the string types depends on the length of the actual data value, as indicated by `data_length`.

Type	Length
<code>MYSQL_TYPE_TINY</code>	1
<code>MYSQL_TYPE_SHORT</code>	2
<code>MYSQL_TYPE_LONG</code>	4
<code>MYSQL_TYPE_LONGLONG</code>	8
<code>MYSQL_TYPE_FLOAT</code>	4
<code>MYSQL_TYPE_DOUBLE</code>	8
<code>MYSQL_TYPE_TIME</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_DATE</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_DATETIME</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_STRING</code>	<code>data length</code>
<code>MYSQL_TYPE_BLOB</code>	<code>data_length</code>

In some cases you might want to determine the length of a column value before fetching it with `mysql_stmt_fetch()`. For example, the value might be a long string or `BLOB` value for which you want to know how much space must be allocated. To accomplish this, you can use these strategies:

- Before invoking `mysql_stmt_fetch()` to retrieve individual rows, pass `STMT_ATTR_UPDATE_MAX_LENGTH` to `mysql_stmt_attr_set()`, then invoke `mysql_stmt_store_result()` to buffer the entire result on the client side. Setting the `STMT_ATTR_UPDATE_MAX_LENGTH` attribute causes the maximal length of column values to be indicated by the `max_length` member of the result set metadata returned by `mysql_stmt_result_metadata()`.
- Invoke `mysql_stmt_fetch()` with a zero-length buffer for the column in question and a pointer in which the real length can be stored. Then use the real length with `mysql_stmt_fetch_column()`.

```
real_length= 0;

bind[0].buffer= 0;
bind[0].buffer_length= 0;
bind[0].length= &real_length
mysql_stmt_bind_result(stmt, bind);

mysql_stmt_fetch(stmt);
if (real_length > 0)
{
    data= malloc(real_length);
    bind[0].buffer= data;
    bind[0].buffer_length= real_length;
    mysql_stmt_fetch_column(stmt, bind, 0, 0);
}
```

Return Values

Return Value	Description
0	Successful, the data has been fetched to application data buffers.

Return Value	Description
1	Error occurred. Error code and message can be obtained by calling <code>mysql_stmt_errno()</code> and <code>mysql_stmt_error()</code> .
<code>MYSQL_NO_DATA</code>	No more rows/data exists
<code>MYSQL_DATA_TRUNCATED</code>	Data truncation occurred

`MYSQL_DATA_TRUNCATED` is returned when truncation reporting is enabled. To determine which column values were truncated when this value is returned, check the `error` members of the `MYSQL_BIND` structures used for fetching values. Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [4030]
Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY` [4030]
Out of memory.
- `CR_SERVER_GONE_ERROR` [4030]
The MySQL server has gone away.
- `CR_SERVER_LOST` [4030]
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR` [4029]
An unknown error occurred.
- `CR_UNSUPPORTED_PARAM_TYPE` [4032]
The buffer type is `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, but the data type is not `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP`.
- All other unsupported conversion errors are returned from `mysql_stmt_bind_result()`.

Example

The following example demonstrates how to fetch data from a table using `mysql_stmt_result_metadata()`, `mysql_stmt_bind_result()`, and `mysql_stmt_fetch()`. (This example expects to retrieve the two rows inserted by the example shown in [Section 22.8.7.10](#), “`mysql_stmt_execute()`”). The `mysql` variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 \
                      FROM test_table"

MYSQL_STMT      *stmt;
MYSQL_BIND      bind[4];
MYSQL_RES       *prepare_meta_result;
MYSQL_TIME      ts;
unsigned long   length[4];
int             param_count, column_count, row_count;
short          small_data;
```

```
int          int_data;
char         str_data[STRING_SIZE];
my_bool     is_null[4];
my_bool     error[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
    fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
    fprintf(stderr, " invalid parameter count returned by MySQL\n");
    exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
    fprintf(stderr,
            " mysql_stmt_result_metadata(), \
            returned no meta information\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout,
        " total columns in SELECT statement: %d\n",
        column_count);

if (column_count != 4) /* validate column count */
{
    fprintf(stderr, " invalid column count returned by MySQL\n");
    exit(0);
}

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute(), failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */
memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
```

```
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];
bind[0].error= &error[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];
bind[1].error= &error[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];
bind[2].error= &error[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];
bind[3].error= &error[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
    fprintf(stderr, "mysql_stmt_bind_result() failed\n");
    fprintf(stderr, "%s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Now buffer all results to client (optional step) */
if (mysql_stmt_store_result(stmt))
{
    fprintf(stderr, "mysql_stmt_store_result() failed\n");
    fprintf(stderr, "%s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
    row_count++;
    fprintf(stdout, " row %d\n", row_count);

    /* column 1 */
    fprintf(stdout, " column1 (integer) : ");
    if (is_null[0])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

    /* column 2 */
    fprintf(stdout, " column2 (string) : ");
    if (is_null[1])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

    /* column 3 */
    fprintf(stdout, " column3 (smallint) : ");
```

```

if (is_null[2])
    fprintf(stdout, " NULL\n");
else
    fprintf(stdout, "%d(%ld)\n", small_data, length[2]);

/* column 4 */
fprintf(stdout, "    column4 (timestamp): ");
if (is_null[3])
    fprintf(stdout, " NULL\n");
else
    fprintf(stdout, "%04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
              ts.year, ts.month, ts.day,
              ts.hour, ts.minute, ts.second,
              length[3]);
fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
    fprintf(stderr, " MySQL failed to return all rows\n");
    exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);

/* Close the statement */
if (mysql_stmt_close(stmt))
{
    fprintf(stderr, " failed while closing the statement\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

```

22.8.7.12. `mysql_stmt_fetch_column()`

```
int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int
column, unsigned long offset)
```

Description

Fetch one column from the current result set row. `bind` provides the buffer where data should be placed. It should be set up the same way as for `mysql_stmt_bind_result()`. `column` indicates which column to fetch. The first column is numbered 0. `offset` is the offset within the data value at which to begin retrieving data. This can be used for fetching the data value in pieces. The beginning of the value is offset 0.

Return Values

Zero if the value was fetched successfully. Nonzero if an error occurred.

Errors

- [CR_INVALID_PARAMETER_NO \[4032\]](#)
Invalid column number.
- [CR_NO_DATA \[4033\]](#)
The end of the result set has already been reached.

22.8.7.13. `mysql_stmt_field_count()`

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

Description

Returns the number of columns for the most recent statement for the statement handler. This value is zero for statements such as `INSERT` or `DELETE` that do not produce result sets.

`mysql_stmt_field_count()` can be called after you have prepared a statement by invoking `mysql_stmt_prepare()`.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

22.8.7.14. `mysql_stmt_free_result()`

```
my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

Description

Releases memory associated with the result set produced by execution of the prepared statement. If there is a cursor open for the statement, `mysql_stmt_free_result()` closes it.

Return Values

Zero if the result set was freed successfully. Nonzero if an error occurred.

Errors

22.8.7.15. `mysql_stmt_init()`

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

Description

Create a `MYSQL_STMT` handle. The handle should be freed with `mysql_stmt_close(MYSQL_STMT *)`.

See also [Section 22.8.5, “C API Prepared Statement Data Structures”](#), for more information.

Return Values

A pointer to a `MYSQL_STMT` structure in case of success. `NULL` if out of memory.

Errors

- `CR_OUT_OF_MEMORY` [4030]

Out of memory.

22.8.7.16. `mysql_stmt_insert_id()`

```
my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the prepared `INSERT` or `UPDATE` statement. Use this function after you have executed a prepared `INSERT` statement on a table which contains an `AUTO_INCREMENT` field.

See [Section 22.8.3.37](#), “`mysql_insert_id()`”, for more information.

Return Values

Value for `AUTO_INCREMENT` column which was automatically generated or explicitly set during execution of prepared statement, or value generated by `LAST_INSERT_ID(expr)` [1194] function. Return value is undefined if statement does not set `AUTO_INCREMENT` value.

Errors

None.

22.8.7.17. `mysql_stmt_next_result()`

```
int mysql_stmt_next_result(MYSQL_STMT *mysql)
```

Description

This function is used when you use prepared `CALL` statements to execute stored procedures, which can return multiple result sets. Use a loop that calls `mysql_stmt_next_result()` to determine whether there are more results. If a procedure has `OUT` or `INOUT` parameters, their values will be returned as a single-row result set following any other result sets. The values will appear in the order in which they are declared in the procedure parameter list.

`mysql_stmt_next_result()` returns a status to indicate whether more results exist. If `mysql_stmt_next_result()` returns an error, there are no more results.

Before each call to `mysql_stmt_next_result()`, you must call `mysql_stmt_free_result()` for the current result if it produced a result set (rather than just a result status).

After calling `mysql_stmt_next_result()` the state of the connection is as if you had called `mysql_stmt_execute()`. This means that you can call `mysql_stmt_bind_result()`, `mysql_stmt_affected_rows()`, and so forth.

It is also possible to test whether there are more results by calling `mysql_more_results()`. However, this function does not change the connection state, so if it returns true, you must still call `mysql_stmt_next_result()` to advance to the next result.

For an example that shows how to use `mysql_stmt_next_result()`, see [Section 22.8.16](#), “C API Support for Prepared `CALL` Statements”.

`mysql_stmt_next_result()` was added in MySQL 5.5.3.

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results

Return Value	Description
>0	An error occurred

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

22.8.7.18. `mysql_stmt_num_rows()`

```
my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

Description

Returns the number of rows in the result set.

The use of `mysql_stmt_num_rows()` depends on whether you used `mysql_stmt_store_result()` to buffer the entire result set in the statement handle. If you use `mysql_stmt_store_result()`, `mysql_stmt_num_rows()` may be called immediately. Otherwise, the row count is unavailable unless you count the rows as you fetch them.

`mysql_stmt_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_stmt_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

22.8.7.19. `mysql_stmt_param_count()`

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

Description

Returns the number of parameter markers present in the prepared statement.

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

See the Example in [Section 22.8.7.10](#), “`mysql_stmt_execute()`”.

22.8.7.20. `mysql_stmt_param_metadata()`

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

This function currently does nothing.

Description

Return Values

Errors

22.8.7.21. `mysql_stmt_prepare()`

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *stmt_str, unsigned long length)
```

Description

Given the statement handle returned by `mysql_stmt_init()`, prepares the SQL statement pointed to by the string `stmt_str` and returns a status value. The string length should be given by the `length` argument. The string must consist of a single SQL statement. You should not add a terminating semicolon (“;”) or `\g` to the statement.

The application can include one or more parameter markers in the SQL statement by embedding question mark (“?”) characters into the SQL string at the appropriate positions.

The markers are legal only in certain places in SQL statements. For example, they are permitted in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value. However, they are not permitted for identifiers (such as table or column names), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

The parameter markers must be bound to application variables using `mysql_stmt_bind_param()` before executing the statement.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic reparation of the statement when it is next executed. For more information, see [Section 13.5.4](#), “Automatic Prepared Statement Repreparation”.

Return Values

Zero if the statement was prepared successfully. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [4030]

Commands were executed in an improper order.

- [CR_OUT_OF_MEMORY \[4030\]](#)

Out of memory.

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

If the prepare operation was unsuccessful (that is, `mysql_stmt_prepare()` returns nonzero), the error message can be obtained by calling `mysql_stmt_error()`.

Example

See the Example in [Section 22.8.7.10](#), “`mysql_stmt_execute()`”.

22.8.7.22. `mysql_stmt_reset()`

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Resets a prepared statement on client and server to state after prepare. It resets the statement on the server, data sent using `mysql_stmt_send_long_data()`, unbuffered result sets and current errors. It does not clear bindings or stored result sets. Stored result sets will be cleared when executing the prepared statement (or closing it).

To re-prepare the statement with another query, use `mysql_stmt_prepare()`.

Return Values

Zero if the statement was reset successfully. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR \[4030\]](#)

The MySQL server has gone away.

- [CR_SERVER_LOST \[4030\]](#)

The connection to the server was lost during the query

- [CR_UNKNOWN_ERROR \[4029\]](#)

An unknown error occurred.

22.8.7.23. `mysql_stmt_result_metadata()`

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```

Description

If a statement passed to `mysql_stmt_prepare()` is one that produces a result set, `mysql_stmt_result_metadata()` returns the result set metadata in the form of a pointer to a `MYSQL_RES` structure that can be used to process the meta information such as number of fields and individual field information. This result set pointer can be passed as an argument to any of the field-based API functions that process result set metadata, such as:

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`
- `mysql_field_seek()`
- `mysql_field_tell()`
- `mysql_free_result()`

The result set structure should be freed when you are done with it, which you can do by passing it to `mysql_free_result()`. This is similar to the way you free a result set obtained from a call to `mysql_store_result()`.

The result set returned by `mysql_stmt_result_metadata()` contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with `mysql_stmt_fetch()`.

Return Values

A `MYSQL_RES` result structure. `NULL` if no meta information exists for the prepared query.

Errors

- `CR_OUT_OF_MEMORY` [4030]
Out of memory.
- `CR_UNKNOWN_ERROR` [4029]
An unknown error occurred.

Example

See the Example in [Section 22.8.7.11](#), “`mysql_stmt_fetch()`”.

22.8.7.24. `mysql_stmt_row_seek()`

```
MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a statement result set. The `offset` value is a row offset that should be a value returned from `mysql_stmt_row_tell()` or from `mysql_stmt_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_stmt_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_stmt_row_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_stmt_row_seek()`.

Errors

None.

22.8.7.25. `mysql_stmt_row_tell()`

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

Description

Returns the current position of the row cursor for the last `mysql_stmt_fetch()`. This value can be used as an argument to `mysql_stmt_row_seek()`.

You should use `mysql_stmt_row_tell()` only after `mysql_stmt_store_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

22.8.7.26. `mysql_stmt_send_long_data()`

```
my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int  
parameter_number, const char *data, unsigned long length)
```

Description

Enables an application to send parameter data to the server in pieces (or “chunks”). Call this function after `mysql_stmt_bind_param()` and before `mysql_stmt_execute()`. It can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the `TEXT` or `BLOB` data types.

`parameter_number` indicates which parameter to associate the data with. Parameters are numbered beginning with 0. `data` is a pointer to a buffer containing data to be sent, and `length` indicates the number of bytes in the buffer.

**Note**

The next `mysql_stmt_execute()` call ignores the bind buffer for all parameters that have been used with `mysql_stmt_send_long_data()` since last `mysql_stmt_execute()` or `mysql_stmt_reset()`.

If you want to reset/forget the sent data, you can do it with `mysql_stmt_reset()`. See [Section 22.8.7.22](#), “`mysql_stmt_reset()`”.

As of MySQL 5.5.11, the `max_long_data_size` [523] system variable controls the maximum size of parameter values that can be sent with `mysql_stmt_send_long_data()`. If this variable not set at server startup, the default is the value of the `max_allowed_packet` [519] system variable. `max_long_data_size` [523] is deprecated. In MySQL 5.6, it is removed and the maximum parameter size is controlled by `max_allowed_packet` [519].

Return Values

Zero if the data is sent successfully to server. Nonzero if an error occurred.

Errors

- `CR_INVALID_BUFFER_USE` [4032]

The parameter does not have a string or binary type.

- `CR_COMMANDS_OUT_OF_SYNC` [4030]

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [4030]

The MySQL server has gone away.

- `CR_OUT_OF_MEMORY` [4030]

Out of memory.

- `CR_UNKNOWN_ERROR` [4029]

An unknown error occurred.

Example

The following example demonstrates how to send the data for a `TEXT` column in chunks. It inserts the data value `'MySQL - The most popular Open Source database'` into the `text_column` column. The `mysql` variable is assumed to be a valid connection handle.

```
#define INSERT_QUERY "INSERT INTO \
                    test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long        length;

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
```

```

}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}
memset(bind, 0, sizeof(bind));
bind[0].buffer_type= MYSQL_TYPE_STRING;
bind[0].length= &length;
bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
    fprintf(stderr, "\n param bind failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Supply data in chunks to server */
if (mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
    fprintf(stderr, "\n send_long_data failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Supply the next piece of data */
if (mysql_stmt_send_long_data(stmt,0,
    " - The most popular Open Source database",40))
{
    fprintf(stderr, "\n send_long_data failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Now, execute the query */
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, "\n mysql_stmt_execute failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

```

22.8.7.27. `mysql_stmt_sqlstate()`

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_sqlstate()` returns a null-terminated string containing the SQLSTATE error code for the most recently invoked prepared statement API function that can succeed or fail. The error code consists of five characters. "00000" means "no error." The values are specified by ANSI SQL and ODBC. For a list of possible values, see [Appendix C, Errors, Error Codes, and Common Problems](#).

Note that not all MySQL errors are yet mapped to SQLSTATE codes. The value "HY000" (general error) is used for unmapped errors.

Return Values

A null-terminated character string containing the SQLSTATE error code.

22.8.7.28. `mysql_stmt_store_result()`

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Description

Result sets are produced by calling `mysql_stmt_execute()` to executed prepared statements for SQL statements such as `SELECT`, `SHOW`, `DESCRIBE`, and `EXPLAIN`. By default, result sets for successfully executed prepared statements are not buffered on the client and `mysql_stmt_fetch()` fetches them one at a time from the server. To cause the complete result set to be buffered on the client, call `mysql_stmt_store_result()` after binding data buffers with `mysql_stmt_bind_result()` and before calling `mysql_stmt_fetch()` to fetch rows. (For an example, see [Section 22.8.7.11](#), “`mysql_stmt_fetch()`”.)

`mysql_stmt_store_result()` is optional for result set processing, unless you will call `mysql_stmt_data_seek()`, `mysql_stmt_row_seek()`, or `mysql_stmt_row_tell()`. Those functions require a seekable result set.

It is unnecessary to call `mysql_stmt_store_result()` after executing an SQL statement that does not produce a result set, but if you do, it does not harm or cause any notable performance problem. You can detect whether the statement produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to [Section 22.8.7.23](#), “`mysql_stmt_result_metadata()`”.



Note

MySQL doesn't by default calculate `MYSQL_FIELD->max_length` for all columns in `mysql_stmt_store_result()` because calculating this would slow down `mysql_stmt_store_result()` considerably and most applications don't need `max_length`. If you want `max_length` to be updated, you can call `mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)` to enable this. See [Section 22.8.7.3](#), “`mysql_stmt_attr_set()`”.

Return Values

Zero if the results are buffered successfully. Nonzero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC \[4030\]](#)
Commands were executed in an improper order.
- [CR_OUT_OF_MEMORY \[4030\]](#)
Out of memory.
- [CR_SERVER_GONE_ERROR \[4030\]](#)
The MySQL server has gone away.
- [CR_SERVER_LOST \[4030\]](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR \[4029\]](#)
An unknown error occurred.

22.8.8. C API Threaded Function Descriptions

To create a threaded client, use the functions described in the following sections. See also [Section 22.8.17.2, "How to Write a Threaded Client"](#).

22.8.8.1. `my_init()`

```
void my_init(void)
```

Description

`my_init()` initializes some global variables that MySQL needs. It also calls `mysql_thread_init()` for this thread.

It is necessary for `my_init()` to be called early in the initialization phase of a program's use of the MySQL library. However, `my_init()` is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you ensure that your program invokes one of those functions before any other MySQL calls, there is no need to invoke `my_init()` explicitly.

To access the prototype for `my_init()`, your program should include these header files:

```
#include <my_global.h>
#include <my_sys.h>
```

Return Values

None.

22.8.8.2. `mysql_thread_end()`

```
void mysql_thread_end(void)
```

Description

This function needs to be called before calling `pthread_exit()` to free memory allocated by `mysql_thread_init()`.

`mysql_thread_end()` *is not invoked automatically by the client library*. It must be called explicitly to avoid a memory leak.

Return Values

None.

22.8.8.3. `mysql_thread_init()`

```
my_bool mysql_thread_init(void)
```

Description

This function must be called early within each created thread to initialize thread-specific variables. However, you may not necessarily need to invoke it explicitly: `mysql_thread_init()` is automatically called by `my_init()`, which itself is automatically called by `mysql_init()`, `mysql_library_init()`,

`mysql_server_init()`, and `mysql_connect()`. If you invoke any of those functions, `mysql_thread_init()` will be called for you.

Return Values

Zero if successful. Nonzero if an error occurred.

22.8.8.4. `mysql_thread_safe()`

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client library is compiled as thread-safe.

Return Values

1 if the client library is thread-safe, 0 otherwise.

22.8.9. C API Embedded Server Function Descriptions

MySQL applications can be written to use an embedded server. See [Section 22.7, “libmysqld, the Embedded MySQL Server Library”](#). To write such an application, you must link it against the `libmysqld` library by using the `-lmysqld` flag rather than linking it against the `libmysqlclient` client library by using the `-libmysqlclient` flag. However, the calls to initialize and finalize the library are the same whether you write a client application or one that uses the embedded server: Call `mysql_library_init()` to initialize the library and `mysql_library_end()` when you are done with it. See [Section 22.8.2, “C API Function Overview”](#).

22.8.9.1. `mysql_server_init()`

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function initializes the MySQL library, which must be done before you call any other MySQL function. However, `mysql_server_init()` is deprecated and you should call `mysql_library_init()` instead. See [Section 22.8.3.40, “mysql_library_init\(\)”](#).

Return Values

Zero if successful. Nonzero if an error occurred.

22.8.9.2. `mysql_server_end()`

```
void mysql_server_end(void)
```

Description

This function finalizes the MySQL library, which should be done when you are done using the library. However, `mysql_server_end()` is deprecated and `mysql_library_end()` should be used instead. See [Section 22.8.3.39, “mysql_library_end\(\)”](#).

Return Values

None.

22.8.10. C API Client Plugin Functions

This section describes functions used for the client-side plugin API. They enable management of client plugins. For a description of the `st_mysql_client_plugin` structure used by these functions, see [Section 23.2.4.2.3, “Client Plugin Descriptors”](#).

It is unlikely that a client program needs to call the functions in this section. For example, a client that supports the use of authentication plugins normally causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

22.8.10.1. `mysql_client_find_plugin()`

```
struct st_mysql_client_plugin *mysql_client_find_plugin(MYSQL *mysql, const
char *name, int type)
```

Description

Returns a pointer to a loaded plugin, loading the plugin first if necessary. An error occurs if the type is invalid or the plugin cannot be found or loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.
- `name`: The plugin name.
- `type`: The plugin type.

This function was added in MySQL 5.5.7.

Return Values

A pointer to the plugin for success. `NULL` if an error occurred.

Errors

To check for errors, call the `mysql_error()` or `mysql_errno()` function. See [Section 22.8.3.15, “`mysql_error\(\)`”](#), and [Section 22.8.3.14, “`mysql_errno\(\)`”](#).

Example

```
MYSQL mysql;
struct st_mysql_client_plugin *p;
```

```

if ((p = mysql_client_find_plugin(&mysql, "myplugin",
                                MYSQL_CLIENT_AUTHENTICATION_PLUGIN, 0)))
{
    printf("Plugin version: %d.%d.%d\n", p->version[0], p->version[1], p->version[2]);
}

```

22.8.10.2. `mysql_client_register_plugin()`

```

struct st_mysql_client_plugin *mysql_client_register_plugin(MYSQL *mysql,
struct st_mysql_client_plugin *plugin)

```

Description

Adds a plugin structure to the list of loaded plugins. An error occurs if the plugin is already loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.
- `plugin`: A pointer to the plugin structure.

This function was added in MySQL 5.5.7.

Return Values

A pointer to the plugin for success. `NULL` if an error occurred.

Errors

To check for errors, call the `mysql_error()` or `mysql_errno()` function. See [Section 22.8.3.15](#), “`mysql_error()`”, and [Section 22.8.3.14](#), “`mysql_errno()`”.

22.8.10.3. `mysql_load_plugin()`

```

struct st_mysql_client_plugin *mysql_load_plugin(MYSQL *mysql, const char
*name, int type, int argc, ...)

```

Description

Loads a MySQL client plugin, specified by name and type. An error occurs if the type is invalid or the plugin cannot be loaded.

It is not possible to load multiple plugins of the same type. An error occurs if you try to load a plugin of a type already loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.
- `name`: The name of the plugin to load.
- `type`: The type of plugin to load, or `-1` to disable type checking. If type is not `-1`, only plugins matching the type are considered for loading.

- `argc`: The number of following arguments (0 if there are none). Interpretation of any following arguments depends on the plugin type.

This function was added in MySQL 5.5.7.

Another way to cause plugins to be loaded is to set the `LIBMYSQL_PLUGINS` environment variable to a semicolon-separated list of plugin names. For example:

```
shell> export LIBMYSQL_PLUGINS="myplugin1;myplugin2"
```

Plugins named by `LIBMYSQL_PLUGINS` are loaded when the client program calls `mysql_library_init()`. No error is reported if problems occur loading these plugins.

Return Values

A pointer to the plugin if it was loaded successfully. `NULL` if an error occurred.

Errors

To check for errors, call the `mysql_error()` or `mysql_errno()` function. See [Section 22.8.3.15](#), “`mysql_error()`”, and [Section 22.8.3.14](#), “`mysql_errno()`”.

Example

```
MYSQL mysql;

if(!mysql_load_plugin(&mysql, "myplugin",
                     MYSQL_CLIENT_AUTHENTICATION_PLUGIN, 0))
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
    exit(-1);
}
```

See Also

See also [Section 22.8.10.3](#), “`mysql_load_plugin()`”, [Section 22.8.3.15](#), “`mysql_error()`”, [Section 22.8.3.14](#), “`mysql_errno()`”.

22.8.10.4. `mysql_load_plugin_v()`

```
struct st_mysql_client_plugin *mysql_load_plugin_v(MYSQL *mysql, const char
*name, int type, int argc, va_list args)
```

Description

This function is equivalent to `mysql_load_plugin()`, but it accepts a `va_list` instead of a variable list of parameters.

This function was added in MySQL 5.5.7.

See Also

See also [Section 22.8.10.3](#), “`mysql_load_plugin()`”.

22.8.10.5. `mysql_plugin_options()`

```
int mysql_plugin_options(struct st_mysql_client_plugin *plugin, const char
*option, const void *value)
```

Description

Passes an option type and value to a plugin. This function can be called multiple times to set several options. If the plugin does not have an option handler, an error occurs.

Specify the parameters as follows:

- `plugin`: A pointer to the plugin structure.
- `option`: The option to be set.
- `value`: A pointer to the option value.

This function was added in MySQL 5.5.7.

Return Values

Zero for success, 1 if an error occurred. If the plugin has an option handler, that handler should also return zero for success and 1 if an error occurred.

22.8.11. Common Questions and Problems When Using the C API

22.8.11.1. Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).
- The data could not be read (an error occurred on the connection).
- The query returned no data (for example, it was an `INSERT`, `UPDATE`, or `DELETE`).

You can always check whether the statement should have produced a nonempty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an `INSERT` or a `DELETE`). If `mysql_field_count()` returns a nonzero value, the statement should have produced a nonempty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

22.8.11.2. What Results You Can Get from a Query

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an `INSERT`, `UPDATE`, or `DELETE`.

For a fast re-create, use `TRUNCATE TABLE`.

- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With

`mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.

- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an `AUTO_INCREMENT` index. See [Section 22.8.3.37](#), “`mysql_insert_id()`”.
- Some queries (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) return additional information. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

22.8.11.3. How to Get the Unique ID for the Last Inserted Row

If you insert a record into a table that contains an `AUTO_INCREMENT` column, you can obtain the value stored into that column by calling the `mysql_insert_id()` function.

You can check from your C applications whether a value was stored in an `AUTO_INCREMENT` column by executing the following code (which assumes that you've checked that the statement succeeded). It determines whether the query was an `INSERT` with an `AUTO_INCREMENT` index:

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

When a new `AUTO_INCREMENT` value has been generated, you can also obtain it by executing a `SELECT LAST_INSERT_ID()` statement with `mysql_query()` and retrieving the value from the result set returned by the statement.

When inserting multiple values, the last automatically incremented value is returned.

For `LAST_INSERT_ID()` [1194], the most recently generated ID is maintained in the server on a per-connection basis. It is not changed by another client. It is not even changed if you update another `AUTO_INCREMENT` column with a nonmagic value (that is, a value that is not `NULL` and not `0`). Using `LAST_INSERT_ID()` [1194] and `AUTO_INCREMENT` columns simultaneously from multiple clients is perfectly valid. Each client will receive the last inserted ID for the last statement *that* client executed.

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
VALUES(NULL,'text');          # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
VALUES(LAST_INSERT_ID(),'text'); # use ID in second table
```

Note that `mysql_insert_id()` returns the value stored into an `AUTO_INCREMENT` column, whether that value is automatically generated by storing `NULL` or `0` or was specified as an explicit value. `LAST_INSERT_ID()` [1194] returns only automatically generated `AUTO_INCREMENT` values. If you store an explicit value other than `NULL` or `0`, it does not affect the value returned by `LAST_INSERT_ID()` [1194].

For more information on obtaining the last ID in an `AUTO_INCREMENT` column:

- For information on `LAST_INSERT_ID()` [1194], which can be used within an SQL statement, see [Section 12.14](#), “Information Functions”.
- For information on `mysql_insert_id()`, the function you use from within the C API, see [Section 22.8.3.37](#), “`mysql_insert_id()`”.

- For information on obtaining the auto-incremented value when using Connector/J, see [Section 22.3.6.4, “Retrieving `AUTO_INCREMENT` Column Values through JDBC”](#).
- For information on obtaining the auto-incremented value when using Connector/ODBC, see [Section 22.1.7.1.1, “Obtaining Auto-Increment Values”](#).

22.8.12. Controlling Automatic Reconnection Behavior

The MySQL client library can perform an automatic reconnection to the server if it finds that the connection is down when you attempt to send a statement to the server to be executed. In this case, the library tries once to reconnect to the server and send the statement again.

In MySQL 5.5, auto-reconnect is disabled by default.

If it is important for your application to know that the connection has been dropped (so that it can exit or take action to adjust for the loss of state information), be sure that auto-reconnect is disabled. To ensure this, call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option:

```
my_bool reconnect = 0;
mysql_options(&mysql, MYSQL_OPT_RECONNECT, &reconnect);
```

If the connection has gone down, the effect of `mysql_ping()` depends on the auto-reconnect state. If auto-reconnect is enabled, `mysql_ping()` performs a reconnect. Otherwise, it returns an error.

Some client programs might provide the capability of controlling automatic reconnection. For example, `mysql` reconnects by default, but the `--skip-reconnect` [281] option can be used to suppress this behavior.

If an automatic reconnection does occur (for example, as a result of calling `mysql_ping()`), there is no explicit indication of it. To check for reconnection, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier has changed.

Automatic reconnection can be convenient because you need not implement your own reconnect code, but if a reconnection does occur, several aspects of the connection state are reset on the server side and your application will not know about it. The connection-related state is affected as follows:

- Any active transactions are rolled back and autocommit mode is reset.
- All table locks are released.
- All `TEMPORARY` tables are closed (and dropped).
- Session variables are reinitialized to the values of the corresponding variables. This also affects variables that are set implicitly by statements such as `SET NAMES`.
- User variable settings are lost.
- Prepared statements are released.
- `HANDLER` variables are closed.
- The value of `LAST_INSERT_ID()` [1194] is reset to 0.
- Locks acquired with `GET_LOCK()` [1199] are released.

If the connection drops, it is possible that the session associated with the connection on the server side will still be running if the server has not yet detected that the client is no longer connected. In this case,

any locks held by the original connection still belong to that session, so you may want to kill it by calling `mysql_kill()`.

22.8.13. C API Support for Multiple Statement Execution

By default, `mysql_query()` and `mysql_real_query()` interpret their statement string argument as a single statement to be executed, and you process the result according to whether the statement produces a result set (a set of rows, as for `SELECT`) or an affected-rows count (as for `INSERT`, `UPDATE`, and so forth).

MySQL 5.5 also supports the execution of a string containing multiple statements separated by semicolon (“;”) characters. This capability is enabled by special options that are specified either when you connect to the server with `mysql_real_connect()` or after connecting by calling `mysql_set_server_option()`.

Executing a multiple-statement string can produce multiple result sets or row-count indicators. Processing these results involves a different approach than for the single-statement case: After handling the result from the first statement, it is necessary to check whether more results exist and process them in turn if so. To support multiple-result processing, the C API includes the `mysql_more_results()` and `mysql_next_result()` functions. These functions are used at the end of a loop that iterates as long as more results are available. *Failure to process the result this way may result in a dropped connection to the server.*

Multiple-result processing also is required if you execute `CALL` statements for stored procedures. Results from a stored procedure have these characteristics:

- Statements within the procedure may produce result sets (for example, if it executes `SELECT` statements). These result sets are returned in the order that they are produced as the procedure executes.

In general, the caller cannot know how many result sets a procedure will return. Procedure execution may depend on loops or conditional statements that cause the execution path to differ from one call to the next. Therefore, you must be prepared to retrieve multiple results.

- The final result from the procedure is a status result that includes no result set. The status indicates whether the procedure succeeded or an error occurred.

The multiple statement and result capabilities can be used only with `mysql_query()` or `mysql_real_query()`. They cannot be used with the prepared statement interface. Prepared statement handles are defined to work only with strings that contain a single statement. See [Section 22.8.4, “C API Prepared Statements”](#).

To enable multiple-statement execution and result processing, the following options may be used:

- The `mysql_real_connect()` function has a `flags` argument for which two option values are relevant:
 - `CLIENT_MULTI_RESULTS` enables the client program to process multiple results. This option *must* be enabled if you execute `CALL` statements for stored procedures that produce result sets. Otherwise, such procedures result in an error `Error 1312 (0A000): PROCEDURE proc_name can't return a result set in the given context`. As of MySQL 5.5.3, `CLIENT_MULTI_RESULTS` is enabled by default.
 - `CLIENT_MULTI_STATEMENTS` enables `mysql_query()` and `mysql_real_query()` to execute statement strings containing multiple statements separated by semicolons. This option also enables `CLIENT_MULTI_RESULTS` implicitly, so a `flags` argument of `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()` is equivalent to an argument of `CLIENT_MULTI_STATEMENTS` |

`CLIENT_MULTI_RESULTS`. That is, `CLIENT_MULTI_STATEMENTS` is sufficient to enable multiple-statement execution and all multiple-result processing.

- After the connection to the server has been established, you can use the `mysql_set_server_option()` function to enable or disable multiple-statement execution by passing it an argument of `MYSQL_OPTION_MULTI_STATEMENTS_ON` or `MYSQL_OPTION_MULTI_STATEMENTS_OFF`. Enabling multiple-statement execution with this function also enables processing of “simple” results for a multiple-statement string where each statement produces a single result, but is *not* sufficient to permit processing of stored procedures that produce result sets.

The following procedure outlines a suggested strategy for handling multiple statements:

1. Pass `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()`, to fully enable multiple-statement execution and multiple-result processing.
2. After calling `mysql_query()` or `mysql_real_query()` and verifying that it succeeds, enter a loop within which you process statement results.
3. For each iteration of the loop, handle the current statement result, retrieving either a result set or an affected-rows count. If an error occurs, exit the loop.
4. At the end of the loop, call `mysql_next_result()` to check whether another result exists and initiate retrieval for it if so. If no more results are available, exit the loop.

One possible implementation of the preceding strategy is shown following. The final part of the loop can be reduced to a simple test of whether `mysql_next_result()` returns nonzero. The code as written distinguishes between no more results and an error, which enables a message to be printed for the latter occurrence.

```

/* connect to server with the CLIENT_MULTI_STATEMENTS option */
if (mysql_real_connect (mysql, host_name, user_name, password,
    db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS) == NULL)
{
    printf("mysql_real_connect() failed\n");
    mysql_close(mysql);
    exit(1);
}

/* execute multiple statements */
status = mysql_query(mysql,
    "DROP TABLE IF EXISTS test_table;\
    CREATE TABLE test_table(id INT);\
    INSERT INTO test_table VALUES(10);\
    UPDATE test_table SET id=20 WHERE id=10;\
    SELECT * FROM test_table;\
    DROP TABLE test_table");

if (status)
{
    printf("Could not execute statement(s)");
    mysql_close(mysql);
    exit(0);
}

/* process each statement result */
do {
    /* did current statement return data? */
    result = mysql_store_result(mysql);
    if (result)
    {
        /* yes; process rows and free the result set */
        process_result_set(mysql, result);
    }
} while (mysql_next_result());

```

```

mysql_free_result(result);
}
else /* no result set or error */
{
    if (mysql_field_count(mysql) == 0)
    {
        printf("%lld rows affected\n",
            mysql_affected_rows(mysql));
    }
    else /* some error occurred */
    {
        printf("Could not retrieve result set\n");
        break;
    }
}
/* more results? -1 = no, >0 = error, 0 = yes (keep looping) */
if ((status = mysql_next_result(mysql)) > 0)
    printf("Could not execute statement\n");
} while (status == 0);

mysql_close(mysql);

```

22.8.14. C API Prepared Statement Problems

Here follows a list of the currently known problems with prepared statements:

- [TIME](#), [TIMESTAMP](#), and [DATETIME](#) do not support parts of seconds (for example, from [DATE_FORMAT\(\)](#) [1138]).
- When converting an integer to string, [ZEROFILL](#) is honored with prepared statements in some cases where the MySQL server doesn't print the leading zeros. (For example, with [MIN\(number-with-zerofill\)](#) [1206]).
- When converting a floating-point number to a string in the client, the rightmost digits of the converted value may differ slightly from those of the original value.
- Prepared statements use the query cache under the conditions described in [Section 8.9.3.1, "How the Query Cache Operates"](#).
- Prepared statements do not support multi-statements (that is, multiple statements within a single string separated by ";" characters).
- Before MySQL 5.5.3, prepared [CALL](#) statements cannot invoke stored procedures that return result sets because prepared statements do not support multiple result sets. Nor can the calling application access a stored procedure's [OUT](#) or [INOUT](#) parameters when the procedure returns. As of MySQL 5.5.3, these capabilities are supported as described in [Section 22.8.16, "C API Support for Prepared CALL Statements"](#).

22.8.15. C API Prepared Statement Handling of Date and Time Values

The binary (prepared statement) protocol enables you to send and receive date and time values ([DATE](#), [TIME](#), [DATETIME](#), and [TIMESTAMP](#)), using the [MYSQL_TIME](#) structure. The members of this structure are described in [Section 22.8.5, "C API Prepared Statement Data Structures"](#).

To send temporal data values, create a prepared statement using [mysql_stmt_prepare\(\)](#). Then, before calling [mysql_stmt_execute\(\)](#) to execute the statement, use the following procedure to set up each temporal parameter:

1. In the [MYSQL_BIND](#) structure associated with the data value, set the [buffer_type](#) member to the type that indicates what kind of temporal value you're sending. For [DATE](#), [TIME](#), [DATETIME](#),

or `TIMESTAMP` values, set `buffer_type` to `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, respectively.

2. Set the `buffer` member of the `MYSQL_BIND` structure to the address of the `MYSQL_TIME` structure in which you pass the temporal value.
3. Fill in the members of the `MYSQL_TIME` structure that are appropriate for the type of temporal value to be passed.

Use `mysql_stmt_bind_param()` to bind the parameter data to the statement. Then you can call `mysql_stmt_execute()`.

To retrieve temporal values, the procedure is similar, except that you set the `buffer_type` member to the type of value you expect to receive, and the `buffer` member to the address of a `MYSQL_TIME` structure into which the returned value should be placed. Use `mysql_stmt_bind_result()` to bind the buffers to the statement after calling `mysql_stmt_execute()` and before fetching the results.

Here is a simple example that inserts `DATE`, `TIME`, and `TIMESTAMP` data. The `mysql` variable is assumed to be a valid connection handle.

```

MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field, \
                timestamp_field) VALUES(?,?,?)");

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query))
{
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* set up input buffers for all 3 parameters */
bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
bind[1]= bind[2]= bind[0];
...

mysql_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..

```

22.8.16. C API Support for Prepared `CALL` Statements

This section describes prepared-statement support in the C API for stored procedures executed using `CALL` statements:

Prior to MySQL 5.5.3, prepared `CALL` statements can be used only for stored procedures that produce at most one result set. Nor can the calling application use placeholders for `OUT` or `INOUT` parameters.

MySQL 5.5.3 expands support for stored procedures executed using prepared `CALL` statements in the following ways:

- A stored procedure can produce any number of result sets. The number of columns and the data types of the columns need not be the same for all result sets.
- The final values of `OUT` and `INOUT` parameters are available to the calling application after the procedure returns. These parameters are returned as an extra single-row result set following any result sets produced by the procedure itself. The row contains the values of the `OUT` and `INOUT` parameters in the order in which they are declared in the procedure parameter list.

The following discussion shows how to use these capabilities through the C API for prepared statements. To use prepared `CALL` statements through the `PREPARE` and `EXECUTE` statements, see [Section 13.2.1, “CALL Syntax”](#).

If an application might be compiled or executed in a context where a version of MySQL older than 5.5.3 is used, prepared `CALL` capabilities for multiple result sets and `OUT` or `INOUT` parameters might not be available:

- For the client side, the application will not compile unless the libraries are from MySQL 5.5.3 or higher (the API function and symbols introduced in that version will not be present).
- To verify at runtime that the server is recent enough, a client can use this test:

```
if (mysql_get_server_version(mysql) < 50503)
{
    fprintf(stderr,
           "Server does not support required CALL capabilities\n");
    mysql_close(mysql);
    exit (1);
}
```

An application that executes a prepared `CALL` statement should use a loop that fetches a result and then invokes `mysql_stmt_next_result()` to determine whether there are more results. The results consist of any result sets produced by the stored procedure followed by a final status value that indicates whether the procedure terminated successfully.

If the procedure has `OUT` or `INOUT` parameters, the result set preceding the final status value contains their values. To determine whether a result set contains parameter values, test whether the `SERVER_PS_OUT_PARAMS` bit is set in the `server_status` member of the `MYSQL` connection handler:

```
mysql->server_status & SERVER_PS_OUT_PARAMS
```

The following example uses a prepared `CALL` statement to execute a stored procedure that produces multiple result sets and that provides parameter values back to the caller by means of `OUT` and `INOUT` parameters. The procedure takes parameters of all three types (`IN`, `OUT`, `INOUT`), displays their initial values, assigns new values, displays the updated values, and returns. The expected return information from the procedure therefore consists of multiple result sets and a final status:

- One result set from a `SELECT` that displays the initial parameter values: `10, NULL, 30`. (The `OUT` parameter is assigned a value by the caller, but this assignment is expected to be ineffective: `OUT` parameters are seen as `NULL` within a procedure until assigned a value within the procedure.)

- One result set from a `SELECT` that displays the modified parameter values: 100, 200, 300.
- One result set containing the final `OUT` and `INOUT` parameter values: 200, 300.
- A final status packet.

The code to execute the procedure:

```
MYSQL_STMT *stmt;
MYSQL_BIND ps_params[3]; /* input parameter buffers */
int int_data[3]; /* input/output values */
my_bool is_null[3]; /* output value nullability */
int status;

/* set up stored procedure */
status = mysql_query(mysql, "DROP PROCEDURE IF EXISTS p1");
test_error(mysql, status);

status = mysql_query(mysql,
    "CREATE PROCEDURE p1("
    " IN p_in INT, "
    " OUT p_out INT, "
    " INOUT p_inout INT) "
    "BEGIN "
    " SELECT p_in, p_out, p_inout; "
    " SET p_in = 100, p_out = 200, p_inout = 300; "
    " SELECT p_in, p_out, p_inout; "
    "END");
test_error(mysql, status);

/* initialize and prepare CALL statement with parameter placeholders */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    printf("Could not initialize statement\n");
    exit(1);
}
status = mysql_stmt_prepare(stmt, "CALL p1(?, ?, ?)", 16);
test_stmt_error(stmt, status);

/* initialize parameters: p_in, p_out, p_inout (all INT) */
memset(ps_params, 0, sizeof (ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_LONG;
ps_params[0].buffer = (char *) &int_data[0];
ps_params[0].length = 0;
ps_params[0].is_null = 0;

ps_params[1].buffer_type = MYSQL_TYPE_LONG;
ps_params[1].buffer = (char *) &int_data[1];
ps_params[1].length = 0;
ps_params[1].is_null = 0;

ps_params[2].buffer_type = MYSQL_TYPE_LONG;
ps_params[2].buffer = (char *) &int_data[2];
ps_params[2].length = 0;
ps_params[2].is_null = 0;

/* bind parameters */
status = mysql_stmt_bind_param(stmt, ps_params);
test_stmt_error(stmt, status);

/* assign values to parameters and execute statement */
int_data[0]= 10; /* p_in */
int_data[1]= 20; /* p_out */
int_data[2]= 30; /* p_inout */
```

```
status = mysql_stmt_execute(stmt);
test_stmt_error(stmt, status);

/* process results until there are no more */
do {
    int i;
    int num_fields;          /* number of columns in result */
    MYSQL_FIELD *fields;    /* for result set metadata */
    MYSQL_BIND *rs_bind;   /* for output buffers */

    /* the column count is > 0 if there is a result set */
    /* 0 if the result is only the final status packet */
    num_fields = mysql_stmt_field_count(stmt);

    if (num_fields > 0)
    {
        /* there is a result set to fetch */
        printf("Number of columns in result: %d\n", (int) num_fields);

        /* what kind of result set is this? */
        printf("Data: ");
        if(mysql->server_status & SERVER_PS_OUT_PARAMS)
            printf("this result set contains OUT/INOUT parameters\n");
        else
            printf("this result set is produced by the procedure\n");

        MYSQL_RES *rs_metadata = mysql_stmt_result_metadata(stmt);
        test_stmt_error(stmt, rs_metadata == NULL);

        fields = mysql_fetch_fields(rs_metadata);

        rs_bind = (MYSQL_BIND *) malloc(sizeof (MYSQL_BIND) * num_fields);
        if (!rs_bind)
        {
            printf("Cannot allocate output buffers\n");
            exit(1);
        }
        memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

        /* set up and bind result set output buffers */
        for (i = 0; i < num_fields; ++i)
        {
            rs_bind[i].buffer_type = fields[i].type;
            rs_bind[i].is_null = &is_null[i];

            switch (fields[i].type)
            {
                case MYSQL_TYPE_LONG:
                    rs_bind[i].buffer = (char *) &(int_data[i]);
                    rs_bind[i].buffer_length = sizeof (int_data);
                    break;

                default:
                    fprintf(stderr, "ERROR: unexpected type: %d.\n", fields[i].type);
                    exit(1);
            }
        }
    }

    status = mysql_stmt_bind_result(stmt, rs_bind);
    test_stmt_error(stmt, status);

    /* fetch and display result set rows */
    while (1)
    {
        status = mysql_stmt_fetch(stmt);

        if (status == 1 || status == MYSQL_NO_DATA)
```

```

        break;

    for (i = 0; i < num_fields; ++i)
    {
        switch (rs_bind[i].buffer_type)
        {
            case MYSQL_TYPE_LONG:
                if (*rs_bind[i].is_null)
                    printf(" val[%d] = NULL;", i);
                else
                    printf(" val[%d] = %ld;",
                            i, (long) *((int *) rs_bind[i].buffer));
                break;

            default:
                printf(" unexpected type (%d)\n",
                        rs_bind[i].buffer_type);
        }
    }
    printf("\n");
}

mysql_free_result(rs_metadata); /* free metadata */
free(rs_bind);                /* free output buffers */
}
else
{
    /* no columns = final status packet */
    printf("End of procedure output\n");
}

/* more results? -1 = no, >0 = error, 0 = yes (keep looking) */
status = mysql_stmt_next_result(stmt);
if (status > 0)
    test_stmt_error(stmt, status);
} while (status == 0);

mysql_stmt_close(stmt);

```

Execution of the procedure should produce the following output:

```

Number of columns in result: 3
Data: this result set is produced by the procedure
  val[0] = 10; val[1] = NULL; val[2] = 30;
Number of columns in result: 3
Data: this result set is produced by the procedure
  val[0] = 100; val[1] = 200; val[2] = 300;
Number of columns in result: 2
Data: this result set contains OUT/INOUT parameters
  val[0] = 200; val[1] = 300;
End of procedure output

```

The code uses two utility routines, `test_error()` `test_stmt_error()`, to check for errors and terminate after printing diagnostic information if an error occurred:

```

static void test_error(MYSQL *mysql, int status)
{
    if (status)
    {
        fprintf(stderr, "Error: %s (errno: %d)\n",
                mysql_error(mysql), mysql_errno(mysql));
        exit(1);
    }
}

static void test_stmt_error(MYSQL_STMT *stmt, int status)
{

```

```

if (status)
{
    fprintf(stderr, "Error: %s (errno: %d)\n",
            mysql_stmt_error(stmt), mysql_stmt_errno(stmt));
    exit(1);
}
}

```

22.8.17. Building Client Programs

If you compile MySQL clients that you've written yourself or that you obtain from a third-party, they must be linked using the `-lmysqlclient -lz` options in the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in `/usr/local/mysql/lib`, use `-L/usr/local/mysql/lib -lmysqlclient -lz` in the link command.

For clients that use MySQL header files, you may need to specify an `-I` option when you compile them (for example, `-I/usr/local/mysql/include`), so that the compiler can find the header files.

Compiling MySQL Clients on Unix

To make it simpler to compile MySQL programs on Unix, we have provided the `mysql_config` script for you. See [Section 4.7.2, “mysql_config — Get Compile Options for Compiling Clients”](#).

You can use it to compile a MySQL client as follows:

```

CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname ` $CFG --cflags ` progname.c ` $CFG --libs `"

```

The `sh -c` is needed to get the shell not to treat the output from `mysql_config` as one word.

Compiling MySQL Clients on Microsoft Windows

On Windows, you can link your code with either the dynamic or static client library. The static library is named `mysqlclient` and the dynamic library is named `libmysql`.

If you link with the static library, failure can occur if certain conditions are not satisfied:

- The client application must be compiled with exactly the same version of Visual Studio as that used to compile the library.
- The client application should link the C runtime statically by using the `/MT` compiler option.

If the client application is built in in debug mode and uses the static debug C runtime (`/MTd` compiler option), it can link the `mysqlclient` if that library was built using the same option. If the client application uses the dynamic C runtime (`/MD` option, or `/MDd` option in debug mode), it cannot link with the static client library and must use the dynamic library.

The MSDN page describing the link options can be found here: <http://msdn.microsoft.com/en-us/library/2kzt1wy3.aspx>

22.8.17.1. Problems Linking to the MySQL Client Library

When linking with the C API, the following errors may occur on some systems:

```

gcc -g -o client test.o -L/usr/local/lib/mysql \
    -lmysqlclient -lsocket -lnsl
Undefined      first referenced

```

```

symbol      in file
floor      /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client

```

If this happens on your system, you must include the math library by adding `-lm` to the end of the compile/link line.

When you are linking an application program to use the MySQL client library, you might get undefined reference errors for symbols that start with `mysql_`, such as those shown here:

```

/tmp/ccFKsdPa.o: In function `main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x57): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'

```

You should be able to solve this problem by adding `-Ldir_path -lmysqlclient` at the end of your link command, where `dir_path` represents the path name of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from `mysql_config` might indicate other libraries that should be specified on the link command as well.

If you get `undefined reference` errors for the `uncompress` or `compress` function, add `-lz` to the end of your link command and try again.

If you get `undefined reference` errors for a function that should exist on your system, such as `connect`, check the manual page for the function in question to determine which libraries you should add to the link command.

You might get `undefined reference` errors such as the following for functions that don't exist on your system:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

This usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours. In this case, you should download the latest MySQL source distribution and compile MySQL yourself. See [Section 2.10, "Installing MySQL from Source"](#).

You might get undefined reference errors at runtime when you try to execute a MySQL program. If these errors specify symbols that start with `mysql_` or indicate that the `mysqlclient` library can't be found, it means that your system can't find the shared `libmysqlclient.so` library. The fix for this is to tell your system to search for shared libraries where the library is located. Use whichever of the following methods is appropriate for your system:

- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY_PATH` environment variable.
- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY` environment variable.
- Copy `libmysqlclient.so` to some directory that is searched by your system, such as `/lib`, and update the shared library information by executing `ldconfig`.

22.8.17.2. How to Write a Threaded Client

The client library is almost thread-safe. The biggest problem is that the subroutines in `net.c` that read from sockets are not interrupt safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server. If you install interrupt handlers for the `SIGPIPE` interrupt, the socket handling should be thread-safe.

To avoid aborting the program when a connection terminates, MySQL blocks `SIGPIPE` on the first call to `mysql_library_init()`, `mysql_init()`, or `mysql_connect()`. If you want to use your own `SIGPIPE` handler, you should first call `mysql_library_init()` and then install your handler.

If “undefined symbol” errors occur when linking against the `libmysqlclient` client library, in most cases this is because you have not included the thread libraries on the link/compile command.

The client library is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Two threads can't send a query to the MySQL server at the same time on the same connection. In particular, you have to ensure that between calls to `mysql_query()` and `mysql_store_result()` no other thread is using the same connection.
- Many threads can access different result sets that are retrieved with `mysql_store_result()`.
- If you use `mysql_use_result()`, you must ensure that no other thread is using the same connection until the result set is closed. However, it really is best for threaded clients that share the same connection to use `mysql_store_result()`.
- If you want to use multiple threads on the same connection, you must have a mutex lock around your pair of `mysql_query()` and `mysql_store_result()` calls. Once `mysql_store_result()` is ready, the lock can be released and other threads may query the same connection.
- If you use POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

You need to know the following if you have a thread that did not create the connection to the MySQL database but is calling MySQL functions:

When you call `mysql_init()`, MySQL creates a thread-specific variable for the thread that is used by the debug library (among other things). If you call a MySQL function before the thread has called `mysql_init()`, the thread does not have the necessary thread-specific variables in place and you are likely to end up with a core dump sooner or later. To avoid problems, you must do the following:

1. Call `mysql_library_init()` before any other MySQL functions. It is not thread-safe, so call it before threads are created, or protect the call with a mutex.
2. Arrange for `mysql_thread_init()` to be called early in the thread handler before calling any MySQL function. If you call `mysql_init()`, it will call `mysql_thread_init()` for you.
3. In the thread, call `mysql_thread_end()` before calling `pthread_exit()`. This frees the memory used by MySQL thread-specific variables.

The preceding notes regarding `mysql_init()` also apply to `mysql_connect()`, which calls `mysql_init()`.

22.9. MySQL PHP API

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic Web pages. It is available for most operating systems and Web servers, and can access most common

databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with a Web server.

PHP provides three different MySQL API extensions:

- [Section 22.9.3, “MySQL Improved Extension \(`mysqli`\)”](#): Stands for “MySQL, Improved”; this extension is available as of PHP 5.0.0. It is intended for use with MySQL 4.1.1 and later. This extension fully supports the authentication protocol used in MySQL 5.0, as well as the Prepared Statements and Multiple Statements APIs. In addition, this extension provides an advanced, object-oriented programming interface.
- [Section 22.9.4, “MySQL Functions \(PDO_MYSQL\) \(`MySQL \(PDO\)`\)”](#): Not its own API, but instead it's a MySQL driver for the PHP database abstraction layer PDO (PHP Data Objects). The PDO MySQL driver sits in the layer below PDO itself, and provides MySQL-specific functionality. This extension is available as of PHP 5.1.0.
- [Section 22.9.2, “Original MySQL API \(`Mysql`\)”](#): Available for PHP versions 4 and 5, this extension is intended for use with MySQL versions prior to MySQL 4.1. This extension does not support the improved authentication protocol used in MySQL 4.1, nor does it support prepared statements or multiple statements. To use this extension with MySQL 4.1, you will likely configure the MySQL server to set the `old_passwords` [534] system variable to 1 (see [Section C.5.2.4, “Client does not support authentication protocol”](#)).

This extension is deprecated as of PHP 5.5.0, and its use is highly discouraged. All users are encouraged to migrate to either `mysqli` or `PDO_MYSQL`. For further information, see [Section 22.9.1.3, “Choosing an API”](#).

The PHP distribution and documentation are available from the [PHP Web site](#).

Portions of this section are Copyright (c) 1997-2012 the PHP Documentation Group This material may be distributed only subject to the terms and conditions set forth in the Creative Commons Attribution 3.0 License or later. A copy of the Creative Commons Attribution 3.0 license is distributed with this manual. The latest version is presently available at <http://creativecommons.org/licenses/by/3.0/>.

22.9.1. Overview of the MySQL PHP drivers

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

22.9.1.1. Introduction

There are three PHP APIs for accessing the MySQL database. This guide explains the [terminology](#) used to describe each API, information about [choosing which API](#) to use, and also information to help choose which MySQL [library to use](#) with the API.

22.9.1.2. Terminology overview

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

This section provides an introduction to the options available to you when developing a PHP application that needs to interact with a MySQL database.

What is an API?

An Application Programming Interface, or API, defines the classes, methods, functions and variables that your application will need to call in order to carry out its desired task. In the case of PHP applications that need to communicate with databases the necessary APIs are usually exposed via PHP extensions.

APIs can be procedural or object-oriented. With a procedural API you call functions to carry out tasks, with the object-oriented API you instantiate classes and then call methods on the resulting objects. Of the two the latter is usually the preferred interface, as it is more modern and leads to better organized code.

When writing PHP applications that need to connect to the MySQL server there are several API options available. This document discusses what is available and how to select the best solution for your application.

What is a Connector?

In the MySQL documentation, the term *connector* refers to a piece of software that allows your application to connect to the MySQL database server. MySQL provides connectors for a variety of languages, including PHP.

If your PHP application needs to communicate with a database server you will need to write PHP code to perform such activities as connecting to the database server, querying the database and other database-related functions. Software is required to provide the API that your PHP application will use, and also handle the communication between your application and the database server, possibly using other intermediate libraries where necessary. This software is known generically as a connector, as it allows your application to *connect* to a database server.

What is a Driver?

A driver is a piece of software designed to communicate with a specific type of database server. The driver may also call a library, such as the MySQL Client Library or the MySQL Native Driver. These libraries implement the low-level protocol used to communicate with the MySQL database server.

By way of an example, the [PHP Data Objects \(PDO\) \[3270\]](#) database abstraction layer may use one of several database-specific drivers. One of the drivers it has available is the PDO MySQL driver, which allows it to interface with the MySQL server.

Sometimes people use the terms connector and driver interchangeably, this can be confusing. In the MySQL-related documentation the term “driver” is reserved for software that provides the database-specific part of a connector package.

What is an Extension?

In the PHP documentation you will come across another term - *extension*. The PHP code consists of a core, with optional extensions to the core functionality. PHP's MySQL-related extensions, such as the `mysqli` extension, and the `mysql` extension, are implemented using the PHP extension framework.

An extension typically exposes an API to the PHP programmer, to allow its facilities to be used programmatically. However, some extensions which use the PHP extension framework do not expose an API to the PHP programmer.

The PDO MySQL driver extension, for example, does not expose an API to the PHP programmer, but provides an interface to the PDO layer above it.

The terms API and extension should not be taken to mean the same thing, as an extension may not necessarily expose an API to the programmer.

22.9.1.3. Choosing an API

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

PHP offers three different APIs to connect to MySQL. Below we show the APIs provided by the `mysql`, `mysqli`, and `PDO` extensions. Each code snippet creates a connection to a MySQL server running on

"example.com" using the username "user" and the password "password". And a query is run to greet the user.

Example 22.13. Comparing the three MySQL APIs

```
<?php
// mysqli
mysqli = new mysqli("example.com", "user", "password", "database");
$result = $mysqli->query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = $result->fetch_assoc();
echo htmlentities($row['_message']);

// PDO
pdo = new PDO('mysql:host=example.com;dbname=database', 'user', 'password');
$stmt = $pdo->query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = $stmt->fetch(PDO::FETCH_ASSOC);
echo htmlentities($row['_message']);

// mysql
$c = mysql_connect("example.com", "user", "password");
mysql_select_db("database");
$result = mysql_query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = mysql_fetch_assoc($result);
echo htmlentities($row['_message']);
?>
```

Recommended API

It is recommended to use either the [mysqli](#) or [PDO_MySQL](#) extensions. It is not recommended to use the old [mysql](#) extension for new development, as it has been deprecated as of PHP 5.5.0 and will be removed in the future. A detailed feature comparison matrix is provided below. The overall performance of all three extensions is considered to be about the same. Although the performance of the extension contributes only a fraction of the total run time of a PHP web request. Often, the impact is as low as 0.1%.

Feature comparison

	ext/mysqli	PDO_MySQL	ext/mysql
PHP version introduced	5.0	5.1	2.0
Included with PHP 5.x	Yes	Yes	Yes
Development status	Active	Active	Maintenance only
Lifecycle	Active	Active	Deprecated
Recommended for new projects	Yes	Yes	No
OOP Interface	Yes	Yes	No
Procedural Interface	Yes	No	Yes
API supports non-blocking, asynchronous queries with mysqlnd	Yes	No	No
Persistent Connections	Yes	Yes	Yes
API supports Charsets	Yes	Yes	Yes
API supports server-side Prepared Statements	Yes	Yes	No

	ext/mysqli	PDO_MySQL	ext/mysql
API supports client-side Prepared Statements	No	Yes	No
API supports Stored Procedures	Yes	Yes	No
API supports Multiple Statements	Yes	Most	No
API supports Transactions	Yes	Yes	No
Transactions can be controlled with SQL	Yes	Yes	Yes
Supports all MySQL 5.1+ functionality	Yes	Most	No

22.9.1.4. Choosing a library

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqli`, `PDO_MySQL` and `mysql` PHP extensions are lightweight wrappers on top of a C client library. The extensions can either use the `mysqlnd` library or the `libmysql` library. Choosing a library is a compile time decision.

The `mysqlnd` library is part of the PHP distribution since 5.3.0. It offers features like lazy connections and query caching, features that are not available with `libmysql`, so using the built-in `mysqlnd` library is highly recommended. See the [mysqlnd documentation](#) for additional details, and a listing of features and functionality that it offers.

Example 22.14. Configure commands for using `mysqlnd` or `libmysql`

```
// Recommended, compiles with mysqlnd
$ ./configure --with-mysqli=mysqlnd --with-pdo-mysql=mysqlnd --with-mysql=mysqlnd

// Not recommended, compiles with libmysql
$ ./configure --with-mysqli=/path/to/mysql_config --with-pdo-mysql=/path/to/mysql_config --with-mysql=/pat
```

Library feature comparison

It is recommended to use the `mysqlnd` library instead of the MySQL Client Server library (`libmysql`). Both libraries are supported and constantly being improved.

	MySQL native driver (<code>mysqlnd</code>)	MySQL client server library (<code>libmysql</code>)
Part of the PHP distribution	Yes	No
PHP version introduced	5.3.0	N/A
License	PHP License 3.01	Dual-License
Development status	Active	Active
Lifecycle	No end announced	No end announced
PHP 5.4 compile default (for all MySQL extensions)	Yes	No

	MySQL native driver (mysqlnd)	MySQL client server library (libmysql)
PHP 5.3 compile default (for all MySQL extensions)	No	Yes
Compression protocol support	Yes (5.3.1+)	Yes
SSL support	Yes (5.3.3+)	Yes
Named pipe support	Yes (5.3.4+)	Yes
Non-blocking, asynchronous queries	Yes	No
Performance statistics	Yes	No
LOAD LOCAL INFILE respects the open_basedir directive	Yes	No
Uses PHP's native memory management system (e.g., follows PHP memory limits)	Yes	No
Return numeric column as double (COM_QUERY)	Yes	No
Return numeric column as string (COM_QUERY)	Yes	Yes
Plugin API	Yes	Limited
Read/Write splitting for MySQL Replication	Yes, with plugin	No
Load Balancing	Yes, with plugin	No
Fail over	Yes, with plugin	No
Lazy connections	Yes, with plugin	No
Query caching	Yes, with plugin	No
Transparent query manipulations (E.g., auto-EXPLAIN or monitoring)	Yes, with plugin	No

22.9.1.5. Concepts

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

These concepts are specific to the MySQL drivers for PHP.

22.9.1.5.1. Buffered and Unbuffered queries

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

Queries are using the buffered mode by default. This means that query results are immediately transferred from the MySQL Server to PHP in is then kept in the memory of the PHP process. This allows additional operations like counting the number of rows, and moving (seeking) the current result pointer. It also allows issuing further queries on the same connection while working on the result set. The downside of the buffered mode is that larger result sets might require quite a lot memory. The memory will be kept occupied till all references to the result set are unset or the result set was explicitly freed, which will automatically happen during request end the latest. The terminology "store result" is also used for buffered mode, as the whole result set is stored at once.

**Note**

When using libmysql as library PHP's memory limit won't count the memory used for result sets unless the data is fetched into PHP variables. With mysqlnd the memory accounted for will include the full result set.

Unbuffered MySQL queries execute the query and then return a resource while the data is still waiting on the MySQL server for being fetched. This uses less memory on the PHP-side, but can increase the load on the server. Unless the full result set was fetched from the server no further queries can be sent over the same connection. Unbuffered queries can also be referred to as "use result".

Following these characteristics buffered queries should be used in cases where you expect only a limited result set or need to know the amount of returned rows before reading all rows. Unbuffered mode should be used when you expect larger results.

Because buffered queries are the default, the examples below will demonstrate how to execute unbuffered queries with each API.

Example 22.15. Unbuffered query example: mysqli

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
$result = $mysqli->query("SELECT Name FROM City", MYSQLI_USE_RESULT);

if ($result) {
    while ($row = $result->fetch_assoc()) {
        echo $row['Name'] . PHP_EOL;
    }
}
$result->close();
?>
```

Example 22.16. Unbuffered query example: pdo_mysql

```
<?php
$pdo = new PDO("mysql:host=localhost;dbname=world", 'my_user', 'my_pass');
$pdo->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, false);

$result = $pdo->query("SELECT Name FROM City");
if ($result) {
    while ($row = $result->fetch(PDO::FETCH_ASSOC)) {
        echo $row['Name'] . PHP_EOL;
    }
}
?>
```

Example 22.17. Unbuffered query example: mysql

```
<?php
$conn = mysql_connect("localhost", "my_user", "my_pass");
$db = mysql_select_db("world");

$result = mysql_unbuffered_query("SELECT Name FROM City");
if ($result) {
    while ($row = mysql_fetch_assoc($result)) {
```

```

        echo $row['Name'] . PHP_EOL;
    }
}
?>

```

22.9.1.5.2. Character sets

Copyright 1997-2012 the PHP Documentation Group. [3186]

Ideally a proper character set will be set at the server level, and doing this is described within the [Character Set Configuration](#) section of the MySQL Server manual. Alternatively, each MySQL API offers a method to set the character set at runtime.



The character set and character escaping

The character set should be understood and defined, as it has an affect on every action, and includes security implications. For example, the escaping mechanism (e.g., `mysqli_real_escape_string` for `mysqli`, `mysql_real_escape_string` for `mysql`, and `PDO::quote` for `PDO_MySQL`) will adhere to this setting. It is important to realize that these functions will not use the character set that is defined with a query, so for example the following will not have an effect on them:

Example 22.18. Problems with setting the character set with SQL

```

<?php

$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

// Will not affect $mysqli->real_escape_string();
$mysqli->query("SET NAMES utf8");

// Will not affect $mysqli->real_escape_string();
$mysqli->query("SET CHARACTER SET utf8");

// But, this will affect $mysqli->real_escape_string();
$mysqli->set_charset('utf8');

?>

```

Below are examples that demonstrate how to properly alter the character set at runtime using each each API.

Example 22.19. Setting the character set example: `mysqli`

```

<?php

$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

if (!$mysqli->set_charset('utf8')) {
    printf("Error loading character set utf8: %s\n", $mysqli->error);
} else {
    printf("Current character set: %s\n", $mysqli->character_set_name());
}

print_r( $mysqli->get_charset() );

```

```
?>
```

Example 22.20. Setting the character set example: [pdo_mysql](#)

Note: This only works as of PHP 5.3.6.

```
<?php
$pdo = new PDO("mysql:host=localhost;dbname=world;charset=utf8", 'my_user', 'my_pass');
?>
```

Example 22.21. Setting the character set example: [mysql](#)

```
<?php

$conn = mysql_connect("localhost", "my_user", "my_pass");
$db    = mysql_select_db("world");

if (!mysql_set_charset('utf8', $conn)) {
    echo "Error: Unable to set the character set.\n";
    exit;
}

echo 'Your current character set is: ' . mysql_client_encoding($conn);

?>
```

22.9.2. Original MySQL API ([Mysql](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

This extension is deprecated as of PHP 5.5.0, and is not recommended for writing new code as it will be removed in the future. Instead, either the [mysql](#) or [PDO_MySQL](#) extension should be used. See also the [MySQL API Overview](#) for further help while choosing a MySQL API.

These functions allow you to access MySQL database servers. More information about MySQL can be found at <http://www.mysql.com/>.

Documentation for MySQL can be found at <http://dev.mysql.com/doc/>.

22.9.2.1. Installing/Configuring

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.2.1.1. Requirements

Copyright 1997-2012 the PHP Documentation Group. [3186]

In order to have these functions available, you must compile PHP with MySQL support.

22.9.2.1.2. Installation

Copyright 1997-2012 the PHP Documentation Group. [3186]

For compiling, simply use the `--with-mysql[=DIR]` configuration option where the optional `[DIR]` points to the MySQL installation directory.

Although this MySQL extension is compatible with MySQL 4.1.0 and greater, it doesn't support the extra functionality that these versions provide. For that, use the [MySQLi](#) extension.

If you would like to install the `mysql` extension along with the `mysqli` extension you have to use the same client library to avoid any conflicts.

22.9.2.1.2.1. Installation on Linux Systems

Copyright 1997-2012 the PHP Documentation Group. [3186]

Note: `[DIR]` is the path to the MySQL client library files (*headers and libraries*), which can be downloaded from MySQL.

Table 22.32. ext/mysql compile time support matrix

PHP Version	Default	Configure Options: <code>mysqlnd</code>	Configure Options: <code>libmysql</code>	Changelog
4.x.x	<code>libmysql</code>	Not Available	<code>--without-mysql</code> to disable	MySQL enabled by default, MySQL client libraries are bundled
5.0.x, 5.1.x, 5.2.x	<code>libmysql</code>	Not Available	<code>--with-mysql=[DIR]</code>	MySQL is no longer enabled by default, and the MySQL client libraries are no longer bundled
5.3.x	<code>libmysql</code>	<code>--with-mysql=mysqlnd</code>	<code>--with-mysql=[DIR]</code>	<code>mysqlnd</code> is now available
5.4.x	<code>mysqlnd</code>	<code>--with-mysql</code>	<code>--with-mysql=[DIR]</code>	<code>mysqlnd</code> is now the default

22.9.2.1.2.2. Installation on Windows Systems

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.2.1.2.2.1. PHP 4

Copyright 1997-2012 the PHP Documentation Group. [3186]

The PHP MySQL extension is compiled into PHP.

22.9.2.1.2.2.2. PHP 5.0.x, 5.1.x, 5.2.x

Copyright 1997-2012 the PHP Documentation Group. [3186]

MySQL is no longer enabled by default, so the `php_mysql.dll` DLL must be enabled inside of `php.ini`. Also, PHP needs access to the MySQL client library. A file named `libmysql.dll` is included in the Windows PHP distribution and in order for PHP to talk to MySQL this file needs to be available to the Windows systems `PATH`. See the FAQ titled "[How do I add my PHP directory to the PATH on Windows](#)" for information on how to do this. Although copying `libmysql.dll` to the Windows system directory also works (because the system directory is by default in the system's `PATH`), it's not recommended.

As with enabling any PHP extension (such as `php_mysql.dll`), the PHP directive `extension_dir` should be set to the directory where the PHP extensions are located. See also the [Manual Windows Installation Instructions](#). An example `extension_dir` value for PHP 5 is `c:\php\ext`

**Note**

If when starting the web server an error similar to the following occurs: `"Unable to load dynamic library './php_mysql.dll'"`, this is because `php_mysql.dll` and/or `libmysql.dll` cannot be found by the system.

22.9.2.1.2.2.3. PHP 5.3.0+

Copyright 1997-2012 the PHP Documentation Group. [3186]

The [MySQL Native Driver](#) is enabled by default. Include `php_mysql.dll`, but `libmysql.dll` is no longer required or used.

22.9.2.1.2.3. MySQL Installation Notes

Copyright 1997-2012 the PHP Documentation Group. [3186]

**Warning**

Crashes and startup problems of PHP may be encountered when loading this extension in conjunction with the `recode` extension. See the [recode](#) extension for more information.

**Note**

If you need charsets other than *latin* (default), you have to install external (not bundled) `libmysql` with compiled charset support.

22.9.2.1.3. Runtime Configuration

Copyright 1997-2012 the PHP Documentation Group. [3186]

The behaviour of these functions is affected by settings in `php.ini`.

Table 22.33. MySQL Configuration Options

Name	Default	Changeable	Changelog
<code>mysql.allow_local_infile</code>	"1"	PHP_INI_SYSTEM	
<code>mysql.allow_persistent</code>	"1"	PHP_INI_SYSTEM	
<code>mysql.max_persistent</code>	"-1"	PHP_INI_SYSTEM	
<code>mysql.max_links</code>	"-1"	PHP_INI_SYSTEM	
<code>mysql.trace_mode</code>	"0"	PHP_INI_ALL	Available since PHP 4.3.0.
<code>mysql.default_port</code>	NULL	PHP_INI_ALL	
<code>mysql.default_socket</code>	NULL	PHP_INI_ALL	Available since PHP 4.0.1.
<code>mysql.default_host</code>	NULL	PHP_INI_ALL	
<code>mysql.default_user</code>	NULL	PHP_INI_ALL	

Name	Default	Changeable	Changelog
mysql.default_password	NULL	PHP_INI_ALL	
mysql.connect_timeout	"60"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP <= 4.3.2. Available since PHP 4.3.0.

For further details and definitions of the PHP_INI_* modes, see the <http://www.php.net/manual/en/configuration.changes.modes>.

Here's a short explanation of the configuration directives.

mysql.allow_local_infile integer	Allow accessing, from PHP's perspective, local files with LOAD DATA statements
mysql.allow_persistent boolean	Whether to allow persistent connections to MySQL.
mysql.max_persistent integer	The maximum number of persistent MySQL connections per process.
mysql.max_links integer	The maximum number of MySQL connections per process, including persistent connections.
mysql.trace_mode boolean	Trace mode. When mysql.trace_mode is enabled, warnings for table/index scans, non free result sets, and SQL-Errors will be displayed. (Introduced in PHP 4.3.0)
mysql.default_port string	The default TCP port number to use when connecting to the database server if no other port is specified. If no default is specified, the port will be obtained from the MYSQL_TCP_PORT environment variable, the mysql-tcp entry in /etc/services or the compile-time MYSQL_PORT constant, in that order. Win32 will only use the MYSQL_PORT constant.
mysql.default_socket string	The default socket name to use when connecting to a local database server if no other socket name is specified.
mysql.default_host string	The default server host to use when connecting to the database server if no other host is specified. Doesn't apply in SQL safe mode .
mysql.default_user string	The default user name to use when connecting to the database server if no other name is specified. Doesn't apply in SQL safe mode .
mysql.default_password string	The default password to use when connecting to the database server if no other password is specified. Doesn't apply in SQL safe mode .
mysql.connect_timeout integer	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.

22.9.2.1.4. Resource Types

Copyright 1997-2012 the PHP Documentation Group. [3186]

There are two resource types used in the MySQL module. The first one is the link identifier for a database connection, the second a resource which holds the result of a query.

22.9.2.2. Changelog

Copyright 1997-2012 the PHP Documentation Group. [3186]

The following changes have been made to classes/functions/methods of this extension.

General Changelog for the ext/mysql extension

This changelog references the ext/mysql extension.

Changes to existing functions

The following list is a compilation of changelog entries from the ext/mysql functions.

Global ext/mysql changes

The following is a list of changes to the entire ext/mysql extension.

Version	Description
5.5.0	This extension has been deprecated. Connecting to a MySQL database via mysql_connect , mysql_pconnect or an implicit connection via any other mysql_* function will generate an E_DEPRECATED error.
5.5.0	All of the old deprecated functions and aliases now emit E_DEPRECATED errors. These functions are: mysql() , mysql_fieldname() , mysql_fieldtable() , mysql_fieldlen() , mysql_fieldtype() , mysql_fieldflags() , mysql_selectdb() , mysql_createdb() , mysql_dropdb() , mysql_freeresult() , mysql_numfields() , mysql_numrows() , mysql_listdbs() , mysql_listtables() , mysql_listfields() , mysql_db_name() , mysql_dbname() , mysql_tablename() , and mysql_table_name() .

22.9.2.3. Predefined Constants

Copyright 1997-2012 the PHP Documentation Group. [3186]

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Since PHP 4.3.0 it is possible to specify additional client flags for the [mysql_connect](#) and [mysql_pconnect](#) functions. The following constants are defined:

Table 22.34. MySQL client constants

Constant	Description
MYSQL_CLIENT_COMPRESS	Use compression protocol
MYSQL_CLIENT_IGNORE_SPACE	Allow space after function names
MYSQL_CLIENT_INTERACTIVE	Allow interactive_timeout seconds (instead of wait_timeout) of inactivity before closing the connection.

Constant	Description
MYSQL_CLIENT_SSL	Use SSL encryption. This flag is only available with version 4.x of the MySQL client library or newer. Version 3.23.x is bundled both with PHP 4 and Windows binaries of PHP 5.

The function [mysql_fetch_array](#) uses a constant for the different types of result arrays. The following constants are defined:

Table 22.35. MySQL fetch constants

Constant	Description
MYSQL_ASSOC	Columns are returned into the array having the fieldname as the array index.
MYSQL_BOTH	Columns are returned into the array having both a numerical index and the fieldname as the array index.
MYSQL_NUM	Columns are returned into the array having a numerical index to the fields. This index starts with 0, the first field in the result.

22.9.2.4. Examples

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

22.9.2.4.1. MySQL extension overview example

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

This simple example shows how to connect, execute a query, print resulting rows and disconnect from a MySQL database.

Example 22.22. MySQL extension overview example

```
<?php
// Connecting, selecting database
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    or die('Could not connect: ' . mysql_error());
echo 'Connected successfully';
mysql_select_db('my_database') or die('Could not select database');

// Performing SQL query
$query = 'SELECT * FROM my_table';
$result = mysql_query($query) or die('Query failed: ' . mysql_error());

// Printing results in HTML
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";

// Free resultset
mysql_free_result($result);
```

```
// Closing connection
mysql_close($link);
?>
```

22.9.2.5. MySQL Functions

Copyright 1997-2012 the PHP Documentation Group. [3186]



Note

Most MySQL functions accept *link_identifier* as the last optional parameter. If it is not provided, last opened connection is used. If it doesn't exist, connection is tried to establish with default parameters defined in `php.ini`. If it is not successful, functions return `FALSE`.

22.9.2.5.1. `mysql_affected_rows`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_affected_rows`

Get number of affected rows in previous MySQL operation



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the `MySQLi` or `PDO_MySQL` extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_affected_rows
PDOStatement::rowCount
```

Description

```
int mysql_affected_rows(
    resource link_identifier
    = =NULL);
```

Get the number of affected rows by the last INSERT, UPDATE, REPLACE or DELETE query associated with *link_identifier*.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the number of affected rows on success, and -1 if the last query failed.

If the last query was a DELETE query with no WHERE clause, all of the records will have been deleted from the table but this function will return zero with MySQL versions prior to 4.1.2.

When using UPDATE, MySQL will not update columns where the new value is the same as the old value. This creates the possibility that `mysql_affected_rows` may not actually equal the number of rows matched, only the number of rows that were literally affected by the query.

The REPLACE statement first deletes the record with the same primary key and then inserts the new record. This function returns the number of deleted records plus the number of inserted records.

In the case of "INSERT ... ON DUPLICATE KEY UPDATE" queries, the return value will be 1 if an insert was performed, or 2 for an update of an existing row.

Examples

Example 22.23. `mysql_affected_rows` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('mydb');

/* this should return the correct numbers of deleted records */
mysql_query('DELETE FROM mytable WHERE id < 10');
printf("Records deleted: %d\n", mysql_affected_rows());

/* with a where clause that is never true, it should return 0 */
mysql_query('DELETE FROM mytable WHERE 0');
printf("Records deleted: %d\n", mysql_affected_rows());
?>
```

The above example will output something similar to:

```
Records deleted: 10
Records deleted: 0
```

Example 22.24. `mysql_affected_rows` example using transactions

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('mydb');

/* Update records */
mysql_query("UPDATE mytable SET used=1 WHERE id < 10");
printf ("Updated records: %d\n", mysql_affected_rows());
mysql_query("COMMIT");
?>
```

The above example will output something similar to:

```
Updated Records: 10
```

Notes



Transactions

If you are using transactions, you need to call `mysql_affected_rows` after your INSERT, UPDATE, or DELETE query, not after the COMMIT.



SELECT Statements

To retrieve the number of rows returned by a SELECT, it is possible to use `mysql_num_rows`.



Cascaded Foreign Keys

`mysql_affected_rows` does not count rows affected implicitly through the use of ON DELETE CASCADE and/or ON UPDATE CASCADE in foreign key constraints.

See Also

[mysql_num_rows](#)

[mysql_info](#)

22.9.2.5.2. [mysql_client_encoding](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_client_encoding](#)

Returns the name of the character set



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_character_set_name](#)

Description

```
string mysql_client_encoding(  
    resource link_identifier  
    = =NULL);
```

Retrieves the `character_set` variable from MySQL.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the default character set name for the current connection.

Examples

Example 22.25. `mysql_client_encoding` example

```
<?php
$link    = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$charset = mysql_client_encoding($link);

echo "The current character set is: $charset\n";
?>
```

The above example will output something similar to:

```
The current character set is: latin1
```

See Also

`mysql_set_charset`
`mysql_real_escape_string`

22.9.2.5.3. `mysql_close`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_close`

Close MySQL connection



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_close`

PDO: Assign the value of `NULL` to the PDO object

Description

```
bool mysql_close(
    resource link_identifier
    = =NULL);
```

`mysql_close` closes the non-persistent connection to the MySQL server that's associated with the specified link identifier. If `link_identifier` isn't specified, the last opened link is used.

Using `mysql_close` isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution. See also [freeing resources](#).

Parameters

`link_identifier`

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.26. [mysql_close](#) example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

The above example will output:

```
Connected successfully
```

Notes



Note

[mysql_close](#) will not close persistent links created by [mysql_pconnect](#).

See Also

[mysql_connect](#)
[mysql_free_result](#)

22.9.2.5.4. [mysql_connect](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_connect](#)

Open a connection to a MySQL Server



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_connect](#)
[PDO::__construct](#)

Description

```
resource mysql_connect(
    string server
    = =ini_get("mysql.default_host"),
```

```

string username
    = =ini_get("mysql.default_user"),
string password
    = =ini_get("mysql.default_password"),
bool new_link
    = =false,
int client_flags
    = =0);

```

Opens or reuses a connection to a MySQL server.

Parameters

- server* The MySQL server. It can also include a port number. e.g. "hostname:port" or a path to a local socket e.g. ":/path/to/socket" for the localhost.
- If the PHP directive `mysql.default_host` is undefined (default), then the default value is 'localhost:3306'. In [SQL safe mode](#), this parameter is ignored and value 'localhost:3306' is always used.
- username* The username. Default value is defined by `mysql.default_user`. In [SQL safe mode](#), this parameter is ignored and the name of the user that owns the server process is used.
- password* The password. Default value is defined by `mysql.default_password`. In [SQL safe mode](#), this parameter is ignored and empty password is used.
- new_link* If a second call is made to `mysql_connect` with the same arguments, no new link will be established, but instead, the link identifier of the already opened link will be returned. The *new_link* parameter modifies this behavior and makes `mysql_connect` always open a new link, even if `mysql_connect` was called before with the same parameters. In [SQL safe mode](#), this parameter is ignored.
- client_flags* The *client_flags* parameter can be a combination of the following constants: 128 (enable `LOAD DATA LOCAL` handling), `MYSQL_CLIENT_SSL`, `MYSQL_CLIENT_COMPRESS`, `MYSQL_CLIENT_IGNORE_SPACE` or `MYSQL_CLIENT_INTERACTIVE`. Read the section about [Table 22.34, "MySQL client constants"](#) for further information. In [SQL safe mode](#), this parameter is ignored.

Return Values

Returns a MySQL link identifier on success or `FALSE` on failure.

Changelog

Version	Description
5.5.0	This function will generate an <code>E_DEPRECATED</code> error.
4.3.0	Added the <i>client_flags</i> parameter.
4.2.0	Added the <i>new_link</i> parameter.

Examples

Example 22.27. `mysql_connect` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Example 22.28. `mysql_connect` example using `hostname:port` syntax

```
<?php
// we connect to example.com and port 3307
$link = mysql_connect('example.com:3307', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);

// we connect to localhost at port 3307
$link = mysql_connect('127.0.0.1:3307', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Example 22.29. `mysql_connect` example using `"/path/to/socket"` syntax

```
<?php
// we connect to localhost and socket e.g. /tmp/mysql.sock

// variant 1: omit localhost
$link = mysql_connect('/:tmp/mysql', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);

// variant 2: with localhost
$link = mysql_connect('localhost:/tmp/mysql.sock', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Notes**Note**

Whenever you specify "localhost" or "localhost:port" as server, the MySQL client library will override this and try to connect to a local socket (named pipe on

Windows). If you want to use TCP/IP, use "127.0.0.1" instead of "localhost". If the MySQL client library tries to connect to the wrong local socket, you should set the correct path as `mysql.default_host` string in your PHP configuration and leave the server field blank.

**Note**

The link to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling `mysql_close`.

**Note**

You can suppress the error message on failure by prepending a `@` to the function name.

**Note**

Error "Can't create TCP/IP socket (10106)" usually means that the `variables_order` configure directive doesn't contain character `E`. On Windows, if the environment is not copied the `SYSTEMROOT` environment variable won't be available and PHP will have problems loading Winsock.

See Also

[mysql_pconnect](#)
[mysql_close](#)

22.9.2.5.5. [mysql_create_db](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_create_db](#)

Create a MySQL database

**Warning**

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_query](#)
[PDO::query](#)

Description

```
bool mysql_create_db(  
    string database_name,  
    resource link_identifier  
    = =NULL);
```

[mysql_create_db](#) attempts to create a new database on the server associated with the specified link identifier.

Parameters

[database_name](#) The name of the database being created.

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.30. [mysql_create_db](#) alternative example

The function [mysql_create_db](#) is deprecated. It is preferable to use [mysql_query](#) to issue an sql [CREATE DATABASE](#) statement instead.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}

$sql = 'CREATE DATABASE my_db';
if (mysql_query($sql, $link)) {
    echo "Database my_db created successfully\n";
} else {
    echo 'Error creating database: ' . mysql_error() . "\n";
}
?>
```

The above example will output something similar to:

```
Database my_db created successfully
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
[mysql_createdb](#)



Note

This function will not be available if the MySQL extension was built against a MySQL 4.x client library.

See Also

[mysql_query](#)

[mysql_select_db](#)

22.9.2.5.6. [mysql_data_seek](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_data_seek](#)

Move internal result pointer



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_data_seek
PDO::FETCH_ORI_ABS
```

Description

```
bool mysql_data_seek(
    resource result,
    int row_number);
```

`mysql_data_seek` moves the internal row pointer of the MySQL result associated with the specified result identifier to point to the specified row number. The next call to a MySQL fetch function, such as `mysql_fetch_assoc`, would return that row.

`row_number` starts at 0. The `row_number` should be a value in the range from 0 to `mysql_num_rows - 1`. However if the result set is empty (`mysql_num_rows == 0`), a seek to 0 will fail with a [E_WARNING](#) and `mysql_data_seek` will return `FALSE`.

Parameters

result The result resource that is being evaluated. This result comes from a call to `mysql_query`.

row_number The desired row number of the new result pointer.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.31. `mysql_data_seek` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
$db_selected = mysql_select_db('sample_db');
if (!$db_selected) {
    die('Could not select database: ' . mysql_error());
}
$query = 'SELECT last_name, first_name FROM friends';
$result = mysql_query($query);
if (!$result) {
    die('Query failed: ' . mysql_error());
}
/* fetch rows in reverse order */
for ($i = mysql_num_rows($result) - 1; $i >= 0; $i--) {
    if (!mysql_data_seek($result, $i)) {
        echo "Cannot seek to row $i: " . mysql_error() . "\n";
    }
}
```

```

        continue;
    }

    if (!$row = mysql_fetch_assoc($result)) {
        continue;
    }

    echo $row['last_name'] . ' ' . $row['first_name'] . "<br />\n";
}

mysql_free_result($result);
?>

```

Notes**Note**

The function `mysql_data_seek` can be used in conjunction only with `mysql_query`, not with `mysql_unbuffered_query`.

See Also

[mysql_query](#)
[mysql_num_rows](#)
[mysql_fetch_row](#)
[mysql_fetch_assoc](#)
[mysql_fetch_array](#)
[mysql_fetch_object](#)

22.9.2.5.7. [mysql_db_name](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_db_name](#)

Retrieves database name from the call to `mysql_list_dbs`

**Warning**

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

Query: `SELECT DATABASE()`

Description

```

string mysql_db_name(
    resource result,
    int row,
    mixed field
    = =NULL);

```

Retrieve the database name from a call to `mysql_list_dbs`.

Parameters

result

The result pointer from a call to `mysql_list_dbs`.

row The index into the result set.

field The field name.

Return Values

Returns the database name on success, and `FALSE` on failure. If `FALSE` is returned, use [mysql_error](#) to determine the nature of the error.

Changelog

Version	Description
5.5.0	The mysql_list_dbs function is deprecated, and emits an <code>E_DEPRECATED</code> level error.

Examples

Example 22.32. [mysql_db_name](#) example

```
<?php
error_reporting(E_ALL);

$link = mysql_connect('dbhost', 'username', 'password');
$db_list = mysql_list_dbs($link);

$i = 0;
$cnt = mysql_num_rows($db_list);
while ($i < $cnt) {
    echo mysql_db_name($db_list, $i) . "\n";
    $i++;
}
?>
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
[mysql_dbname](#)

See Also

[mysql_list_dbs](#)
[mysql_tablename](#)

22.9.2.5.8. [mysql_db_query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_db_query](#)

Selects a database and executes a query on it



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL:](#)

[choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_select_db then the query
PDO::__construct
```

Description

```
resource mysql_db_query(
    string database,
    string query,
    resource link_identifier
    = =NULL);
```

`mysql_db_query` selects a database, and executes a query on it.



Warning

This function has been *DEPRECATED* as of PHP 5.3.0. Relying on this feature is highly discouraged.

Parameters

database The name of the database that will be selected.

query The MySQL query.

Data inside the query should be [properly escaped](#).

link_identifier The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a positive MySQL result resource to the query result, or `FALSE` on error. The function also returns `TRUE` / `FALSE` for `INSERT/UPDATE/DELETE` queries to indicate success/failure.

Changelog

Version	Description
5.3.0	This function now throws an <code>E_DEPRECATED</code> notice.
4.0.6	This function is deprecated, do not use this function. Use <code>mysqli_select_db</code> and <code>mysql_query</code> instead.

Examples

Example 22.33. `mysql_db_query` alternative example

```
<?php
if (!$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')) {
    echo 'Could not connect to mysql';
    exit;
}
```

```
}  
  
if (!mysql_select_db('mysql_dbname', $link)) {  
    echo 'Could not select database';  
    exit;  
}  
  
$sql    = 'SELECT foo FROM bar WHERE id = 42';  
$result = mysql_query($sql, $link);  
  
if (!$result) {  
    echo "DB Error, could not query the database\n";  
    echo 'MySQL Error: ' . mysql_error();  
    exit;  
}  
  
while ($row = mysql_fetch_assoc($result)) {  
    echo $row['foo'];  
}  
  
mysql_free_result($result);  
  
?>
```

Notes



Note

Be aware that this function does *NOT* switch back to the database you were connected before. In other words, you can't use this function to *temporarily* run a sql query on another database, you would have to manually switch back. Users are strongly encouraged to use the `database.table` syntax in their sql queries or [mysql_select_db](#) instead of this function.

See Also

[mysql_query](#)
[mysql_select_db](#)

22.9.2.5.9. [mysql_drop_db](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_drop_db](#)

Drop (delete) a MySQL database



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

Execute a `DROP DATABASE` query

Description

```
bool mysql_drop_db(  
    string database_name,
```

```
resource link_identifier  
= =NULL);
```

[mysql_drop_db](#) attempts to drop (remove) an entire database from the server associated with the specified link identifier. This function is deprecated, it is preferable to use [mysql_query](#) to issue an `sql DROP DATABASE` statement instead.

Parameters

<i>database_name</i>	The name of the database that will be deleted.
<i>link_identifier</i>	The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect is assumed. If no such link is found, it will try to create one as if mysql_connect was called with no arguments. If no connection is found or established, an <code>E_WARNING</code> level error is generated.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.34. [mysql_drop_db](#) alternative example

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
if (!$link) {  
    die('Could not connect: ' . mysql_error());  
}  
  
$sql = 'DROP DATABASE my_db';  
if (mysql_query($sql, $link)) {  
    echo "Database my_db was successfully dropped\n";  
} else {  
    echo 'Error dropping database: ' . mysql_error() . "\n";  
}  
?>
```

Notes



Warning

This function will not be available if the MySQL extension was built against a MySQL 4.x client library.



Note

For backward compatibility, the following deprecated alias may be used:
[mysql_dropdb](#)

See Also

[mysql_query](#)

22.9.2.5.10. [mysql_errno](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_errno`

Returns the numerical value of the error message from previous MySQL operation

**Warning**

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_errno`
`PDO::errorCode`

Description

```
int mysql_errno(  
    resource link_identifier  
    = =NULL);
```

Returns the error number from the last MySQL function.

Errors coming back from the MySQL database backend no longer issue warnings. Instead, use `mysql_errno` to retrieve the error code. Note that this function only returns the error code from the most recently executed MySQL function (not including `mysql_error` and `mysql_errno`), so if you want to use it, make sure you check the value before calling another MySQL function.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the error number from the last MySQL function, or 0 (zero) if no error occurred.

Examples**Example 22.35. `mysql_errno` example**

```
<?php  
$link = mysql_connect("localhost", "mysql_user", "mysql_password");  
  
if (!mysql_select_db("nonexistentdb", $link)) {  
    echo mysql_errno($link) . ": " . mysql_error($link) . "\n";  
}  
  
mysql_select_db("kossu", $link);  
if (!mysql_query("SELECT * FROM nonexistenttable", $link)) {  
    echo mysql_errno($link) . ": " . mysql_error($link) . "\n";  
}  
?>
```

The above example will output something similar to:

```
1049: Unknown database 'nonexistentdb'  
1146: Table 'kossu.nonexistenttable' doesn't exist
```

See Also

[mysql_error](#)
[MySQL error codes](#)

22.9.2.5.11. [mysql_error](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_error](#)

Returns the text of the error message from previous MySQL operation



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_error](#)
[PDO::errorInfo](#)

Description

```
string mysql_error(  
    resource link_identifier  
    = =NULL);
```

Returns the error text from the last MySQL function. Errors coming back from the MySQL database backend no longer issue warnings. Instead, use [mysql_error](#) to retrieve the error text. Note that this function only returns the error text from the most recently executed MySQL function (not including [mysql_error](#) and [mysql_errno](#)), so if you want to use it, make sure you check the value before calling another MySQL function.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns the error text from the last MySQL function, or '' (empty string) if no error occurred.

Examples

Example 22.36. [mysql_error](#) example

```
<?php  
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
```

```
mysql_select_db("nonexistentdb", $link);
echo mysql_errno($link) . ": " . mysql_error($link). "\n";

mysql_select_db("kossu", $link);
mysql_query("SELECT * FROM nonexistenttable", $link);
echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
?>
```

The above example will output something similar to:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

See Also

[mysql_errno](#)
[MySQL error codes](#)

22.9.2.5.12. [mysql_escape_string](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_escape_string](#)

Escapes a string for use in a `mysql_query`



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_escape_string
PDO::quote
```

Description

```
string mysql_escape_string(
    string unescaped_string);
```

This function will escape the *unescaped_string*, so that it is safe to place it in a `mysql_query`. This function is deprecated.

This function is identical to `mysql_real_escape_string` except that `mysql_real_escape_string` takes a connection handler and escapes the string according to the current character set. `mysql_escape_string` does not take a connection argument and does not respect the current charset setting.



Warning

This function has been *DEPRECATED* as of PHP 5.3.0. Relying on this feature is highly discouraged.

Parameters

[unescaped_string](#)

The string that is to be escaped.

Return Values

Returns the escaped string.

Changelog

Version	Description
5.3.0	This function now throws an E_DEPRECATED notice.
4.3.0	This function became deprecated, do not use this function. Instead, use mysql_real_escape_string .

Examples

Example 22.37. [mysql_escape_string](#) example

```
<?php
$item = "Zak's Laptop";
$escaped_item = mysql_escape_string($item);
printf("Escaped string: %s\n", $escaped_item);
?>
```

The above example will output:

```
Escaped string: Zak\'s Laptop
```

Notes



Note

[mysql_escape_string](#) does not escape % and _.

See Also

[mysql_real_escape_string](#)

[addslashes](#)

The [magic_quotes_gpc](#) directive.

22.9.2.5.13. [mysql_fetch_array](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_fetch_array](#)

Fetch a result row as an associative array, a numeric array, or both



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL:](#)

[choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_fetch_array
PDOStatement::fetch
```

Description

```
array mysql_fetch_array(
    resource result,
    int result_type
    = =MYSQL_BOTH);
```

Returns an array that corresponds to the fetched row and moves the internal data pointer ahead.

Parameters

result The result resource that is being evaluated. This result comes from a call to `mysql_query`.

result_type The type of array that is to be fetched. It's a constant and can take the following values: `MYSQL_ASSOC` , `MYSQL_NUM` , and `MYSQL_BOTH` .

Return Values

Returns an array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows. The type of returned array depends on how *result_type* is defined. By using `MYSQL_BOTH` (default), you'll get an array with both associative and number indices. Using `MYSQL_ASSOC` , you only get associative indices (as `mysql_fetch_assoc` works), using `MYSQL_NUM` , you only get number indices (as `mysql_fetch_row` works).

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you must use the numeric index of the column or make an alias for the column. For aliased columns, you cannot access the contents with the original column name.

Examples

Example 22.38. Query with aliased duplicate field names

```
SELECT table1.field AS foo, table2.field AS bar FROM table1, table2
```

Example 22.39. `mysql_fetch_array` with `MYSQL_NUM`

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");

$result = mysql_query("SELECT id, name FROM mytable");

while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    printf("ID: %s Name: %s", $row[0], $row[1]);
}

mysql_free_result($result);
?>
```

Example 22.40. [mysql_fetch_array](#) with [MYSQL_ASSOC](#)

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");

$result = mysql_query("SELECT id, name FROM mytable");

while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    printf("ID: %s Name: %s", $row["id"], $row["name"]);
}

mysql_free_result($result);
?>
```

Example 22.41. [mysql_fetch_array](#) with [MYSQL_BOTH](#)

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");

$result = mysql_query("SELECT id, name FROM mytable");

while ($row = mysql_fetch_array($result, MYSQL_BOTH)) {
    printf("ID: %s Name: %s", $row[0], $row["name"]);
}

mysql_free_result($result);
?>
```

Notes**Performance**

An important thing to note is that using [mysql_fetch_array](#) is *not significantly* slower than using [mysql_fetch_row](#), while it provides a significant added value.

**Note**

Field names returned by this function are *case-sensitive*.

**Note**

This function sets NULL fields to the PHP [NULL](#) value.

See Also

[mysql_fetch_row](#)
[mysql_fetch_assoc](#)
[mysql_data_seek](#)
[mysql_query](#)

22.9.2.5.14. *mysql_fetch_assoc*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysql_fetch_assoc*

Fetch a result row as an associative array



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_fetch_assoc  
PDOStatement::fetch(PDO::FETCH_ASSOC)
```

Description

```
array mysql_fetch_assoc(  
    resource result);
```

Returns an associative array that corresponds to the fetched row and moves the internal data pointer ahead. *mysql_fetch_assoc* is equivalent to calling *mysql_fetch_array* with `MYSQL_ASSOC` for the optional second parameter. It only returns an associative array.

Parameters

result The result resource that is being evaluated. This result comes from a call to *mysql_query*.

Return Values

Returns an associative array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you either need to access the result with numeric indices by using *mysql_fetch_row* or add alias names. See the example at the *mysql_fetch_array* description about aliases.

Examples

Example 22.42. An expanded *mysql_fetch_assoc* example

```
<?php  
  
$conn = mysql_connect("localhost", "mysql_user", "mysql_password");  
  
if (!$conn) {  
    echo "Unable to connect to DB: " . mysql_error();  
    exit;  
}  
  
if (!mysql_select_db("mydbname")) {  
    echo "Unable to select mydbname: " . mysql_error();  
    exit;  
}
```

```
$sql = "SELECT id as userid, fullname, userstatus
      FROM   sometable
      WHERE  userstatus = 1";

$result = mysql_query($sql);

if (!$result) {
    echo "Could not successfully run query ($sql) from DB: " . mysql_error();
    exit;
}

if (mysql_num_rows($result) == 0) {
    echo "No rows found, nothing to print so am exiting";
    exit;
}

// While a row of data exists, put that row in $row as an associative array
// Note: If you're expecting just one row, no need to use a loop
// Note: If you put extract($row); inside the following loop, you'll
//       then create $userid, $fullname, and $userstatus
while ($row = mysql_fetch_assoc($result)) {
    echo $row["userid"];
    echo $row["fullname"];
    echo $row["userstatus"];
}

mysql_free_result($result);

?>
```

Notes



Performance

An important thing to note is that using [mysql_fetch_assoc](#) is *not significantly* slower than using [mysql_fetch_row](#), while it provides a significant added value.



Note

Field names returned by this function are *case-sensitive*.



Note

This function sets NULL fields to the PHP [NULL](#) value.

See Also

[mysql_fetch_row](#)
[mysql_fetch_array](#)
[mysql_data_seek](#)
[mysql_query](#)
[mysql_error](#)

22.9.2.5.15. [mysql_fetch_field](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_fetch_field](#)

Get column information from a result and return as an object



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_fetch_field`
`PDOStatement::getColumnMeta`

Description

```
object mysql_fetch_field(  
    resource result,  
    int field_offset  
    = 0);
```

Returns an object containing field information. This function can be used to obtain information about fields in the provided query result.

Parameters

<i>result</i>	The result resource that is being evaluated. This result comes from a call to <code>mysql_query</code> .
<i>field_offset</i>	The numerical field offset. If the field offset is not specified, the next field that was not yet retrieved by this function is retrieved. The <i>field_offset</i> starts at 0.

Return Values

Returns an object containing field information. The properties of the object are:

- name - column name
- table - name of the table the column belongs to, which is the alias name if one is defined
- max_length - maximum length of the column
- not_null - 1 if the column cannot be `NULL`
- primary_key - 1 if the column is a primary key
- unique_key - 1 if the column is a unique key
- multiple_key - 1 if the column is a non-unique key
- numeric - 1 if the column is numeric
- blob - 1 if the column is a BLOB
- type - the type of the column
- unsigned - 1 if the column is unsigned
- zerofill - 1 if the column is zero-filled

Examples

Example 22.43. [mysql_fetch_field](#) example

```
<?php
$conn = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$conn) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('database');
$result = mysql_query('select * from table');
if (!$result) {
    die('Query failed: ' . mysql_error());
}
/* get column metadata */
$i = 0;
while ($i < mysql_num_fields($result)) {
    echo "Information for column $i:<br />\n";
    $meta = mysql_fetch_field($result, $i);
    if (!$meta) {
        echo "No information available<br />\n";
    }
    echo "<pre>
blob:          $meta->blob
max_length:    $meta->max_length
multiple_key:  $meta->multiple_key
name:          $meta->name
not_null:      $meta->not_null
numeric:       $meta->numeric
primary_key:   $meta->primary_key
table:         $meta->table
type:          $meta->type
unique_key:    $meta->unique_key
unsigned:      $meta->unsigned
zerofill:      $meta->zerofill
</pre>";
    $i++;
}
mysql_free_result($result);
?>
```

Notes



Note

Field names returned by this function are *case-sensitive*.



Note

If field or tablename are aliased in the SQL query the aliased name will be returned. The original name can be retrieved for instance by using [mysqli_result::fetch_field](#).

See Also

[mysql_field_seek](#)

22.9.2.5.16. [mysql_fetch_lengths](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_fetch_lengths`

Get the length of each output in a result

**Warning**

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_fetch_lengths`
`PDOStatement::getColumnMeta`

Description

```
array mysql_fetch_lengths(  
    resource result);
```

Returns an array that corresponds to the lengths of each field in the last row fetched by MySQL.

`mysql_fetch_lengths` stores the lengths of each result column in the last row returned by `mysql_fetch_row`, `mysql_fetch_assoc`, `mysql_fetch_array`, and `mysql_fetch_object` in an array, starting at offset 0.

Parameters

result The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

An array of lengths on success or `FALSE` on failure.

Examples**Example 22.44. A `mysql_fetch_lengths` example**

```
<?php  
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");  
if (!$result) {  
    echo 'Could not run query: ' . mysql_error();  
    exit;  
}  
$row     = mysql_fetch_assoc($result);  
$lengths = mysql_fetch_lengths($result);  
  
print_r($row);  
print_r($lengths);  
?>
```

The above example will output something similar to:

```
Array  
(  
    [id] => 42  
    [email] => user@example.com  
)
```

```
Array
(
    [0] => 2
    [1] => 16
)
```

See Also

[mysql_field_len](#)
[mysql_fetch_row](#)
[strlen](#)

22.9.2.5.17. [mysql_fetch_object](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_fetch_object](#)

Fetch a result row as an object



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_fetch_object](#)
`PDOStatement::fetch(PDO::FETCH_OBJ)`

Description

```
object mysql_fetch_object(
    resource result,
    string class_name,
    array params);
```

Returns an object with properties that correspond to the fetched row and moves the internal data pointer ahead.

Parameters

<i>result</i>	The result resource that is being evaluated. This result comes from a call to mysql_query .
<i>class_name</i>	The name of the class to instantiate, set the properties of and return. If not specified, a stdClass object is returned.
<i>params</i>	An optional array of parameters to pass to the constructor for class_name objects.

Return Values

Returns an object with string properties that correspond to the fetched row, or [FALSE](#) if there are no more rows.

Changelog

Version	Description
5.0.0	Added the ability to return as a different object.

Examples

Example 22.45. [mysql_fetch_object](#) example

```
<?php
mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");
$result = mysql_query("select * from mytable");
while ($row = mysql_fetch_object($result)) {
    echo $row->user_id;
    echo $row->fullname;
}
mysql_free_result($result);
?>
```

Example 22.46. [mysql_fetch_object](#) example

```
<?php
class foo {
    public $name;
}

mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");

$result = mysql_query("select name from mytable limit 1");
$obj = mysql_fetch_object($result, 'foo');
var_dump($obj);
?>
```

Notes



Performance

Speed-wise, the function is identical to [mysql_fetch_array](#), and almost as quick as [mysql_fetch_row](#) (the difference is insignificant).



Note

[mysql_fetch_object](#) is similar to [mysql_fetch_array](#), with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).



Note

Field names returned by this function are *case-sensitive*.



Note

This function sets NULL fields to the PHP [NULL](#) value.

See Also

[mysql_fetch_array](#)
[mysql_fetch_assoc](#)
[mysql_fetch_row](#)
[mysql_data_seek](#)
[mysql_query](#)

22.9.2.5.18. [mysql_fetch_row](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_fetch_row](#)

Get a result row as an enumerated array



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_fetch_row](#)
`PDOStatement::fetch(PDO::FETCH_NUM)`

Description

```
array mysql_fetch_row(  
    resource result);
```

Returns a numerical array that corresponds to the fetched row and moves the internal data pointer ahead.

Parameters

result The result resource that is being evaluated. This result comes from a call to [mysql_query](#).

Return Values

Returns an numerical array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows.

[mysql_fetch_row](#) fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Examples

Example 22.47. Fetching one row with [mysql_fetch_row](#)

```
<?php  
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");  
if (!$result) {  
    echo 'Could not run query: ' . mysql_error();  
    exit;  
}  
$row = mysql_fetch_row($result);  
  
echo $row[0]; // 42
```

```
echo $row[1]; // the email value
?>
```

Notes



Note

This function sets NULL fields to the PHP [NULL](#) value.

See Also

[mysql_fetch_array](#)
[mysql_fetch_assoc](#)
[mysql_fetch_object](#)
[mysql_data_seek](#)
[mysql_fetch_lengths](#)
[mysql_result](#)

22.9.2.5.19. [mysql_field_flags](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_field_flags](#)

Get the flags associated with the specified field in a result



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli\_fetch\_fields [flags]  
PDOStatement::getColumnMeta [flags]
```

Description

```
string mysql_field_flags(  
    resource result,  
    int field_offset);
```

[mysql_field_flags](#) returns the field flags of the specified field. The flags are reported as a single word per flag separated by a single space, so that you can split the returned value using [explode](#).

Parameters

result

The result resource that is being evaluated. This result comes from a call to [mysql_query](#).

field_offset

The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level [E_WARNING](#) is also issued.

Return Values

Returns a string of flags associated with the result or [FALSE](#) on failure.

The following flags are reported, if your version of MySQL is current enough to support them: `"not_null"`, `"primary_key"`, `"unique_key"`, `"multiple_key"`, `"blob"`, `"unsigned"`, `"zerofill"`, `"binary"`, `"enum"`, `"auto_increment"` and `"timestamp"`.

Examples

Example 22.48. A `mysql_field_flags` example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
$flags = mysql_field_flags($result, 0);

echo $flags;
print_r(explode(' ', $flags));
?>
```

The above example will output something similar to:

```
not_null primary_key auto_increment
Array
(
    [0] => not_null
    [1] => primary_key
    [2] => auto_increment
)
```

Notes



Note

For backward compatibility, the following deprecated alias may be used: `mysql_fieldflags`

See Also

`mysql_field_type`
`mysql_field_len`

22.9.2.5.20. `mysql_field_len`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_field_len`

Returns the length of the specified field



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the `MySQLi` or `PDO_MySQL` extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_fetch_field_direct [length]  
PDOStatement::getColumnMeta [len]
```

Description

```
int mysql_field_len(  
    resource result,  
    int field_offset);
```

`mysql_field_len` returns the length of the specified field.

Parameters

result The result resource that is being evaluated. This result comes from a call to `mysql_query`.

field_offset The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level `E_WARNING` is also issued.

Return Values

The length of the specified field index on success or `FALSE` on failure.

Examples

Example 22.49. `mysql_field_len` example

```
<?php  
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");  
if (!$result) {  
    echo 'Could not run query: ' . mysql_error();  
    exit;  
}  
  
// Will get the length of the id field as specified in the database  
// schema.  
$length = mysql_field_len($result, 0);  
echo $length;  
?>
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
`mysql_fieldlen`

See Also

[mysql_fetch_lengths](#)
[strlen](#)

22.9.2.5.21. `mysql_field_name`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_field_name`

Get the name of the specified field in a result



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_fetch_field_direct [name] or [orgname]
PDOStatement::getColumnMeta [name]
```

Description

```
string mysql_field_name(
    resource result,
    int field_offset);
```

`mysql_field_name` returns the name of the specified field index.

Parameters

<i>result</i>	The result resource that is being evaluated. This result comes from a call to <code>mysql_query</code> .
<i>field_offset</i>	The numerical field offset. The <i>field_offset</i> starts at 0. If <i>field_offset</i> does not exist, an error of level <code>E_WARNING</code> is also issued.

Return Values

The name of the specified field index on success or `FALSE` on failure.

Examples

Example 22.50. `mysql_field_name` example

```
<?php
/* The users table consists of three fields:
 *  user_id
 *  username
 *  password.
 */
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect to MySQL server: ' . mysql_error());
}
$dbname = 'mydb';
$db_selected = mysql_select_db($dbname, $link);
if (!$db_selected) {
    die("Could not set $dbname: " . mysql_error());
}
$res = mysql_query('select * from users', $link);

echo mysql_field_name($res, 0) . "\n";
echo mysql_field_name($res, 2);
?>
```

The above example will output:

```
user_id
password
```

Notes



Note

Field names returned by this function are *case-sensitive*.



Note

For backward compatibility, the following deprecated alias may be used:
`mysql_fieldname`

See Also

`mysql_field_type`
`mysql_field_len`

22.9.2.5.22. `mysql_field_seek`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_field_seek`

Set result pointer to a specified field offset



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the `MySQLi` or `PDO_MySQL` extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_field_seek`
`PDOStatement::fetch` using optional parameters

Description

```
bool mysql_field_seek(
    resource result,
    int field_offset);
```

Seeks to the specified field offset. If the next call to `mysql_fetch_field` doesn't include a field offset, the field offset specified in `mysql_field_seek` will be returned.

Parameters

result

The result resource that is being evaluated. This result comes from a call to `mysql_query`.

field_offset

The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level `E_WARNING` is also issued.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

See Also

[mysql_fetch_field](#)

22.9.2.5.23. [mysql_field_table](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_field_table](#)

Get name of the table the specified field is in



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_fetch_field_direct](#) [table] or [orgtable]
[PDOStatement::getColumnMeta](#) [table]

Description

```
string mysql_field_table(  
    resource result,  
    int field_offset);
```

Returns the name of the table that the specified field is in.

Parameters

- result* The result resource that is being evaluated. This result comes from a call to [mysql_query](#).
- field_offset* The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level [E_WARNING](#) is also issued.

Return Values

The name of the table on success.

Examples

Example 22.51. A [mysql_field_table](#) example

```
<?php  
  
$query = "SELECT account.*, country.* FROM account, country WHERE country.name = 'Portugal' AND account.co  
  
// get the result from the DB  
$result = mysql_query($query);  
  
// Lists the table name and then the field name  
for ($i = 0; $i < mysql_num_fields($result); ++$i) {
```

```
$table = mysql_field_table($result, $i);
$field = mysql_field_name($result, $i);

echo "$table: $field\n";
}
?>
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
[mysql_fieldtable](#)

See Also

[mysql_list_tables](#)

22.9.2.5.24. [mysql_field_type](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_field_type](#)

Get the type of the specified field in a result



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_fetch_field_direct](#) [type]
[PDOStatement::getColumnMeta](#) [driver:decl_type] or [pdo_type]

Description

```
string mysql_field_type(
    resource result,
    int field_offset);
```

[mysql_field_type](#) is similar to the [mysql_field_name](#) function. The arguments are identical, but the field type is returned instead.

Parameters

result

The result resource that is being evaluated. This result comes from a call to [mysql_query](#).

field_offset

The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level [E_WARNING](#) is also issued.

Return Values

The returned field type will be one of "int", "real", "string", "blob", and others as detailed in the [MySQL documentation](#).

Examples

Example 22.52. `mysql_field_type` example

```

<?php
mysql_connect("localhost", "mysql_username", "mysql_password");
mysql_select_db("mysql");
$result = mysql_query("SELECT * FROM func");
$fields = mysql_num_fields($result);
$rows   = mysql_num_rows($result);
$table  = mysql_field_table($result, 0);
echo "Your '" . $table . "' table has " . $fields . " fields and " . $rows . " record(s)\n";
echo "The table has the following fields:\n";
for ($i=0; $i < $fields; $i++) {
    $type = mysql_field_type($result, $i);
    $name = mysql_field_name($result, $i);
    $len  = mysql_field_len($result, $i);
    $flags = mysql_field_flags($result, $i);
    echo $type . " " . $name . " " . $len . " " . $flags . "\n";
}
mysql_free_result($result);
mysql_close();
?>

```

The above example will output something similar to:

```

Your 'func' table has 4 fields and 1 record(s)
The table has the following fields:
string name 64 not_null primary_key binary
int ret 1 not_null
string dl 128 not_null
string type 9 not_null enum

```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
[mysql_fieldtype](#)

See Also

[mysql_field_name](#)
[mysql_field_len](#)

22.9.2.5.25. `mysql_free_result`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_free_result](#)

Free result memory



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL](#):

[choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_free_result](#)

Assign the value of `NULL` to the PDO object, or `PDOStatement::closeCursor`

Description

```
bool mysql_free_result(  
    resource result);
```

`mysql_free_result` will free all memory associated with the result identifier `result`.

`mysql_free_result` only needs to be called if you are concerned about how much memory is being used for queries that return large result sets. All associated result memory is automatically freed at the end of the script's execution.

Parameters

`result` The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

If a non-resource is used for the `result`, an error of level `E_WARNING` will be emitted. It's worth noting that `mysql_query` only returns a resource for `SELECT`, `SHOW`, `EXPLAIN`, and `DESCRIBE` queries.

Examples

Example 22.53. A `mysql_free_result` example

```
<?php  
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");  
if (!$result) {  
    echo 'Could not run query: ' . mysql_error();  
    exit;  
}  
/* Use the result, assuming we're done with it afterwards */  
$row = mysql_fetch_assoc($result);  
  
/* Now we free up the result and continue on with our script */  
mysql_free_result($result);  
  
echo $row['id'];  
echo $row['email'];  
?>
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
`mysql_freeresult`

See Also

[mysql_query](#)

[is_resource](#)

22.9.2.5.26. [mysql_get_client_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_get_client_info](#)

Get MySQL client info



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_get_client_info  
PDO::getAttribute(PDO::ATTR_CLIENT_VERSION)
```

Description

```
string mysql_get_client_info();
```

[mysql_get_client_info](#) returns a string that represents the client library version.

Return Values

The MySQL client version.

Examples

Example 22.54. [mysql_get_client_info](#) example

```
<?php  
printf("MySQL client info: %s\n", mysql_get_client_info());  
?>
```

The above example will output something similar to:

```
MySQL client info: 3.23.39
```

See Also

[mysql_get_host_info](#)
[mysql_get_proto_info](#)
[mysql_get_server_info](#)

22.9.2.5.27. [mysql_get_host_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_get_host_info](#)

Get MySQL host info

**Warning**

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_get_host_info  
PDO::getAttribute(PDO::ATTR_CONNECTION_STATUS)
```

Description

```
string mysql_get_host_info(  
    resource link_identifier  
    = =NULL);
```

Describes the type of connection in use for the connection, including the server host name.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns a string describing the type of MySQL connection in use for the connection or [FALSE](#) on failure.

Examples**Example 22.55. [mysql_get_host_info](#) example**

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
if (!$link) {  
    die('Could not connect: ' . mysql_error());  
}  
printf("MySQL host info: %s\n", mysql_get_host_info());  
?>
```

The above example will output something similar to:

```
MySQL host info: Localhost via UNIX socket
```

See Also

[mysql_get_client_info](#)
[mysql_get_proto_info](#)
[mysql_get_server_info](#)

22.9.2.5.28. [mysql_get_proto_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_get_proto_info](#)

Get MySQL protocol info



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_get_proto_info](#)

Description

```
int mysql_get_proto_info(  
    resource link_identifier  
    = =NULL);
```

Retrieves the MySQL protocol.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns the MySQL protocol on success or [FALSE](#) on failure.

Examples

Example 22.56. [mysql_get_proto_info](#) example

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
if (!$link) {  
    die('Could not connect: ' . mysql_error());  
}  
printf("MySQL protocol version: %s\n", mysql_get_proto_info());  
?>
```

The above example will output something similar to:

```
MySQL protocol version: 10
```

See Also

[mysql_get_client_info](#)
[mysql_get_host_info](#)
[mysql_get_server_info](#)

22.9.2.5.29. *mysql_get_server_info*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysql_get_server_info*

Get MySQL server info



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_get_server_info  
PDO::getAttribute(PDO::ATTR_SERVER_VERSION)
```

Description

```
string mysql_get_server_info(  
    resource link_identifier  
    = =NULL);
```

Retrieves the MySQL server version.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by *mysql_connect* is assumed. If no such link is found, it will try to create one as if *mysql_connect* was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns the MySQL server version on success or [FALSE](#) on failure.

Examples

Example 22.57. *mysql_get_server_info* example

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
if (!$link) {  
    die('Could not connect: ' . mysql_error());  
}  
printf("MySQL server version: %s\n", mysql_get_server_info());  
?>
```

The above example will output something similar to:

```
MySQL server version: 4.0.1-alpha
```

See Also

```
mysql_get_client_info
mysql_get_host_info
mysql_get_proto_info
phpversion
```

22.9.2.5.30. *mysql_info*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysql_info*

Get information about the most recent query



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_info](#)

Description

```
string mysql_info(
    resource link_identifier
    = =NULL);
```

Returns detailed information about the last query.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns information about the statement on success, or [FALSE](#) on failure. See the example below for which statements provide information, and what the returned value may look like. Statements that are not listed will return [FALSE](#).

Examples

Example 22.58. Relevant MySQL Statements

Statements that return string values. The numbers are only for illustrating purpose; their values will correspond to the query.

```
INSERT INTO ... SELECT ...
String format: Records: 23 Duplicates: 0 Warnings: 0
INSERT INTO ... VALUES (...),(...),(...)...
String format: Records: 37 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...
String format: Records: 42 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE
```

```
String format: Records: 60 Duplicates: 0 Warnings: 0
UPDATE
String format: Rows matched: 65 Changed: 65 Warnings: 0
```

Notes



Note

`mysql_info` returns a non-`FALSE` value for the `INSERT ... VALUES` statement only if multiple value lists are specified in the statement.

See Also

`mysql_affected_rows`
`mysql_insert_id`
`mysql_stat`

22.9.2.5.31. `mysql_insert_id`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_insert_id`

Get the ID generated in the last query



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_insert_id
PDO::lastInsertId
```

Description

```
int mysql_insert_id(
    resource link_identifier
    = =NULL);
```

Retrieves the ID generated for an `AUTO_INCREMENT` column by the previous query (usually `INSERT`).

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

The ID generated for an `AUTO_INCREMENT` column by the previous query on success, `0` if the previous query does not generate an `AUTO_INCREMENT` value, or `FALSE` if no MySQL connection was established.

Examples

Example 22.59. `mysql_insert_id` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('mydb');

mysql_query("INSERT INTO mytable (product) values ('kossu')");
printf("Last inserted record has id %d\n", mysql_insert_id());
?>
```

Notes



Caution

`mysql_insert_id` will convert the return type of the native MySQL C API function `mysql_insert_id()` to a type of `long` (named `int` in PHP). If your `AUTO_INCREMENT` column has a column type of `BIGINT` (64 bits) the conversion may result in an incorrect value. Instead, use the internal MySQL SQL function `LAST_INSERT_ID()` in an SQL query. For more information about PHP's maximum integer values, please see the [integer](#) documentation.



Note

Because `mysql_insert_id` acts on the last performed query, be sure to call `mysql_insert_id` immediately after the query that generates the value.



Note

The value of the MySQL SQL function `LAST_INSERT_ID()` always contains the most recently generated `AUTO_INCREMENT` value, and is not reset between queries.

See Also

[mysql_query](#)
[mysql_info](#)

22.9.2.5.32. `mysql_list_dbs`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_list_dbs](#)

List databases available on a MySQL server



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

SQL Query: `SHOW DATABASES`**Description**

```
resource mysql_list_dbs(  
    resource link_identifier  
    = =NULL);
```

Returns a result pointer containing the databases available from the current mysql daemon.

**Warning**

This function has been *DEPRECATED* as of PHP 5.4.0. Relying on this function is highly discouraged.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a result pointer resource on success, or `FALSE` on failure. Use the `mysql_tablename` function to traverse this result pointer, or any function for result tables, such as `mysql_fetch_array`.

Examples**Example 22.60. `mysql_list_dbs` example**

```
<?php  
// Usage without mysql_list_dbs()  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
$res = mysql_query("SHOW DATABASES");  
  
while ($row = mysql_fetch_assoc($res)) {  
    echo $row['Database'] . "\n";  
}  
  
// Deprecated as of PHP 5.4.0  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
$db_list = mysql_list_dbs($link);  
  
while ($row = mysql_fetch_object($db_list)) {  
    echo $row->Database . "\n";  
}  
?>
```

The above example will output something similar to:

```
database1  
database2  
database3
```

Notes**Note**

For backward compatibility, the following deprecated alias may be used:

`mysql_listdbs`

See Also

`mysql_db_name`

`mysql_select_db`

22.9.2.5.33. `mysql_list_fields`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_list_fields`

List MySQL table fields

**Warning**

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

SQL Query: `SHOW COLUMNS FROM sometable`

Description

```
resource mysql_list_fields(
    string database_name,
    string table_name,
    resource link_identifier
    = =NULL);
```

Retrieves information about the given table name.

This function is deprecated. It is preferable to use `mysql_query` to issue an SQL `SHOW COLUMNS FROM table [LIKE 'name']` statement instead.

Parameters

database_name

The name of the database that's being queried.

table_name

The name of the table that's being queried.

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

A result pointer resource on success, or `FALSE` on failure.

The returned result can be used with [mysql_field_flags](#), [mysql_field_len](#), [mysql_field_name](#) and [mysql_field_type](#).

Examples

Example 22.61. Alternate to deprecated [mysql_list_fields](#)

```
<?php
$result = mysql_query("SHOW COLUMNS FROM sometable");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
if (mysql_num_rows($result) > 0) {
    while ($row = mysql_fetch_assoc($result)) {
        print_r($row);
    }
}
?>
```

The above example will output something similar to:

```
Array
(
    [Field] => id
    [Type] => int(7)
    [Null] =>
    [Key] => PRI
    [Default] =>
    [Extra] => auto_increment
)
Array
(
    [Field] => email
    [Type] => varchar(100)
    [Null] =>
    [Key] =>
    [Default] =>
    [Extra] =>
)
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
[mysql_listfields](#)

See Also

[mysql_field_flags](#)
[mysql_info](#)

22.9.2.5.34. [mysql_list_processes](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_list_processes](#)

List MySQL processes



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_thread_id](#)

Description

```
resource mysql_list_processes(  
    resource link_identifier  
    = =NULL);
```

Retrieves the current MySQL server threads.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

A result pointer resource on success or [FALSE](#) on failure.

Examples

Example 22.62. [mysql_list_processes](#) example

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
  
$result = mysql_list_processes($link);  
while ($row = mysql_fetch_assoc($result)){  
    printf("%s %s %s %s %s\n", $row["Id"], $row["Host"], $row["db"],  
        $row["Command"], $row["Time"]);  
}  
mysql_free_result($result);  
?>
```

The above example will output something similar to:

```
1 localhost test Processlist 0  
4 localhost mysql sleep 5
```

See Also

[mysqli_thread_id](#)
[mysqli_stat](#)

22.9.2.5.35. *mysql_list_tables*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysql_list_tables*

List tables in a MySQL database



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

SQL Query: `SHOW TABLES FROM sometable`

Description

```
resource mysql_list_tables(
    string database,
    resource link_identifier
    = =NULL);
```

Retrieves a list of table names from a MySQL database.

This function is deprecated. It is preferable to use [mysql_query](#) to issue an SQL `SHOW TABLES [FROM db_name] [LIKE 'pattern']` statement instead.

Parameters

database

The name of the database

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

A result pointer resource on success or [FALSE](#) on failure.

Use the [mysql_tablename](#) function to traverse this result pointer, or any function for result tables, such as [mysql_fetch_array](#).

Changelog

Version	Description
4.3.7	This function became deprecated.

Examples

Example 22.63. *mysql_list_tables* alternative example

```
<?php
$dbname = 'mysql_dbname';
```

```
if (!mysql_connect('mysql_host', 'mysql_user', 'mysql_password')) {
    echo 'Could not connect to mysql';
    exit;
}

$sql = "SHOW TABLES FROM $dbname";
$result = mysql_query($sql);

if (!$result) {
    echo "DB Error, could not list tables\n";
    echo 'MySQL Error: ' . mysql_error();
    exit;
}

while ($row = mysql_fetch_row($result)) {
    echo "Table: {$row[0]}\n";
}

mysql_free_result($result);
?>
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
[mysql_listtables](#)

See Also

[mysql_list_dbs](#)
[mysql_tablename](#)

22.9.2.5.36. [mysql_num_fields](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_num_fields](#)

Get number of fields in result



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_field_count](#)
[PDOStatement::columnCount](#)

Description

```
int mysql_num_fields(
    resource result);
```

Retrieves the number of fields from a query.

Parameters

`result`

The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

Returns the number of fields in the result set resource on success or `FALSE` on failure.

Examples

Example 22.64. A `mysql_num_fields` example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}

/* returns 2 because id,email === two fields */
echo mysql_num_fields($result);
?>
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
`mysql_numfields`

See Also

`mysql_select_db`
`mysql_query`
`mysql_fetch_field`
`mysql_num_rows`

22.9.2.5.37. `mysql_num_rows`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_num_rows`

Get number of rows in result



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API guide](#) and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_stmt_num_rows`
`PDOStatement::rowCount`

Description

```
int mysql_num_rows(
```


**Warning**

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_connect` with `p:` host prefix
`PDO::__construct` with `PDO::ATTR_PERSISTENT` as a driver option

Description

```
resource mysql_pconnect(
    string server
        = =ini_get("mysql.default_host"),
    string username
        = =ini_get("mysql.default_user"),
    string password
        = =ini_get("mysql.default_password"),
    int client_flags
        = =0);
```

Establishes a persistent connection to a MySQL server.

`mysql_pconnect` acts very much like `mysql_connect` with two major differences.

First, when connecting, the function would first try to find a (persistent) link that's already open with the same host, username and password. If one is found, an identifier for it will be returned instead of opening a new connection.

Second, the connection to the SQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (`mysql_close` will not close links established by `mysql_pconnect`).

This type of link is therefore called 'persistent'.

Parameters

<i>server</i>	The MySQL server. It can also include a port number. e.g. "hostname:port" or a path to a local socket e.g. ":/path/to/socket" for the localhost. If the PHP directive <code>mysql.default_host</code> is undefined (default), then the default value is 'localhost:3306'
<i>username</i>	The username. Default value is the name of the user that owns the server process.
<i>password</i>	The password. Default value is an empty password.
<i>client_flags</i>	The <i>client_flags</i> parameter can be a combination of the following constants: 128 (enable <code>LOAD DATA LOCAL</code> handling), <code>MYSQL_CLIENT_SSL</code> , <code>MYSQL_CLIENT_COMPRESS</code> , <code>MYSQL_CLIENT_IGNORE_SPACE</code> or <code>MYSQL_CLIENT_INTERACTIVE</code> .

Return Values

Returns a MySQL persistent link identifier on success, or `FALSE` on failure.

Changelog

Version	Description
5.5.0	This function will generate an <code>E_DEPRECATED</code> error.
4.3.0	Added the <code>client_flags</code> parameter.

Notes



Note

Note, that these kind of links only work if you are using a module version of PHP. See the [Persistent Database Connections](#) section for more information.



Warning

Using persistent connections can require a bit of tuning of your Apache and MySQL configurations to ensure that you do not exceed the number of connections allowed by MySQL.



Note

You can suppress the error message on failure by prepending a `@` to the function name.

See Also

[mysql_connect](#)
[Persistent Database Connections](#)

22.9.2.5.39. [mysql_ping](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_ping](#)

Ping a server connection or reconnect if there is no connection



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_ping](#)

Description

```
bool mysql_ping(  
    resource link_identifier  
    = =NULL);
```

Checks whether or not the connection to the server is working. If it has gone down, an automatic reconnection is attempted. This function can be used by scripts that remain idle for a long while, to check whether or not the server has closed the connection and reconnect if necessary.



Note

Automatic reconnection is disabled by default in versions of MySQL \geq 5.0.3.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns [TRUE](#) if the connection to the server MySQL server is working, otherwise [FALSE](#) .

Examples

Example 22.66. A [mysql_ping](#) example

```
<?php
set_time_limit(0);

$conn = mysql_connect('localhost', 'mysqluser', 'mypass');
$db   = mysql_select_db('mydb');

/* Assuming this query will take a long time */
$result = mysql_query($sql);
if (!$result) {
    echo 'Query #1 failed, exiting.';
    exit;
}

/* Make sure the connection is still alive, if not, try to reconnect */
if (!mysql_ping($conn)) {
    echo 'Lost connection, exiting after query #1';
    exit;
}
mysql_free_result($result);

/* So the connection is still alive, let's run another query */
$result2 = mysql_query($sql2);
?>
```

See Also

[mysql_thread_id](#)
[mysql_list_processes](#)

22.9.2.5.40. [mysql_query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_query](#)

Send a MySQL query



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_query`
`PDO::query`

Description

```
resource mysqli_query(
    string query,
    resource link_identifier
    = =NULL);
```

`mysqli_query` sends a unique query (multiple queries are not supported) to the currently active database on the server that's associated with the specified `link_identifier`.

Parameters

<code>query</code>	An SQL query The query string should not end with a semicolon. Data inside the query should be properly escaped .
<code>link_identifier</code>	The MySQL connection. If the link identifier is not specified, the last link opened by <code>mysqli_connect</code> is assumed. If no such link is found, it will try to create one as if <code>mysqli_connect</code> was called with no arguments. If no connection is found or established, an <code>E_WARNING</code> level error is generated.

Return Values

For SELECT, SHOW, DESCRIBE, EXPLAIN and other statements returning resultset, `mysqli_query` returns a resource on success, or `FALSE` on error.

For other type of SQL statements, INSERT, UPDATE, DELETE, DROP, etc, `mysqli_query` returns `TRUE` on success or `FALSE` on error.

The returned result resource should be passed to `mysqli_fetch_array`, and other functions for dealing with result tables, to access the returned data.

Use `mysqli_num_rows` to find out how many rows were returned for a SELECT statement or `mysqli_affected_rows` to find out how many rows were affected by a DELETE, INSERT, REPLACE, or UPDATE statement.

`mysqli_query` will also fail and return `FALSE` if the user does not have permission to access the table(s) referenced by the query.

Examples

Example 22.67. Invalid Query

The following query is syntactically invalid, so `mysqli_query` fails and returns `FALSE`.

```
<?php
$result = mysqli_query('SELECT * WHERE 1=1');
if (!$result) {
    die('Invalid query: ' . mysqli_error());
}
?>
```

Example 22.68. Valid Query

The following query is valid, so `mysql_query` returns a resource.

```
<?php
// This could be supplied by a user, for example
$firstname = 'fred';
$lastname  = 'fox';

// Formulate Query
// This is the best way to perform an SQL query
// For more examples, see mysql_real_escape_string()
$query = sprintf("SELECT firstname, lastname, address, age FROM friends
    WHERE firstname='%s' AND lastname='%s'",
    mysql_real_escape_string($firstname),
    mysql_real_escape_string($lastname));

// Perform Query
$result = mysql_query($query);

// Check result
// This shows the actual query sent to MySQL, and the error. Useful for debugging.
if (!$result) {
    $message = 'Invalid query: ' . mysql_error() . "\n";
    $message .= 'Whole query: ' . $query;
    die($message);
}

// Use result
// Attempting to print $result won't allow access to information in the resource
// One of the mysql result functions must be used
// See also mysql_result(), mysql_fetch_array(), mysql_fetch_row(), etc.
while ($row = mysql_fetch_assoc($result)) {
    echo $row['firstname'];
    echo $row['lastname'];
    echo $row['address'];
    echo $row['age'];
}

// Free the resources associated with the result set
// This is done automatically at the end of the script
mysql_free_result($result);
?>
```

See Also

[mysql_connect](#)
[mysql_error](#)
[mysql_real_escape_string](#)
[mysql_result](#)
[mysql_fetch_assoc](#)
[mysql_unbuffered_query](#)

22.9.2.5.41. [mysql_real_escape_string](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_real_escape_string](#)

Escapes special characters in a string for use in an SQL statement

**Warning**

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_real_escape_string
PDO::quote
```

Description

```
string mysql_real_escape_string(
    string unescaped_string,
    resource link_identifier
    = =NULL);
```

Escapes special characters in the *unescaped_string*, taking into account the current character set of the connection so that it is safe to place it in a *mysql_query*. If binary data is to be inserted, this function must be used.

mysql_real_escape_string calls MySQL's library function *mysql_real_escape_string*, which prepends backslashes to the following characters: `\x00`, `\n`, `\r`, `\`, `'`, `"` and `\x1a`.

This function must always (with few exceptions) be used to make data safe before sending a query to MySQL.

**Security: the default character set**

The character set must be set either at the server level, or with the API function *mysql_set_charset* for it to affect *mysql_real_escape_string*. See the concepts section on [character sets](#) for more information.

Parameters

unescaped_string

The string that is to be escaped.

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by *mysql_connect* is assumed. If no such link is found, it will try to create one as if *mysql_connect* was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the escaped string, or `FALSE` on error.

Examples**Example 22.69. Simple *mysql_real_escape_string* example**

```
<?php
// Connect
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    OR die(mysql_error());

// Query
$query = sprintf("SELECT * FROM users WHERE user='%s' AND password='%s'",
    mysql_real_escape_string($user),
```

```
mysql_real_escape_string($password);  
?>
```

Example 22.70. An example SQL Injection Attack

```
<?php  
// We didn't check $_POST['password'], it could be anything the user wanted! For example:  
$_POST['username'] = 'aidan';  
$_POST['password'] = "' OR ''='";  
  
// Query database to check if there are any matching users  
$query = "SELECT * FROM users WHERE user='{$_POST['username']}' AND password='{$_POST['password']}'";  
mysql_query($query);  
  
// This means the query sent to MySQL would be:  
echo $query;  
?>
```

The query sent to MySQL:

```
SELECT * FROM users WHERE user='aidan' AND password='' OR ''='
```

This would allow anyone to log in without a valid password.

Notes



Note

A MySQL connection is required before using `mysql_real_escape_string` otherwise an error of level `E_WARNING` is generated, and `FALSE` is returned. If `link_identifier` isn't defined, the last MySQL connection is used.



Note

If `magic_quotes_gpc` is enabled, first apply `stripslashes` to the data. Using this function on data which has already been escaped will escape the data twice.



Note

If this function is not used to escape data, the query is vulnerable to [SQL Injection Attacks](#).



Note

`mysql_real_escape_string` does not escape `%` and `_`. These are wildcards in MySQL if combined with `LIKE`, `GRANT`, or `REVOKE`.

See Also

[mysql_set_charset](#)
[mysql_client_encoding](#)
[addslashes](#)
[stripslashes](#)
The [magic_quotes_gpc](#) directive

The `magic_quotes_runtime` directive

22.9.2.5.42. `mysql_result`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysql_result`

Get result data



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_data_seek` in conjunction with `mysqli_field_seek` and `mysqli_fetch_field`
`PDOStatement::fetchColumn`

Description

```
string mysql_result(  
    resource result,  
    int row,  
    mixed field  
    = =0);
```

Retrieves the contents of one cell from a MySQL result set.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than `mysql_result`. Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Parameters

<i>result</i>	The result resource that is being evaluated. This result comes from a call to <code>mysql_query</code> .
<i>row</i>	The row number from the result that's being retrieved. Row numbers start at 0.
<i>field</i>	The name or offset of the field being retrieved. It can be the field's offset, the field's name, or the field's table dot field name (tablename.fieldname). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name. If undefined, the first field is retrieved.

Return Values

The contents of one cell from a MySQL result set on success, or `FALSE` on failure.

Examples

Example 22.71. `mysql_result` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
if (!mysql_select_db('database_name')) {
    die('Could not select database: ' . mysql_error());
}
$result = mysql_query('SELECT name FROM work.employee');
if (!$result) {
    die('Could not query:' . mysql_error());
}
echo mysql_result($result, 2); // outputs third employee's name

mysql_close($link);
?>
```

Notes



Note

Calls to `mysql_result` should not be mixed with calls to other functions that deal with the result set.

See Also

[mysql_fetch_row](#)
[mysql_fetch_array](#)
[mysql_fetch_assoc](#)
[mysql_fetch_object](#)

22.9.2.5.43. [mysql_select_db](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_select_db](#)

Select a MySQL database



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_select_db](#)
[PDO::__construct](#) (part of dsn)

Description

```
bool mysql_select_db(
    string database_name,
    resource link_identifier
    = =NULL);
```

Sets the current active database on the server that's associated with the specified link identifier. Every subsequent call to `mysql_query` will be made on the active database.

Parameters

<i>database_name</i>	The name of the database that is to be selected.
<i>link_identifier</i>	The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect is assumed. If no such link is found, it will try to create one as if mysql_connect was called with no arguments. If no connection is found or established, an E_WARNING level error is generated.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.72. [mysql_select_db](#) example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Not connected : ' . mysql_error());
}

// make foo the current db
$db_selected = mysql_select_db('foo', $link);
if (!$db_selected) {
    die ('Can\'t use foo : ' . mysql_error());
}
?>
```

Notes



Note

For backward compatibility, the following deprecated alias may be used:
[mysql_selectdb](#)

See Also

[mysql_connect](#)
[mysql_pconnect](#)
[mysql_query](#)

22.9.2.5.44. [mysql_set_charset](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_set_charset](#)

Sets the client character set



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_character_set_name](#)PDO: Add `charset` to the connection string, such as `charset=utf8`

Description

```
bool mysql_set_charset(  
    string charset,  
    resource link_identifier  
    = =NULL);
```

Sets the default character set for the current connection.

Parameters

charset

A valid character set name.

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Notes



Note

This function requires MySQL 5.0.7 or later.



Note

This is the preferred way to change the charset. Using [mysql_query](#) to set it (such as `SET NAMES utf8`) is not recommended. See the [MySQL character set concepts](#) section for more information.

See Also

[mysql_client_encoding](#)

[Setting character sets in MySQL](#)

[List of character sets that MySQL supports](#)

22.9.2.5.45. [mysql_stat](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_stat](#)

Get current system status



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_stat
PDO::getAttribute(PDO::ATTR_SERVER_INFO)
```

Description

```
string mysqli_stat(
    resource link_identifier
    = =NULL);
```

`mysqli_stat` returns the current server status.

Parameters

link_identifier The MySQL connection. If the link identifier is not specified, the last link opened by `mysqli_connect` is assumed. If no such link is found, it will try to create one as if `mysqli_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a string with the status for uptime, threads, queries, open tables, flush tables and queries per second. For a complete list of other status variables, you have to use the `SHOW STATUS` SQL command. If *link_identifier* is invalid, `NULL` is returned.

Examples

Example 22.73. `mysqli_stat` example

```
<?php
$link = mysqli_connect('localhost', 'mysql_user', 'mysql_password');
$status = explode(' ', mysqli_stat($link));
print_r($status);
?>
```

The above example will output something similar to:

```
Array
(
    [0] => Uptime: 5380
    [1] => Threads: 2
    [2] => Questions: 1321299
    [3] => Slow queries: 0
    [4] => Opens: 26
    [5] => Flush tables: 1
    [6] => Open tables: 17
    [7] => Queries per second avg: 245.595
)
```

Example 22.74. Alternative `mysqli_stat` example

```
<?php
$link = mysqli_connect('localhost', 'mysql_user', 'mysql_password');
$result = mysqli_query('SHOW STATUS', $link);
while ($row = mysqli_fetch_assoc($result)) {
```

```
    echo $row['Variable_name'] . ' = ' . $row['Value'] . "\n";
}
?>
```

The above example will output something similar to:

```
back_log = 50
basedir = /usr/local/
bdb_cache_size = 8388600
bdb_log_buffer_size = 32768
bdb_home = /var/db/mysql/
bdb_max_lock = 10000
bdb_logdir =
bdb_shared_data = OFF
bdb_tmpdir = /var/tmp/
...
```

See Also

[mysql_get_server_info](#)
[mysql_list_processes](#)

22.9.2.5.46. [mysql_tablename](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_tablename](#)

Get table name of field



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

SQL Query: `SHOW TABLES`

Description

```
string mysql_tablename(
    resource result,
    int i);
```

Retrieves the table name from a *result*.

This function is deprecated. It is preferable to use [mysql_query](#) to issue an SQL `SHOW TABLES [FROM db_name] [LIKE 'pattern']` statement instead.

Parameters

result A result pointer resource that's returned from [mysql_list_tables](#).
i The integer index (row/table number)

Return Values

The name of the table on success or `FALSE` on failure.

Use the `mysql_tablename` function to traverse this result pointer, or any function for result tables, such as `mysql_fetch_array`.

Changelog

Version	Description
5.5.0	The <code>mysql_tablename</code> function is deprecated, and emits an <code>E_DEPRECATED</code> level error.

Examples

Example 22.75. `mysql_tablename` example

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password");
$result = mysql_list_tables("mydb");
$num_rows = mysql_num_rows($result);
for ($i = 0; $i < $num_rows; $i++) {
    echo "Table: ", mysql_tablename($result, $i), "\n";
}

mysql_free_result($result);
?>
```

Notes



Note

The `mysql_num_rows` function may be used to determine the number of tables in the result pointer.

See Also

[mysql_list_tables](#)
[mysql_field_table](#)
[mysql_db_name](#)

22.9.2.5.47. `mysql_thread_id`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_thread_id](#)

Return the current thread ID



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_thread_id](#)

Description

```
int mysql_thread_id(
    resource link_identifier
    = =NULL);
```

Retrieves the current thread ID. If the connection is lost, and a reconnect with [mysql_ping](#) is executed, the thread ID will change. This means only retrieve the thread ID when needed.

Parameters

[link_identifier](#)

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) was called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

The thread ID on success or [FALSE](#) on failure.

Examples

Example 22.76. [mysql_thread_id](#) example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$thread_id = mysql_thread_id($link);
if ($thread_id){
    printf("current thread id is %d\n", $thread_id);
}
?>
```

The above example will output something similar to:

```
current thread id is 73
```

See Also

[mysql_ping](#)
[mysql_list_processes](#)

22.9.2.5.48. [mysql_unbuffered_query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysql_unbuffered_query](#)

Send an SQL query to MySQL without fetching and buffering the result rows.



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

See: [Buffered and Unbuffered queries](#)

Description

```
resource mysqli_unbuffered_query(  
    string query,  
    resource link_identifier  
    = =NULL);
```

`mysqli_unbuffered_query` sends the SQL query *query* to MySQL without automatically fetching and buffering the result rows as `mysqli_query` does. This saves a considerable amount of memory with SQL queries that produce large result sets, and you can start working on the result set immediately after the first row has been retrieved as you don't have to wait until the complete SQL query has been performed. To use `mysqli_unbuffered_query` while multiple database connections are open, you must specify the optional parameter *link_identifier* to identify which connection you want to use.

Parameters

query

The SQL query to execute.

Data inside the query should be [properly escaped](#).

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysqli_connect` is assumed. If no such link is found, it will try to create one as if `mysqli_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

For SELECT, SHOW, DESCRIBE or EXPLAIN statements, `mysqli_unbuffered_query` returns a resource on success, or `FALSE` on error.

For other type of SQL statements, UPDATE, DELETE, DROP, etc, `mysqli_unbuffered_query` returns `TRUE` on success or `FALSE` on error.

Notes



Note

The benefits of `mysqli_unbuffered_query` come at a cost: you cannot use `mysqli_num_rows` and `mysqli_data_seek` on a result set returned from `mysqli_unbuffered_query`, until all rows are fetched. You also have to fetch all result rows from an unbuffered SQL query before you can send a new SQL query to MySQL, using the same *link_identifier*.

See Also

[mysqli_query](#)

22.9.3. MySQL Improved Extension (*Mysqli*)

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqli` extension allows you to access the functionality provided by MySQL 4.1 and above. More information about the MySQL Database server can be found at <http://www.mysql.com/>

An overview of software available for using MySQL from PHP can be found at [Section 22.9.3.2, “Overview”](#)

Documentation for MySQL can be found at <http://dev.mysql.com/doc/>.

Parts of this documentation included from MySQL manual with permissions of Oracle Corporation.

22.9.3.1. Examples

Copyright 1997-2012 the PHP Documentation Group. [3186]

All examples in the `mysqli` documentation use the world database. The world database can be found at <http://downloads.mysql.com/docs/world.sql.gz>

22.9.3.2. Overview

Copyright 1997-2012 the PHP Documentation Group. [3186]

This section provides an introduction to the options available to you when developing a PHP application that needs to interact with a MySQL database.

What is an API?

An Application Programming Interface, or API, defines the classes, methods, functions and variables that your application will need to call in order to carry out its desired task. In the case of PHP applications that need to communicate with databases the necessary APIs are usually exposed via PHP extensions.

APIs can be procedural or object-oriented. With a procedural API you call functions to carry out tasks, with the object-oriented API you instantiate classes and then call methods on the resulting objects. Of the two the latter is usually the preferred interface, as it is more modern and leads to better organised code.

When writing PHP applications that need to connect to the MySQL server there are several API options available. This document discusses what is available and how to select the best solution for your application.

What is a Connector?

In the MySQL documentation, the term *connector* refers to a piece of software that allows your application to connect to the MySQL database server. MySQL provides connectors for a variety of languages, including PHP.

If your PHP application needs to communicate with a database server you will need to write PHP code to perform such activities as connecting to the database server, querying the database and other database-related functions. Software is required to provide the API that your PHP application will use, and also handle the communication between your application and the database server, possibly using other intermediate libraries where necessary. This software is known generically as a connector, as it allows your application to *connect* to a database server.

What is a Driver?

A driver is a piece of software designed to communicate with a specific type of database server. The driver may also call a library, such as the MySQL Client Library or the MySQL Native Driver. These libraries implement the low-level protocol used to communicate with the MySQL database server.

By way of an example, the [PHP Data Objects \(PDO\) \[3270\]](#) database abstraction layer may use one of several database-specific drivers. One of the drivers it has available is the PDO MYSQL driver, which allows it to interface with the MySQL server.

Sometimes people use the terms connector and driver interchangeably, this can be confusing. In the MySQL-related documentation the term “driver” is reserved for software that provides the database-specific part of a connector package.

What is an Extension?

In the PHP documentation you will come across another term - *extension*. The PHP code consists of a core, with optional extensions to the core functionality. PHP's MySQL-related extensions, such as the `mysqli` extension, and the `mysql` extension, are implemented using the PHP extension framework.

An extension typically exposes an API to the PHP programmer, to allow its facilities to be used programmatically. However, some extensions which use the PHP extension framework do not expose an API to the PHP programmer.

The PDO MySQL driver extension, for example, does not expose an API to the PHP programmer, but provides an interface to the PDO layer above it.

The terms API and extension should not be taken to mean the same thing, as an extension may not necessarily expose an API to the programmer.

What are the main PHP API offerings for using MySQL?

There are three main API options when considering connecting to a MySQL database server:

- PHP's MySQL Extension
- PHP's `mysqli` Extension
- PHP Data Objects (PDO)

Each has its own advantages and disadvantages. The following discussion aims to give a brief introduction to the key aspects of each API.

What is PHP's MySQL Extension?

This is the original extension designed to allow you to develop PHP applications that interact with a MySQL database. The `mysql` extension provides a procedural interface and is intended for use only with MySQL versions older than 4.1.3. This extension can be used with versions of MySQL 4.1.3 or newer, but not all of the latest MySQL server features will be available.



Note

If you are using MySQL versions 4.1.3 or later it is *strongly* recommended that you use the `mysqli` extension instead.

The `mysql` extension source code is located in the PHP extension directory `ext/mysql`.

For further information on the `mysql` extension, see [Section 22.9.2, “Original MySQL API \(`Mysql`\)”](#).

What is PHP's `mysqli` Extension?

The `mysqli` extension, or as it is sometimes known, the MySQL *improved* extension, was developed to take advantage of new features found in MySQL systems versions 4.1.3 and newer. The `mysqli` extension is included with PHP versions 5 and later.

The `mysqli` extension has a number of benefits, the key enhancements over the `mysql` extension being:

- Object-oriented interface

- Support for Prepared Statements
- Support for Multiple Statements
- Support for Transactions
- Enhanced debugging capabilities
- Embedded server support



Note

If you are using MySQL versions 4.1.3 or later it is *strongly* recommended that you use this extension.

As well as the object-oriented interface the extension also provides a procedural interface.

The `mysqli` extension is built using the PHP extension framework, its source code is located in the directory `ext/mysqli`.

For further information on the `mysqli` extension, see [Section 22.9.3, “MySQL Improved Extension \(`mysqli`\)”](#).

What is PDO?

PHP Data Objects, or PDO, is a database abstraction layer specifically for PHP applications. PDO provides a consistent API for your PHP application regardless of the type of database server your application will connect to. In theory, if you are using the PDO API, you could switch the database server you used, from say Firebird to MySQL, and only need to make minor changes to your PHP code.

Other examples of database abstraction layers include JDBC for Java applications and DBI for Perl.

While PDO has its advantages, such as a clean, simple, portable API, its main disadvantage is that it doesn't allow you to use all of the advanced features that are available in the latest versions of MySQL server. For example, PDO does not allow you to use MySQL's support for Multiple Statements.

PDO is implemented using the PHP extension framework, its source code is located in the directory `ext/pdo`.

For further information on PDO, see the <http://www.php.net/book.pdo>.

What is the PDO MYSQL driver?

The PDO MYSQL driver is not an API as such, at least from the PHP programmer's perspective. In fact the PDO MYSQL driver sits in the layer below PDO itself and provides MySQL-specific functionality. The programmer still calls the PDO API, but PDO uses the PDO MYSQL driver to carry out communication with the MySQL server.

The PDO MYSQL driver is one of several available PDO drivers. Other PDO drivers available include those for the Firebird and PostgreSQL database servers.

The PDO MYSQL driver is implemented using the PHP extension framework. Its source code is located in the directory `ext/pdo_mysql`. It does not expose an API to the PHP programmer.

For further information on the PDO MYSQL driver, see [Section 22.9.4, “MySQL Functions \(PDO_MYSQL\) \(MySQL \(PDO\)\)”](#).

What is PHP's MySQL Native Driver?

In order to communicate with the MySQL database server the `mysql` extension, `mysqli` and the PDO MySQL driver each use a low-level library that implements the required protocol. In the past, the only available library was the MySQL Client Library, otherwise known as `libmysql`.

However, the interface presented by `libmysql` was not optimized for communication with PHP applications, as `libmysql` was originally designed with C applications in mind. For this reason the MySQL Native Driver, `mysqlnd`, was developed as an alternative to `libmysql` for PHP applications.

The `mysql` extension, the `mysqli` extension and the PDO MySQL driver can each be individually configured to use either `libmysql` or `mysqlnd`. As `mysqlnd` is designed specifically to be utilised in the PHP system it has numerous memory and speed enhancements over `libmysql`. You are strongly encouraged to take advantage of these improvements.



Note

The MySQL Native Driver can only be used with MySQL server versions 4.1.3 and later.

The MySQL Native Driver is implemented using the PHP extension framework. The source code is located in `ext/mysqlnd`. It does not expose an API to the PHP programmer.

Comparison of Features

The following table compares the functionality of the three main methods of connecting to MySQL from PHP:

Table 22.36. Comparison of MySQL API options for PHP

	PHP's <code>mysqli</code> Extension	PDO (Using PDO MySQL Driver and MySQL Native Driver)	PHP's MySQL Extension
PHP version introduced	5.0	5.0	Prior to 3.0
Included with PHP 5.x	yes	yes	Yes
MySQL development status	Active development	Active development as of PHP 5.3	Maintenance only
Recommended by MySQL for new projects	Yes - preferred option	Yes	No
API supports Charsets	Yes	Yes	No
API supports server-side Prepared Statements	Yes	Yes	No
API supports client-side Prepared Statements	No	Yes	No
API supports Stored Procedures	Yes	Yes	No
API supports Multiple Statements	Yes	Most	No
Supports all MySQL 4.1+ functionality	Yes	Most	No

22.9.3.3. Quick start guide

Copyright 1997-2012 the PHP Documentation Group. [3186]

This quick start guide will help with choosing and gaining familiarity with the PHP MySQL API.

This quick start gives an overview on the `mysqli` extension. Code examples are provided for all major aspects of the API. Database concepts are explained to the degree needed for presenting concepts specific to MySQL.

Required: A familiarity with the PHP programming language, the SQL language, and basic knowledge of the MySQL server.

22.9.3.3.1. Dual procedural and object-oriented interface

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqli` extension features a dual interface. It supports the procedural and object-oriented programming paradigm.

Users migrating from the old `mysql` extension may prefer the procedural interface. The procedural interface is similar to that of the old `mysql` extension. In many cases, the function names differ only by prefix. Some `mysqli` functions take a connection handle as their first argument, whereas matching functions in the old `mysql` interface take it as an optional last argument.

Example 22.77. Easy migration from the old `mysql` extension

```
<?php
$mysqli = mysqli_connect("example.com", "user", "password", "database");
$res = mysqli_query($mysqli, "SELECT 'Please, do not use ' AS _msg FROM DUAL");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];

$mysql = mysql_connect("localhost", "root", "");
mysql_select_db("test");
$res = mysql_query("SELECT 'the mysql extension for new developments.' AS _msg FROM DUAL", $mysql);
$row = mysql_fetch_assoc($res);
echo $row['_msg'];
?>
```

The above example will output:

```
Please, do not use the mysql extension for new developments.
```

The object-oriented interface

In addition to the classical procedural interface, users can choose to use the object-oriented interface. The documentation is organized using the object-oriented interface. The object-oriented interface shows functions grouped by their purpose, making it easier to get started. The reference section gives examples for both syntax variants.

There are no significant performance differences between the two interfaces. Users can base their choice on personal preference.

Example 22.78. Object-oriented and procedural interface

```
<?php
```

```
$mysqli = mysqli_connect("example.com", "user", "password", "database");
if (mysqli_connect_errno($mysqli)) {
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$res = mysqli_query($mysqli, "SELECT 'A world full of ' AS _msg FROM DUAL");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];

$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: " . $mysqli->connect_error;
}

$res = $mysqli->query("SELECT 'choices to please everybody.' AS _msg FROM DUAL");
$row = $res->fetch_assoc();
echo $row['_msg'];
?>
```

The above example will output:

```
A world full of choices to please everybody.
```

The object oriented interface is used for the quickstart because the reference section is organized that way.

Mixing styles

It is possible to switch between styles at any time. Mixing both styles is not recommended for code clarity and coding style reasons.

Example 22.79. Bad coding style

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: " . $mysqli->connect_error;
}

$res = mysqli_query($mysqli, "SELECT 'Possible but bad style.' AS _msg FROM DUAL");
if (!$res) {
    echo "Failed to run query: (" . $mysqli->errno . ") " . $mysqli->error;
}

if ($row = $res->fetch_assoc()) {
    echo $row['_msg'];
}
?>
```

The above example will output:

```
Possible but bad style.
```

See also

```
mysqli::__construct  
mysqli::query  
mysqli_result::fetch_assoc  
$mysqli::connect_errno  
$mysqli::connect_error  
$mysqli::errno  
$mysqli::error  
The MySQLi Extension Function Summary
```

22.9.3.3.2. Connections

Copyright 1997-2012 the PHP Documentation Group. [3186]

The MySQL server supports the use of different transport layers for connections. Connections use TCP/IP, Unix domain sockets or Windows named pipes.

The hostname `localhost` has a special meaning. It is bound to the use of Unix domain sockets. It is not possible to open a TCP/IP connection using the hostname `localhost` you must use `127.0.0.1` instead.

Example 22.80. Special meaning of localhost

```
<?php  
$mysqli = new mysqli("localhost", "user", "password", "database");  
if ($mysqli->connect_errno) {  
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;  
}  
echo $mysqli->host_info . "\n";  
  
$mysqli = new mysqli("127.0.0.1", "user", "password", "database", 3306);  
if ($mysqli->connect_errno) {  
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;  
}  
  
echo $mysqli->host_info . "\n";  
?>
```

The above example will output:

```
localhost via UNIX socket  
127.0.0.1 via TCP/IP
```

Connection parameter defaults

Depending on the connection function used, assorted parameters can be omitted. If a parameter is not provided, then the extension attempts to use the default values that are set in the PHP configuration file.

Example 22.81. Setting defaults

```
mysqli.default_host=192.168.2.27  
mysqli.default_user=root  
mysqli.default_pw=""  
mysqli.default_port=3306  
mysqli.default_socket=/tmp/mysql.sock
```

The resulting parameter values are then passed to the client library that is used by the extension. If the client library detects empty or unset parameters, then it may default to the library built-in values.

Built-in connection library defaults

If the host value is unset or empty, then the client library will default to a Unix socket connection on `localhost`. If socket is unset or empty, and a Unix socket connection is requested, then a connection to the default socket on `/tmp/mysql.sock` is attempted.

On Windows systems, the host name `.` is interpreted by the client library as an attempt to open a Windows named pipe based connection. In this case the socket parameter is interpreted as the pipe name. If not given or empty, then the socket (pipe name) defaults to `\\.pipe\MySQL`.

If neither a Unix domain socket based not a Windows named pipe based connection is to be established and the port parameter value is unset, the library will default to port `3306`.

The `mysqlnd` library and the MySQL Client Library (`libmysql`) implement the same logic for determining defaults.

Connection options

Connection options are available to, for example, set init commands which are executed upon connect, or for requesting use of a certain charset. Connection options must be set before a network connection is established.

For setting a connection option, the connect operation has to be performed in three steps: creating a connection handle with `mysqli_init`, setting the requested options using `mysqli_options`, and establishing the network connection with `mysqli_real_connect`.

Connection pooling

The `mysqli` extension supports persistent database connections, which are a special kind of pooled connections. By default, every database connection opened by a script is either explicitly closed by the user during runtime or released automatically at the end of the script. A persistent connection is not. Instead it is put into a pool for later reuse, if a connection to the same server using the same username, password, socket, port and default database is opened. Reuse saves connection overhead.

Every PHP process is using its own `mysqli` connection pool. Depending on the web server deployment model, a PHP process may serve one or multiple requests. Therefore, a pooled connection may be used by one or more scripts subsequently.

Persistent connection

If a unused persistent connection for a given combination of host, username, password, socket, port and default database can not be found in the connection pool, then `mysqli` opens a new connection. The use of persistent connections can be enabled and disabled using the PHP directive `mysqli.allow_persistent`. The total number of connections opened by a script can be limited with `mysqli.max_links`. The maximum number of persistent connections per PHP process can be restricted with `mysqli.max_persistent`. Please note, that the web server may spawn many PHP processes.

A common complain about persistent connections is that their state is not reset before reuse. For example, open and unfinished transactions are not automatically rolled back. But also, authorization changes which happened in the time between putting the connection into the pool and reusing it are not reflected. This may be seen as an unwanted side-effect. On the contrary, the name `persistent` may be understood as a promise that the state is persisted.

The `mysqli` extension supports both interpretations of a persistent connection: state persisted, and state reset before reuse. The default is reset. Before a persistent connection is reused, the `mysqli` extension

implicitly calls [mysqli_change_user](#) to reset the state. The persistent connection appears to the user as if it was just opened. No artifacts from previous usages are visible.

The [mysqli_change_user](#) function is an expensive operation. For best performance, users may want to recompile the extension with the compile flag [MYSQLI_NO_CHANGE_USER_ON_PCONNECT](#) being set.

It is left to the user to choose between safe behavior and best performance. Both are valid optimization goals. For ease of use, the safe behavior has been made the default at the expense of maximum performance.

See also

[mysqli::__construct](#)
[mysqli::init](#)
[mysqli::options](#)
[mysqli::real_connect](#)
[mysqli::change_user](#)
[\\$mysqli::host_info](#)
[MySQLi Configuration Options](#)
[Persistent Database Connections](#)

22.9.3.3.3. Executing statements

Copyright 1997-2012 the PHP Documentation Group. [3186]

Statements can be executed with the [mysqli_query](#), [mysqli_real_query](#) and [mysqli_multi_query](#) functions. The [mysqli_query](#) function is the most common, and combines the executing statement with a buffered fetch of its result set, if any, in one call. Calling [mysqli_query](#) is identical to calling [mysqli_real_query](#) followed by [mysqli_store_result](#).

Example 22.82. Connecting to MySQL

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Buffered result sets

After statement execution results can be retrieved at once to be buffered by the client or by read row by row. Client-side result set buffering allows the server to free resources associated with the statement results as early as possible. Generally speaking, clients are slow consuming result sets. Therefore, it is recommended to use buffered result sets. [mysqli_query](#) combines statement execution and result set buffering.

PHP applications can navigate freely through buffered results. Navigation is fast because the result sets are held in client memory. Please, keep in mind that it is often easier to scale by client than it is to scale the server.

Example 22.83. Navigation through buffered results

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

$res = $mysqli->query("SELECT id FROM test ORDER BY id ASC");

echo "Reverse order...\n";
for ($row_no = $res->num_rows - 1; $row_no >= 0; $row_no--) {
    $res->data_seek($row_no);
    $row = $res->fetch_assoc();
    echo " id = " . $row['id'] . "\n";
}

echo "Result set order...\n";
$res->data_seek(0);
while ($row = $res->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
?>
```

The above example will output:

```
Reverse order...
 id = 3
 id = 2
 id = 1
Result set order...
 id = 1
 id = 2
 id = 3
```

Unbuffered result sets

If client memory is a short resource and freeing server resources as early as possible to keep server load low is not needed, unbuffered results can be used. Scrolling through unbuffered results is not possible before all rows have been read.

Example 22.84. Navigation through unbuffered results

```
<?php
$mysqli->real_query("SELECT id FROM test ORDER BY id ASC");
$res = $mysqli->use_result();

echo "Result set order...\n";
while ($row = $res->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
?>
```

Result set values data types

The `mysqli_query`, `mysqli_real_query` and `mysqli_multi_query` functions are used to execute non-prepared statements. At the level of the MySQL Client Server Protocol, the command `COM_QUERY` and the text protocol are used for statement execution. With the text protocol, the MySQL server converts all data of a result sets into strings before sending. This conversion is done regardless of the SQL result set column data type. The `mysqli` client libraries receive all column values as strings. No further client-side casting is done to convert columns back to their native types. Instead, all values are provided as PHP strings.

Example 22.85. Text protocol returns strings by default

```
<?php
mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

$res = $mysqli->query("SELECT id, label FROM test WHERE id = 1");
$row = $res->fetch_assoc();

printf("id = %s (%s)\n", $row['id'], gettype($row['id']));
printf("label = %s (%s)\n", $row['label'], gettype($row['label']));
?>
```

The above example will output:

```
id = 1 (string)
label = a (string)
```

It is possible to convert integer and float columns back to PHP numbers by setting the `MYSQLI_OPT_INT_AND_FLOAT_NATIVE` connection option, if using the `mysqlnd` library. If set, the `mysqlnd` library will check the result set meta data column types and convert numeric SQL columns to PHP numbers, if the PHP data type value range allows for it. This way, for example, SQL INT columns are returned as integers.

Example 22.86. Native data types with `mysqlnd` and connection option

```
<?php
mysqli = mysqli_init();
mysqli->options(MYSQLI_OPT_INT_AND_FLOAT_NATIVE, 1);
mysqli->real_connect("example.com", "user", "password", "database");

if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
```

```

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

$res = $mysqli->query("SELECT id, label FROM test WHERE id = 1");
$row = $res->fetch_assoc();

printf("id = %s (%s)\n", $row['id'], gettype($row['id']));
printf("label = %s (%s)\n", $row['label'], gettype($row['label']));
?>

```

The above example will output:

```

id = 1 (integer)
label = a (string)

```

See also

```

mysqli::__construct
mysqli::init
mysqli::options
mysqli::real_connect
mysqli::query
mysqli::multi_query
mysqli::use_result
mysqli::store_result
mysqli_result::free

```

22.9.3.3.4. Prepared Statements

Copyright 1997-2012 the PHP Documentation Group. [3186]

The MySQL database supports prepared statements. A prepared statement or a parameterized statement is used to execute the same statement repeatedly with high efficiency.

Basic workflow

The prepared statement execution consists of two stages: prepare and execute. At the prepare stage a statement template is send to the database server. The server performs a syntax check and initializes server internal resources for later use.

The MySQL server supports using anonymous, positional placeholder with `?`.

Example 22.87. First stage: prepare

```

<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

/* Non-prepared statement */
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)") {

```

```

    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

/* Prepared statement, stage 1: prepare */
if (!$stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)")) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>

```

Prepare is followed by execute. During execute the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values to execute it using the previously created internal resources.

Example 22.88. Second stage: bind and execute

```

<?php
/* Prepared statement, stage 2: bind and execute */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
    echo "Binding parameters failed: (" . $stmt->errno . ") " . $stmt->error;
}

if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
?>

```

Repeated execution

A prepared statement can be executed repeatedly. Upon every execution the current value of the bound variable is evaluated and send to the server. The statement is not parsed again. The statement template is not transferred to the server again.

Example 22.89. INSERT prepared once, executed multiple times

```

<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

/* Non-prepared statement */
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

/* Prepared statement, stage 1: prepare */
if (!$stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)")) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

/* Prepared statement, stage 2: bind and execute */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
    echo "Binding parameters failed: (" . $stmt->errno . ") " . $stmt->error;
}

if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}

```

```
}

/* Prepared statement: repeated execution, only data transferred from client to server */
for ($id = 2; $id < 5; $id++) {
    if (!$stmt->execute()) {
        echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
    }
}

/* explicit close recommended */
$stmt->close();

/* Non-prepared statement */
$res = $mysqli->query("SELECT id FROM test");
var_dump($res->fetch_all());
?>
```

The above example will output:

```
array(4) {
  [0]=>
  array(1) {
    [0]=>
    string(1) "1"
  }
  [1]=>
  array(1) {
    [0]=>
    string(1) "2"
  }
  [2]=>
  array(1) {
    [0]=>
    string(1) "3"
  }
  [3]=>
  array(1) {
    [0]=>
    string(1) "4"
  }
}
```

Every prepared statement occupies server resources. Statements should be closed explicitly immediately after use. If not done explicitly, the statement will be closed when the statement handle is freed by PHP.

Using a prepared statement is not always the most efficient way of executing a statement. A prepared statement executed only once causes more client-server round-trips than a non-prepared statement. This is why the [SELECT](#) is not run as a prepared statement above.

Also, consider the use of the MySQL multi-INSERT SQL syntax for INSERTs. For the example, multi-INSERT requires less round-trips between the server and client than the prepared statement shown above.

Example 22.90. Less round trips using multi-INSERT SQL

```
<?php
if (!$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3), (4)")) {
    echo "Multi-INSERT failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Result set values data types

The MySQL Client Server Protocol defines a different data transfer protocol for prepared statements and non-prepared statements. Prepared statements are using the so called binary protocol. The MySQL server sends result set data "as is" in binary format. Results are not serialized into strings before sending. The client libraries do not receive strings only. Instead, they will receive binary data and try to convert the values into appropriate PHP data types. For example, results from an SQL `INT` column will be provided as PHP integer variables.

Example 22.91. Native datatypes

```
<?php
mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

$stmt = $mysqli->prepare("SELECT id, label FROM test WHERE id = 1");
$stmt->execute();
$res = $stmt->get_result();
$row = $res->fetch_assoc();

printf("id = %s (%s)\n", $row['id'], gettype($row['id']));
printf("label = %s (%s)\n", $row['label'], gettype($row['label']));
?>
```

The above example will output:

```
id = 1 (integer)
label = a (string)
```

This behavior differs from non-prepared statements. By default, non-prepared statements return all results as strings. This default can be changed using a connection option. If the connection option is used, there are no differences.

Fetching results using bound variables

Results from prepared statements can either be retrieved by binding output variables, or by requesting a `mysqli_result` object.

Output variables must be bound after statement execution. One variable must be bound for every column of the statements result set.

Example 22.92. Output variable binding

```
<?php
```

```
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$stmt = $mysqli->prepare("SELECT id, label FROM test")) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$stmt->execute()) {
    echo "Execute failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

$out_id    = NULL;
$out_label = NULL;
if (!$stmt->bind_result($out_id, $out_label)) {
    echo "Binding output parameters failed: (" . $stmt->errno . ") " . $stmt->error;
}

while ($stmt->fetch()) {
    printf("id = %s (%s), label = %s (%s)\n", $out_id, gettype($out_id), $out_label, gettype($out_label));
}
?>
```

The above example will output:

```
id = 1 (integer), label = a (string)
```

Prepared statements return unbuffered result sets by default. The results of the statement are not implicitly fetched and transferred from the server to the client for client-side buffering. The result set takes server resources until all results have been fetched by the client. Thus it is recommended to consume results timely. If a client fails to fetch all results or the client closes the statement before having fetched all data, the data has to be fetched implicitly by [mysqli](#).

It is also possible to buffer the results of a prepared statement using [mysqli_stmt_store_result](#).

Fetching results using [mysqli_result](#) interface

Instead of using bound results, results can also be retrieved through the [mysqli_result](#) interface. [mysqli_stmt_get_result](#) returns a buffered result set.

Example 22.93. Using [mysqli_result](#) to fetch results

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
```

```

    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$stmt = $mysqli->prepare("SELECT id, label FROM test ORDER BY id ASC")) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}

if (!$res = $stmt->get_result()) {
    echo "Getting result set failed: (" . $stmt->errno . ") " . $stmt->error;
}

var_dump($res->fetch_all());
?>

```

The above example will output:

```

array(1) {
  [0]=>
  array(2) {
    [0]=>
    int(1)
    [1]=>
    string(1) "a"
  }
}

```

Using the [mysqli_result interface](#) offers the additional benefit of flexible client-side result set navigation.

Example 22.94. Buffered result set for flexible read out

```

<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a'), (2, 'b'), (3, 'c')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$stmt = $mysqli->prepare("SELECT id, label FROM test")) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}

if (!$res = $stmt->get_result()) {
    echo "Getting result set failed: (" . $stmt->errno . ") " . $stmt->error;
}

for ($row_no = ($res->num_rows - 1); $row_no >= 0; $row_no--) {

```

```

    $res->data_seek($row_no);
    var_dump($res->fetch_assoc());
}
$res->close();
?>

```

The above example will output:

```

array(2) {
  ["id"]=>
  int(3)
  ["label"]=>
  string(1) "c"
}
array(2) {
  ["id"]=>
  int(2)
  ["label"]=>
  string(1) "b"
}
array(2) {
  ["id"]=>
  int(1)
  ["label"]=>
  string(1) "a"
}

```

Escaping and SQL injection

Bound variables will be escaped automatically by the server. The server inserts their escaped values at the appropriate places into the statement template before execution. A hint must be provided to the server for the type of bound variable, to create an appropriate conversion. See the [mysqli_stmt_bind_param](#) function for more information.

The automatic escaping of values within the server is sometimes considered a security feature to prevent SQL injection. The same degree of security can be achieved with non-prepared statements, if input values are escaped correctly.

Client-side prepared statement emulation

The API does not include emulation for client-side prepared statement emulation.

Quick prepared - non-prepared statement comparison

The table below compares server-side prepared and non-prepared statements.

Table 22.37. Comparison of prepared and non-prepared statements

	Prepared Statement	Non-prepared statement
Client-server round trips, SELECT, single execution	2	1
Statement string transferred from client to server	1	1
Client-server round trips, SELECT, repeated (n) execution	1 + n	n

	Prepared Statement	Non-prepared statement
Statement string transferred from client to server	1 template, n times bound parameter, if any	n times together with parameter, if any
Input parameter binding API	Yes, automatic input escaping	No, manual input escaping
Output variable binding API	Yes	No
Supports use of <code>mysqli_result</code> API	Yes, use <code>mysqli_stmt_get_result</code>	Yes
Buffered result sets	Yes, use <code>mysqli_stmt_get_result</code> or binding with <code>mysqli_stmt_store_result</code>	Yes, default of <code>mysqli_query</code>
Unbuffered result sets	Yes, use output binding API	Yes, use <code>mysqli_real_query</code> with <code>mysqli_use_result</code>
MySQL Client Server protocol data transfer flavor	Binary protocol	Text protocol
Result set values SQL data types	Preserved when fetching	Converted to string or preserved when fetching
Supports all SQL statements	Recent MySQL versions support most but not all	Yes

See also

```
mysqli::__construct
mysqli::query
mysqli::prepare
mysqli_stmt::prepare
mysqli_stmt::execute
mysqli_stmt::bind_param
mysqli_stmt::bind_result
```

22.9.3.3.5. Stored Procedures

Copyright 1997-2012 the PHP Documentation Group. [3186]

The MySQL database supports stored procedures. A stored procedure is a subroutine stored in the database catalog. Applications can call and execute the stored procedure. The `CALL` SQL statement is used to execute a stored procedure.

Parameter

Stored procedures can have `IN`, `INOUT` and `OUT` parameters, depending on the MySQL version. The `mysqli` interface has no special notion for the different kinds of parameters.

IN parameter

Input parameters are provided with the `CALL` statement. Please, make sure values are escaped correctly.

Example 22.95. Calling a stored procedure

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
```

```

if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: ( " . $mysqli->connect_errno . " ) " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {
    echo "Table creation failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||
    !$mysqli->query("CREATE PROCEDURE p(IN id_val INT) BEGIN INSERT INTO test(id) VALUES(id_val); END;")) {
    echo "Stored procedure creation failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

if (!$mysqli->query("CALL p(1)")) {
    echo "CALL failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

if (!($res = $mysqli->query("SELECT id FROM test"))) {
    echo "SELECT failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

var_dump($res->fetch_assoc());
?>

```

The above example will output:

```

array(1) {
    ["id"]=>
    string(1) "1"
}

```

INOUT/OUT parameter

The values of *INOUT/OUT* parameters are accessed using session variables.

Example 22.96. Using session variables

```

<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: ( " . $mysqli->connect_errno . " ) " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||
    !$mysqli->query('CREATE PROCEDURE p(OUT msg VARCHAR(50)) BEGIN SELECT "Hi!" INTO msg; END;')) {
    echo "Stored procedure creation failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

if (!$mysqli->query("SET @msg = ''") || !$mysqli->query("CALL p(@msg)")) {
    echo "CALL failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

if (!($res = $mysqli->query("SELECT @msg as _p_out"))) {
    echo "Fetch failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

$row = $res->fetch_assoc();
echo $row['_p_out'];
?>

```

The above example will output:

```
Hi!
```

Application and framework developers may be able to provide a more convenient API using a mix of session variables and databased catalog inspection. However, please note the possible performance impact of a custom solution based on catalog inspection.

Handling result sets

Stored procedures can return result sets. Result sets returned from a stored procedure cannot be fetched correctly using `mysqli_query`. The `mysqli_query` function combines statement execution and fetching the first result set into a buffered result set, if any. However, there are additional stored procedure result sets hidden from the user which cause `mysqli_query` to fail returning the user expected result sets.

Result sets returned from a stored procedure are fetched using `mysqli_real_query` or `mysqli_multi_query`. Both functions allow fetching any number of result sets returned by a statement, such as `CALL`. Failing to fetch all result sets returned by a stored procedure causes an error.

Example 22.97. Fetching results from stored procedures

```
<?php
mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||
    !$mysqli->query('CREATE PROCEDURE p() READS SQL DATA BEGIN SELECT id FROM test; SELECT id + 1 FROM test; E')) {
    echo "Stored procedure creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$mysqli->multi_query("CALL p()")) {
    echo "CALL failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

do {
    if ($res = $mysqli->store_result()) {
        printf("---\n");
        var_dump($res->fetch_all());
        $res->free();
    } else {
        if ($mysqli->errno) {
            echo "Store failed: (" . $mysqli->errno . ") " . $mysqli->error;
        }
    }
} while ($mysqli->more_results() && $mysqli->next_result());
?>
```

The above example will output:

```
---
array(3) {
  [0]=>
  array(1) {
    [0]=>
    string(1) "1"
  }
  [1]=>
  array(1) {
    [0]=>
    string(1) "2"
  }
  [2]=>
  array(1) {
    [0]=>
    string(1) "3"
  }
}
---
array(3) {
  [0]=>
  array(1) {
    [0]=>
    string(1) "2"
  }
  [1]=>
  array(1) {
    [0]=>
    string(1) "3"
  }
  [2]=>
  array(1) {
    [0]=>
    string(1) "4"
  }
}
```

Use of prepared statements

No special handling is required when using the prepared statement interface for fetching results from the same stored procedure as above. The prepared statement and non-prepared statement interfaces are similar. Please note, that not every MySQL server version may support preparing the [CALL](#) SQL statement.

Example 22.98. Stored Procedures and Prepared Statements

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||
    !$mysqli->query('CREATE PROCEDURE p() READS SQL DATA BEGIN SELECT id FROM test; SELECT id + 1 FROM tes
```

```
    echo "Stored procedure creation failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

if (!(($stmt = $mysqli->prepare("CALL p()"))) {
    echo "Prepare failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

if (!$stmt->execute()) {
    echo "Execute failed: ( " . $stmt->errno . " ) " . $stmt->error;
}

do {
    if ($res = $stmt->get_result()) {
        printf("---\n");
        var_dump(mysqli_fetch_all($res));
        mysqli_free_result($res);
    } else {
        if ($stmt->errno) {
            echo "Store failed: ( " . $stmt->errno . " ) " . $stmt->error;
        }
    }
} while ($stmt->more_results() && $stmt->next_result());
?>
```

Of course, use of the bind API for fetching is supported as well.

Example 22.99. Stored Procedures and Prepared Statements using bind API

```
<?php
if (!(($stmt = $mysqli->prepare("CALL p()"))) {
    echo "Prepare failed: ( " . $mysqli->errno . " ) " . $mysqli->error;
}

if (!$stmt->execute()) {
    echo "Execute failed: ( " . $stmt->errno . " ) " . $stmt->error;
}

do {

    $id_out = NULL;
    if (!$stmt->bind_result($id_out)) {
        echo "Bind failed: ( " . $stmt->errno . " ) " . $stmt->error;
    }

    while ($stmt->fetch()) {
        echo "id = $id_out\n";
    }
} while ($stmt->more_results() && $stmt->next_result());
?>
```

See also

[mysqli::query](#)
[mysqli::multi_query](#)
[mysqli_result::next-result](#)
[mysqli_result::more-results](#)

22.9.3.3.6. Multiple Statements

Copyright 1997-2012 the PHP Documentation Group. [3186]

MySQL optionally allows having multiple statements in one statement string. Sending multiple statements at once reduces client-server round trips but requires special handling.

Multiple statements or multi queries must be executed with [mysqli_multi_query](#). The individual statements of the statement string are separated by semicolon. Then, all result sets returned by the executed statements must be fetched.

The MySQL server allows having statements that do return result sets and statements that do not return result sets in one multiple statement.

Example 22.100. Multiple Statements

```
<?php
mysqli = new mysqli("example.com", "user", "password", "database");
if (mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . mysqli->connect_errno . ") " . mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {
    echo "Table creation failed: (" . mysqli->errno . ") " . mysqli->error;
}

$sql = "SELECT COUNT(*) AS _num FROM test; ";
$sql.= "INSERT INTO test(id) VALUES (1); ";
$sql.= "SELECT COUNT(*) AS _num FROM test; ";

if (!$mysqli->multi_query($sql)) {
    echo "Multi query failed: (" . mysqli->errno . ") " . mysqli->error;
}

do {
    if ($res = mysqli->store_result()) {
        var_dump($res->fetch_all(MYSQLI_ASSOC));
        $res->free();
    }
} while (mysqli->more_results() && mysqli->next_result());
?>
```

The above example will output:

```
array(1) {
  [0]=>
  array(1) {
    ["_num"]=>
    string(1) "0"
  }
}
array(1) {
  [0]=>
  array(1) {
    ["_num"]=>
    string(1) "1"
  }
}
```

Security considerations

The API functions [mysqli_query](#) and [mysqli_real_query](#) do not set a connection flag necessary for activating multi queries in the server. An extra API call is used for multiple statements to reduce the

likelihood of accidental SQL injection attacks. An attacker may try to add statements such as `; DROP DATABASE mysql` or `; SELECT SLEEP(999)`. If the attacker succeeds in adding SQL to the statement string but `mysqli_multi_query` is not used, the server will not execute the second, injected and malicious SQL statement.

Example 22.101. SQL Injection

```
<?php
mysqli = new mysqli("example.com", "user", "password", "database");
$res    = mysqli->query("SELECT 1; DROP TABLE mysql.user");
if (!$res) {
    echo "Error executing query: (" . mysqli->errno . ") " . mysqli->error;
}
?>
```

The above example will output:

```
Error executing query: (1064) You have an error in your SQL syntax;
check the manual that corresponds to your MySQL server version for the right syntax
to use near 'DROP TABLE mysql.user' at line 1
```

Prepared statements

Use of the multiple statement with prepared statements is not supported.

See also

```
mysqli::query
mysqli::multi_query
mysqli_result::next-result
mysqli_result::more-results
```

22.9.3.3.7. API support for transactions

Copyright 1997-2012 the PHP Documentation Group. [3186]

The MySQL server supports transactions depending on the storage engine used. Since MySQL 5.5, the default storage engine is InnoDB. InnoDB has full ACID transaction support.

Transactions can either be controlled using SQL or API calls. It is recommended to use API calls for enabling and disabling the auto commit mode and for committing and rolling back transactions.

Example 22.102. Setting auto commit mode with SQL and through the API

```
<?php
mysqli = new mysqli("example.com", "user", "password", "database");
if (mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . mysqli->connect_errno . ") " . mysqli->connect_error;
}

/* Recommended: using API to control transactional settings */
mysqli->autocommit(false);

/* Won't be monitored and recognized by the replication and the load balancing plugin */
```

```
if (!$mysqli->query('SET AUTOCOMMIT = 0')) {
    echo "Query failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Optional feature packages, such as the replication and load balancing plugin, can easily monitor API calls. The replication plugin offers transaction aware load balancing, if transactions are controlled with API calls. Transaction aware load balancing is not available if SQL statements are used for setting auto commit mode, committing or rolling back a transaction.

Example 22.103. Commit and rollback

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
$mysqli->autocommit(false);

$mysqli->query("INSERT INTO test(id) VALUES (1)");
$mysqli->rollback();

$mysqli->query("INSERT INTO test(id) VALUES (2)");
$mysqli->commit();
?>
```

Please note, that the MySQL server cannot roll back all statements. Some statements cause an implicit commit.

See also

```
mysqli::autocommit
mysqli_result::commit
mysqli_result::rollback
```

22.9.3.3.8. Metadata

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

A MySQL result set contains metadata. The metadata describes the columns found in the result set. All metadata sent by MySQL is accessible through the [mysqli](#) interface. The extension performs no or negligible changes to the information it receives. Differences between MySQL server versions are not aligned.

Meta data is accessed through the [mysqli_result](#) interface.

Example 22.104. Accessing result set meta data

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

$res = $mysqli->query("SELECT 1 AS _one, 'Hello' AS _two FROM DUAL");
var_dump($res->fetch_fields());
?>
```

The above example will output:

```
array(2) {
  [0]=>
  object(stdClass)#3 (13) {
    ["name"]=>
    string(4) "_one"
    ["orgname"]=>
    string(0) ""
    ["table"]=>
    string(0) ""
    ["orgtable"]=>
    string(0) ""
    ["def"]=>
    string(0) ""
    ["db"]=>
    string(0) ""
    ["catalog"]=>
    string(3) "def"
    ["max_length"]=>
    int(1)
    ["length"]=>
    int(1)
    ["charsetnr"]=>
    int(63)
    ["flags"]=>
    int(32897)
    ["type"]=>
    int(8)
    ["decimals"]=>
    int(0)
  }
  [1]=>
  object(stdClass)#4 (13) {
    ["name"]=>
    string(4) "_two"
    ["orgname"]=>
    string(0) ""
    ["table"]=>
    string(0) ""
    ["orgtable"]=>
    string(0) ""
    ["def"]=>
    string(0) ""
    ["db"]=>
    string(0) ""
    ["catalog"]=>
    string(3) "def"
    ["max_length"]=>
    int(5)
    ["length"]=>
    int(5)
    ["charsetnr"]=>
    int(8)
    ["flags"]=>
    int(1)
    ["type"]=>
    int(253)
    ["decimals"]=>
    int(31)
  }
}
```

Prepared statements

Meta data of result sets created using prepared statements are accessed the same way. A suitable `mysqli_result` handle is returned by `mysqli_stmt_result_metadata`.

Example 22.105. Prepared statements metadata

```
<?php
$stmt = $mysqli->prepare("SELECT 1 AS _one, 'Hello' AS _two FROM DUAL");
$stmt->execute();
$res = $stmt->result_metadata();
var_dump($res->fetch_fields());
?>
```

See also

`mysqli::query`
`mysqli_result::fetch_fields`

22.9.3.4. Installing/Configuring

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

22.9.3.4.1. Requirements

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

In order to have these functions available, you must compile PHP with support for the `mysqli` extension.



Note

The `mysqli` extension is designed to work with MySQL version 4.1.13 or newer, or 5.0.7 or newer. For previous versions, please see the [MySQL](#) extension documentation.

22.9.3.4.2. Installation

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The `mysqli` extension was introduced with PHP version 5.0.0. The MySQL Native Driver was included in PHP version 5.3.0.

22.9.3.4.2.1. Installation on Linux

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The common Unix distributions include binary versions of PHP that can be installed. Although these binary versions are typically built with support for MySQL extensions enabled, the extension libraries themselves may need to be installed using an additional package. Check the package manager that comes with your chosen distribution for availability.

Unless your Unix distribution comes with a binary package of PHP with the `mysqli` extension available, you will need to build PHP from source code. Building PHP from source allows you to specify the MySQL extensions you want to use, as well as your choice of client library for each extension.

The MySQL Native Driver is the recommended option, as it results in improved performance and gives access to features not available when using the MySQL Client Library. Refer to [What is PHP's MySQL Native Driver? \[3270\]](#) for a brief overview of the advantages of MySQL Native Driver.

The `/path/to/mysql_config` represents the location of the `mysql_config` program that comes with MySQL Server.

Table 22.38. mysqli compile time support matrix

PHP Version	Default	Configure Options: <code>mysqlnd</code>	Configure Options: <code>libmysql</code>	Changelog
5.0.x, 5.1.x, 5.2.x	libmysql	Not Available	<code>--with-mysqli=/path/to/mysql_config</code>	
5.3.x	libmysql	<code>--with-mysqli=mysqlnd</code>	<code>--with-mysqli=/path/to/mysql_config</code>	mysqlnd is now supported
5.4.x	mysqlnd	<code>--with-mysqli</code>	<code>--with-mysqli=/path/to/mysql_config</code>	mysqlnd is now the default

Note that it is possible to freely mix MySQL extensions and client libraries. For example, it is possible to enable the MySQL extension to use the MySQL Client Library (libmysql), while configuring the `mysqli` extension to use the MySQL Native Driver. However, all permutations of extension and client library are possible.

The following example builds the MySQL extension to use the MySQL Client Library, and the `mysqli` and PDO MySQL extensions to use the MySQL Native Driver:

```
./configure --with-mysql=/usr/bin/mysql_config \
--with-mysqli=mysqlnd \
--with-pdo-mysql=mysqlnd
[other options]
```

22.9.3.4.2.2. Installation on Windows Systems

Copyright 1997-2012 the PHP Documentation Group. [3186]

On Windows, PHP is most commonly installed using the binary installer.

22.9.3.4.2.2.1. PHP 5.0, 5.1, 5.2

Copyright 1997-2012 the PHP Documentation Group. [3186]

Once PHP has been installed, some configuration is required to enable `mysqli` and specify the client library you want it to use.

The `mysqli` extension is not enabled by default, so the `php_mysqli.dll` DLL must be enabled inside of `php.ini`. In order to do this you need to find the `php.ini` file (typically located in `c:\php`), and make sure you remove the comment (semi-colon) from the start of the line `extension=php_mysqli.dll`, in the section marked `[PHP_MYSQLI]`.

Also, if you want to use the MySQL Client Library with `mysqli`, you need to make sure PHP can access the client library file. The MySQL Client Library is included as a file named `libmysql.dll` in the Windows PHP distribution. This file needs to be available in the Windows system's `PATH` environment variable, so that it can be successfully loaded. See the FAQ titled "[How do I add my PHP directory to the PATH on Windows](#)" for information on how to do this. Copying `libmysql.dll` to the Windows system directory (typically `c:\Windows\system`) also works, as the system directory is by default in the system's `PATH`. However, this practice is strongly discouraged.

As with enabling any PHP extension (such as `php_mysqli.dll`), the PHP directive `extension_dir` should be set to the directory where the PHP extensions are located. See also the [Manual Windows Installation Instructions](#). An example `extension_dir` value for PHP 5 is `c:\php\ext`.

**Note**

If when starting the web server an error similar to the following occurs: `"Unable to load dynamic library './php_mysqli.dll'"`, this is because `php_mysqli.dll` and/or `libmysql.dll` cannot be found by the system.

22.9.3.4.2.2.2. PHP 5.3.0+

Copyright 1997-2012 the PHP Documentation Group. [3186]

On Windows, for PHP versions 5.3 and newer, the `mysqli` extension is enabled and uses the MySQL Native Driver by default. This means you don't need to worry about configuring access to `libmysql.dll`.

22.9.3.4.3. Runtime Configuration

Copyright 1997-2012 the PHP Documentation Group. [3186]

The behaviour of these functions is affected by settings in `php.ini`.

Table 22.39. MySQLi Configuration Options

Name	Default	Changeable	Changelog
<code>mysqli.allow_local_infile</code>	"1"	PHP_INI_SYSTEM	Available since PHP 5.2.4.
<code>mysqli.allow_persistent</code>	"1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
<code>mysqli.max_persistent</code>	"-1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
<code>mysqli.max_links</code>	"-1"	PHP_INI_SYSTEM	Available since PHP 5.0.0.
<code>mysqli.default_port</code>	"3306"	PHP_INI_ALL	Available since PHP 5.0.0.
<code>mysqli.default_socket</code>	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
<code>mysqli.default_host</code>	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
<code>mysqli.default_user</code>	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
<code>mysqli.default_pw</code>	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
<code>mysqli.reconnect</code>	"0"	PHP_INI_SYSTEM	Available since PHP 4.3.5.
<code>mysqli.cache_size</code>	"2000"	PHP_INI_SYSTEM	Available since PHP 5.3.0.

For further details and definitions of the preceding `PHP_INI_*` constants, see the chapter on [configuration changes](#).

Here's a short explanation of the configuration directives.

<i>mysqli.allow_local_infile</i> integer	Allow accessing, from PHP's perspective, local files with LOAD DATA statements
<i>mysqli.allow_persistent</i> integer	Enable the ability to create persistent connections using <i>mysqli_connect</i> .
<i>mysqli.max_persistent</i> integer	Maximum of persistent connections that can be made. Set to 0 for unlimited.
<i>mysqli.max_links</i> integer	The maximum number of MySQL connections per process.
<i>mysqli.default_port</i> integer	The default TCP port number to use when connecting to the database server if no other port is specified. If no default is specified, the port will be obtained from the <i>MYSQL_TCP_PORT</i> environment variable, the <i>mysql-tcp</i> entry in <i>/etc/services</i> or the compile-time <i>MYSQL_PORT</i> constant, in that order. Win32 will only use the <i>MYSQL_PORT</i> constant.
<i>mysqli.default_socket</i> string	The default socket name to use when connecting to a local database server if no other socket name is specified.
<i>mysqli.default_host</i> string	The default server host to use when connecting to the database server if no other host is specified. Doesn't apply in safe mode .
<i>mysqli.default_user</i> string	The default user name to use when connecting to the database server if no other name is specified. Doesn't apply in safe mode .
<i>mysqli.default_pw</i> string	The default password to use when connecting to the database server if no other password is specified. Doesn't apply in safe mode .
<i>mysqli.reconnect</i> integer	Automatically reconnect if the connection was lost.
<i>mysqli.cache_size</i> integer	Available only with <i>mysqlnd</i> .

Users cannot set *MYSQL_OPT_READ_TIMEOUT* through an API call or runtime configuration setting. Note that if it were possible there would be differences between how *libmysql* and streams would interpret the value of *MYSQL_OPT_READ_TIMEOUT*.

22.9.3.4.4. Resource Types

Copyright 1997-2012 the PHP Documentation Group. [3186]

This extension has no resource types defined.

22.9.3.5. The *mysqli* Extension and Persistent Connections

Copyright 1997-2012 the PHP Documentation Group. [3186]

Persistent connection support was introduced in PHP 5.3 for the *mysqli* extension. Support was already present in PDO MySQL and *ext/mysql*. The idea behind persistent connections is that a connection between a client process and a database can be reused by a client process, rather than being created and destroyed multiple times. This reduces the overhead of creating fresh connections every time one is required, as unused connections are cached and ready to be reused.

Unlike the *mysql* extension, *mysqli* does not provide a separate function for opening persistent connections. To open a persistent connection you must prepend *p:* to the hostname when connecting.

The problem with persistent connections is that they can be left in unpredictable states by clients. For example, a table lock might be activated before a client terminates unexpectedly. A new client process reusing this persistent connection will get the connection “as is”. Any cleanup would need to be done by the new client process before it could make good use of the persistent connection, increasing the burden on the programmer.

The persistent connection of the `mysqli` extension however provides built-in cleanup handling code. The cleanup carried out by `mysqli` includes:

- Rollback active transactions
- Close and drop temporary tables
- Unlock tables
- Reset session variables
- Close prepared statements (always happens with PHP)
- Close handler
- Release locks acquired with `GET_LOCK`

This ensures that persistent connections are in a clean state on return from the connection pool, before the client process uses them.

The `mysqli` extension does this cleanup by automatically calling the C-API function `mysql_change_user()`.

The automatic cleanup feature has advantages and disadvantages though. The advantage is that the programmer no longer needs to worry about adding cleanup code, as it is called automatically. However, the disadvantage is that the code could *potentially* be a little slower, as the code to perform the cleanup needs to run each time a connection is returned from the connection pool.

It is possible to switch off the automatic cleanup code, by compiling PHP with `MYSQLI_NO_CHANGE_USER_ON_PCONNECT` defined.

**Note**

The `mysqli` extension supports persistent connections when using either MySQL Native Driver or MySQL Client Library.

22.9.3.6. Predefined Constants

Copyright 1997-2012 the PHP Documentation Group. [3186]

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

`MYSQLI_READ_DEFAULT_GROUP` Read options from the named group from `my.cnf` or the file specified with `MYSQLI_READ_DEFAULT_FILE`

`MYSQLI_READ_DEFAULT_FILE` Read options from the named option file instead of from `my.cnf`

`MYSQLI_OPT_CONNECT_TIMEOUT` Connect timeout in seconds

`MYSQLI_OPT_LOCAL_INFILE` Enables command `LOAD LOCAL INFILE`

`MYSQLI_INIT_COMMAND` Command to execute when connecting to MySQL server. Will automatically be re-executed when reconnecting.

<code>MYSQLI_CLIENT_SSL</code>	Use SSL (encrypted protocol). This option should not be set by application programs; it is set internally in the MySQL client library
<code>MYSQLI_CLIENT_COMPRESS</code>	Use compression protocol
<code>MYSQLI_CLIENT_INTERACTIVE</code>	Allow <code>interactive_timeout</code> seconds (instead of <code>wait_timeout</code> seconds) of inactivity before closing the connection. The client's session <code>wait_timeout</code> variable will be set to the value of the session <code>interactive_timeout</code> variable.
<code>MYSQLI_CLIENT_IGNORE_SPACE</code>	Allow spaces after function names. Makes all functions names reserved words.
<code>MYSQLI_CLIENT_NO_SCHEMA</code>	Don't allow the <code>db_name.tbl_name.col_name</code> syntax.
<code>MYSQLI_CLIENT_MULTI_QUERIES</code>	Allows multiple semicolon-delimited queries in a single <code>mysqli_query</code> call.
<code>MYSQLI_STORE_RESULT</code>	For using buffered resultsets
<code>MYSQLI_USE_RESULT</code>	For using unbuffered resultsets
<code>MYSQLI_ASSOC</code>	Columns are returned into the array having the fieldname as the array index.
<code>MYSQLI_NUM</code>	Columns are returned into the array having an enumerated index.
<code>MYSQLI_BOTH</code>	Columns are returned into the array having both a numerical index and the fieldname as the associative index.
<code>MYSQLI_NOT_NULL_FLAG</code>	Indicates that a field is defined as <code>NOT NULL</code>
<code>MYSQLI_PRI_KEY_FLAG</code>	Field is part of a primary index
<code>MYSQLI_UNIQUE_KEY_FLAG</code>	Field is part of a unique index.
<code>MYSQLI_MULTIPLE_KEY_FLAG</code>	Field is part of an index.
<code>MYSQLI_BLOB_FLAG</code>	Field is defined as <code>BLOB</code>
<code>MYSQLI_UNSIGNED_FLAG</code>	Field is defined as <code>UNSIGNED</code>
<code>MYSQLI_ZEROFILL_FLAG</code>	Field is defined as <code>ZEROFILL</code>
<code>MYSQLI_AUTO_INCREMENT_FLAG</code>	Field is defined as <code>AUTO_INCREMENT</code>
<code>MYSQLI_TIMESTAMP_FLAG</code>	Field is defined as <code>TIMESTAMP</code>
<code>MYSQLI_SET_FLAG</code>	Field is defined as <code>SET</code>
<code>MYSQLI_NUM_FLAG</code>	Field is defined as <code>NUMERIC</code>
<code>MYSQLI_PART_KEY_FLAG</code>	Field is part of an multi-index
<code>MYSQLI_GROUP_FLAG</code>	Field is part of <code>GROUP BY</code>
<code>MYSQLI_TYPE_DECIMAL</code>	Field is defined as <code>DECIMAL</code>
<code>MYSQLI_TYPE_NEWDECIMAL</code>	Precision math <code>DECIMAL</code> or <code>NUMERIC</code> field (MySQL 5.0.3 and up)
<code>MYSQLI_TYPE_BIT</code>	Field is defined as <code>BIT</code> (MySQL 5.0.3 and up)

<code>MYSQLI_TYPE_TINY</code>	Field is defined as <code>TINYINT</code>
<code>MYSQLI_TYPE_SHORT</code>	Field is defined as <code>SMALLINT</code>
<code>MYSQLI_TYPE_LONG</code>	Field is defined as <code>INT</code>
<code>MYSQLI_TYPE_FLOAT</code>	Field is defined as <code>FLOAT</code>
<code>MYSQLI_TYPE_DOUBLE</code>	Field is defined as <code>DOUBLE</code>
<code>MYSQLI_TYPE_NULL</code>	Field is defined as <code>DEFAULT NULL</code>
<code>MYSQLI_TYPE_TIMESTAMP</code>	Field is defined as <code>TIMESTAMP</code>
<code>MYSQLI_TYPE_LONGLONG</code>	Field is defined as <code>BIGINT</code>
<code>MYSQLI_TYPE_INT24</code>	Field is defined as <code>MEDIUMINT</code>
<code>MYSQLI_TYPE_DATE</code>	Field is defined as <code>DATE</code>
<code>MYSQLI_TYPE_TIME</code>	Field is defined as <code>TIME</code>
<code>MYSQLI_TYPE_DATETIME</code>	Field is defined as <code>DATETIME</code>
<code>MYSQLI_TYPE_YEAR</code>	Field is defined as <code>YEAR</code>
<code>MYSQLI_TYPE_NEWDATE</code>	Field is defined as <code>DATE</code>
<code>MYSQLI_TYPE_INTERVAL</code>	Field is defined as <code>INTERVAL</code>
<code>MYSQLI_TYPE_ENUM</code>	Field is defined as <code>ENUM</code>
<code>MYSQLI_TYPE_SET</code>	Field is defined as <code>SET</code>
<code>MYSQLI_TYPE_TINY_BLOB</code>	Field is defined as <code>TINYBLOB</code>
<code>MYSQLI_TYPE_MEDIUM_BLOB</code>	Field is defined as <code>MEDIUMBLOB</code>
<code>MYSQLI_TYPE_LONG_BLOB</code>	Field is defined as <code>LOB</code>
<code>MYSQLI_TYPE_BLOB</code>	Field is defined as <code>BLOB</code>
<code>MYSQLI_TYPE_VAR_STRING</code>	Field is defined as <code>VARCHAR</code>
<code>MYSQLI_TYPE_STRING</code>	Field is defined as <code>CHAR</code> or <code>BINARY</code>
<code>MYSQLI_TYPE_CHAR</code>	Field is defined as <code>TINYINT</code> . For <code>CHAR</code> , see <code>MYSQLI_TYPE_STRING</code>
<code>MYSQLI_TYPE_GEOMETRY</code>	Field is defined as <code>GEOMETRY</code>
<code>MYSQLI_NEED_DATA</code>	More data available for bind variable
<code>MYSQLI_NO_DATA</code>	No more data available for bind variable
<code>MYSQLI_DATA_TRUNCATED</code>	Data truncation occurred. Available since PHP 5.1.0 and MySQL 5.0.5.
<code>MYSQLI_ENUM_FLAG</code>	Field is defined as <code>ENUM</code> . Available since PHP 5.3.0.
<code>MYSQLI_CURSOR_TYPE_FOR_UPDATE</code>	
<code>MYSQLI_CURSOR_TYPE_NO_CURSOR</code>	

<code>MYSQLI_CURSOR_TYPE_READ_ONLY</code>	
<code>MYSQLI_CURSOR_TYPE_SCROLLABLE</code>	
<code>MYSQLI_STMT_ATTR_CURSOR_TYPE</code>	
<code>MYSQLI_STMT_ATTR_PREFETCH_ROWS</code>	
<code>MYSQLI_STMT_ATTR_UPDATE_MAX_LENGTH</code>	
<code>MYSQLI_SET_CHARSET_NAME</code>	
<code>MYSQLI_REPORT_INDEX</code>	Report if no index or bad index was used in a query.
<code>MYSQLI_REPORT_ERROR</code>	Report errors from <code>mysqli</code> function calls.
<code>MYSQLI_REPORT_STRICT</code>	Throw a <code>mysqli_sql_exception</code> for errors instead of warnings.
<code>MYSQLI_REPORT_ALL</code>	Set all options on (report all).
<code>MYSQLI_REPORT_OFF</code>	Turns reporting off.
<code>MYSQLI_DEBUG_TRACE_ENABLED</code>	Is set to 1 if <code>mysqli_debug</code> functionality is enabled.
<code>MYSQLI_SERVER_QUERY_NO_GOOD_INDEX_USED</code>	
<code>MYSQLI_SERVER_QUERY_NO_INDEX_USED</code>	
<code>MYSQLI_REFRESH_GRANT</code>	Refreshes the grant tables.
<code>MYSQLI_REFRESH_LOG</code>	Flushes the logs, like executing the <code>FLUSH LOGS</code> SQL statement.
<code>MYSQLI_REFRESH_TABLES</code>	Flushes the table cache, like executing the <code>FLUSH TABLES</code> SQL statement.
<code>MYSQLI_REFRESH_HOSTS</code>	Flushes the host cache, like executing the <code>FLUSH HOSTS</code> SQL statement.
<code>MYSQLI_REFRESH_STATUS</code>	Reset the status variables, like executing the <code>FLUSH STATUS</code> SQL statement.
<code>MYSQLI_REFRESH_THREADS</code>	Flushes the thread cache.
<code>MYSQLI_REFRESH_SLAVE</code>	On a slave replication server: resets the master server information, and restarts the slave. Like executing the <code>RESET SLAVE</code> SQL statement.
<code>MYSQLI_REFRESH_MASTER</code>	On a master replication server: removes the binary log files listed in the binary log index, and truncates the index file. Like executing the <code>RESET MASTER</code> SQL statement.

22.9.3.7. Notes

Copyright 1997-2012 the PHP Documentation Group. [3186]

Some implementation notes:

1. Support was added for `MYSQL_TYPE_GEOMETRY` to the `MySQLi` extension in PHP 5.3.
2. Note there are different internal implementations within `libmysql` and `mysqlnd` for handling columns of type `MYSQL_TYPE_GEOMETRY`. Generally speaking, `mysqlnd` will allocate significantly less memory.

For example, if there is a `POINT` column in a result set, `libmysql` may pre-allocate up to 4GB of RAM although less than 50 bytes are needed for holding a `POINT` column in memory. Memory allocation is much lower, less than 50 bytes, if using `mysqlnd`.

22.9.3.8. The MySQLi Extension Function Summary

Copyright 1997-2012 the PHP Documentation Group. [3186]

Table 22.40. Summary of `mysqli` methods

mysqli Class			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Properties</i>			
<code>\$mysqli::affected_rows</code>	<code>mysqli_affected_rows</code>	N/A	Gets the number of affected rows in a previous MySQL operation
<code>\$mysqli::client_info</code>	<code>mysqli_get_client_info</code>	N/A	Returns the MySQL client version as a string
<code>\$mysqli::client_version</code>	<code>mysqli_get_client_version</code>	N/A	Returns MySQL client version info as an integer
<code>\$mysqli::connect_errno</code>	<code>mysqli_connect_errno</code>	N/A	Returns the error code from last connect call
<code>\$mysqli::connect_error</code>	<code>mysqli_connect_error</code>	N/A	Returns a string description of the last connect error
<code>\$mysqli::errno</code>	<code>mysqli_errno</code>	N/A	Returns the error code for the most recent function call
<code>\$mysqli::error</code>	<code>mysqli_error</code>	N/A	Returns a string description of the last error
<code>\$mysqli::field_count</code>	<code>mysqli_field_count</code>	N/A	Returns the number of columns for the most recent query
<code>\$mysqli::host_info</code>	<code>mysqli_get_host_info</code>	N/A	Returns a string representing the type of connection used
<code>\$mysqli::protocol_version</code>	<code>mysqli_get_proto_info</code>	N/A	Returns the version of the MySQL protocol used
<code>\$mysqli::server_info</code>	<code>mysqli_get_server_info</code>	N/A	Returns the version of the MySQL server
<code>\$mysqli::server_version</code>	<code>mysqli_get_server_version</code>	N/A	Returns the version of the MySQL server as an integer
<code>\$mysqli::info</code>	<code>mysqli_info</code>	N/A	Retrieves information about the most recently executed query

mysqli Class			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
\$mysqli::insert_id	<code>mysqli_insert_id</code>	N/A	Returns the auto generated id used in the last query
\$mysqli::sqlstate	<code>mysqli_sqlstate</code>	N/A	Returns the SQLSTATE error from previous MySQL operation
\$mysqli::warning_count	<code>mysqli_warning_count</code>	N/A	Returns the number of warnings from the last query for the given link
<i>Methods</i>			
<code>mysqli::autocommit</code>	<code>mysqli_autocommit</code>	N/A	Turns on or off auto-committing database modifications
<code>mysqli::change_user</code>	<code>mysqli_change_user</code>	N/A	Changes the user of the specified database connection
<code>mysqli::character_set_name</code> mysqli::client_encoding	<code>mysqli_character_set_name</code>	<code>mysqli_client_encoding</code>	Returns the default character set for the database connection
<code>mysqli::close</code>	<code>mysqli_close</code>	N/A	Closes a previously opened database connection
<code>mysqli::commit</code>	<code>mysqli_commit</code>	N/A	Commits the current transaction
<code>mysqli::__construct</code>	<code>mysqli_connect</code>	N/A	Open a new connection to the MySQL server [Note: static (i.e. class) method]
<code>mysqli::debug</code>	<code>mysqli_debug</code>	N/A	Performs debugging operations
<code>mysqli::dump_debug_info</code>	<code>mysqli_dump_debug_info</code>	N/A	Dump debugging information into the log
<code>mysqli::get_charset</code>	<code>mysqli_get_charset</code>	N/A	Returns a character set object
<code>mysqli::get_connection_stats</code>	<code>mysqli_get_connection_stats</code>	N/A	Returns client connection statistics. Available only with mysqli .
<code>mysqli::get_client_info</code>	<code>mysqli_get_client_info</code>	N/A	Returns the MySQL client version as a string
<code>mysqli::get_client_stats</code>	<code>mysqli_get_client_stats</code>	N/A	Returns client per-process statistics. Available only with mysqli .

mysqli Class			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>mysqli::get_cache_stats</code>	<code>mysqli_get_cache_stats</code>	N/A	Returns client Zval cache statistics. Available only with mysqlind .
<code>mysqli::get_server_info</code>	<code>mysqli_get_server_info</code>	N/A	NOT DOCUMENTED
<code>mysqli::get_warnings</code>	<code>mysqli_get_warnings</code>	N/A	NOT DOCUMENTED
<code>mysqli::init</code>	<code>mysqli_init</code>	N/A	Initializes MySQLi and returns a resource for use with <code>mysqli_real_connect</code> . [Not called on an object, as it returns a \$mysqli object.]
<code>mysqli::kill</code>	<code>mysqli_kill</code>	N/A	Asks the server to kill a MySQL thread
<code>mysqli::more_results</code>	<code>mysqli_more_results</code>	N/A	Check if there are any more query results from a multi query
<code>mysqli::multi_query</code>	<code>mysqli_multi_query</code>	N/A	Performs a query on the database
<code>mysqli::next_result</code>	<code>mysqli_next_result</code>	N/A	Prepare next result from multi_query
<code>mysqli::options</code>	<code>mysqli_options</code>	<code>mysqli_set_opt</code>	Set options
<code>mysqli::ping</code>	<code>mysqli_ping</code>	N/A	Pings a server connection, or tries to reconnect if the connection has gone down
<code>mysqli::prepare</code>	<code>mysqli_prepare</code>	N/A	Prepare an SQL statement for execution
<code>mysqli::query</code>	<code>mysqli_query</code>	N/A	Performs a query on the database
<code>mysqli::real_connect</code>	<code>mysqli_real_connect</code>	N/A	Opens a connection to a mysql server
<code>mysqli::real_escape_string</code> <code>mysqli::escape_string</code>	<code>mysqli_real_escape_string</code> <code>mysqli_escape_string</code>		Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection
<code>mysqli::real_query</code>	<code>mysqli_real_query</code>	N/A	Execute an SQL query
<code>mysqli::refresh</code>	<code>mysqli_refresh</code>	N/A	Flushes tables or caches, or resets the replication server information

mysqli Class			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>mysqli::rollback</code>	<code>mysqli_rollback</code>	N/A	Rolls back current transaction
<code>mysqli::select_db</code>	<code>mysqli_select_db</code>	N/A	Selects the default database for database queries
<code>mysqli::set_charset</code>	<code>mysqli_set_charset</code>	N/A	Sets the default client character set
<code>mysqli::set_local_infile_handler_default</code>	<code>mysqli_set_local_infile_handler_default</code>	N/A	Unsets user defined handler for load local infile command
<code>mysqli::set_local_infile_handler</code>	<code>mysqli_set_local_infile_handler</code>	N/A	Set callback function for LOAD DATA LOCAL INFILE command
<code>mysqli::ssl_set</code>	<code>mysqli_ssl_set</code>	N/A	Used for establishing secure connections using SSL
<code>mysqli::stat</code>	<code>mysqli_stat</code>	N/A	Gets the current system status
<code>mysqli::stmt_init</code>	<code>mysqli_stmt_init</code>	N/A	Initializes a statement and returns an object for use with <code>mysqli_stmt_prepare</code>
<code>mysqli::store_result</code>	<code>mysqli_store_result</code>	N/A	Transfers a result set from the last query
<code>mysqli::thread_id</code>	<code>mysqli_thread_id</code>	N/A	Returns the thread ID for the current connection
<code>mysqli::thread_safe</code>	<code>mysqli_thread_safe</code>	N/A	Returns whether thread safety is given or not
<code>mysqli::use_result</code>	<code>mysqli_use_result</code>	N/A	Initiate a result set retrieval

Table 22.41. Summary of `mysqli_stmt` methods

MySQL_STMT			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Properties</i>			
<code>\$mysqli_stmt::affected_rows</code>	<code>mysqli_stmt_affected_rows</code>	N/A	Returns the total number of rows changed, deleted, or inserted by the last executed statement
<code>\$mysqli_stmt::errno</code>	<code>mysqli_stmt_errno</code>	N/A	Returns the error code for the most recent statement call

MySQL_STMT			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>\$mysqli_stmt::error</code>	<code>mysqli_stmt_error</code>	N/A	Returns a string description for last statement error
<code>\$mysqli_stmt::field_count</code>	<code>mysqli_stmt_field_count</code>	N/A	Returns the number of field in the given statement - not documented
<code>\$mysqli_stmt::insert_id</code>	<code>mysqli_stmt_insert_id</code>	N/A	Get the ID generated from the previous INSERT operation
<code>\$mysqli_stmt::num_rows</code>	<code>mysqli_stmt_num_rows</code>	N/A	Return the number of rows in statements result set
<code>\$mysqli_stmt::param_count</code>	<code>mysqli_stmt_param_count</code>	<code>mysqli_param_count</code>	Returns the number of parameter for the given statement
<code>\$mysqli_stmt::sqlstate</code>	<code>mysqli_stmt_sqlstate</code>	N/A	Returns SQLSTATE error from previous statement operation
<i>Methods</i>			
<code>mysqli_stmt::attr_get</code>	<code>mysqli_stmt_attr_get</code>	N/A	Used to get the current value of a statement attribute
<code>mysqli_stmt::attr_set</code>	<code>mysqli_stmt_attr_set</code>	N/A	Used to modify the behavior of a prepared statement
<code>mysqli_stmt::bind_param</code>	<code>mysqli_stmt_bind_param</code>	<code>mysqli_bind_param</code>	Binds variables to a prepared statement as parameters
<code>mysqli_stmt::bind_result</code>	<code>mysqli_stmt_bind_result</code>	<code>mysqli_bind_result</code>	Binds variables to a prepared statement for result storage
<code>mysqli_stmt::close</code>	<code>mysqli_stmt_close</code>	N/A	Closes a prepared statement
<code>mysqli_stmt::data_seek</code>	<code>mysqli_stmt_data_seek</code>	N/A	Seeks to an arbitrary row in statement result set
<code>mysqli_stmt::execute</code>	<code>mysqli_stmt_execute</code>	<code>mysqli_execute</code>	Executes a prepared Query
<code>mysqli_stmt::fetch</code>	<code>mysqli_stmt_fetch</code>	<code>mysqli_fetch</code>	Fetch results from a prepared statement into the bound variables
<code>mysqli_stmt::free_result</code>	<code>mysqli_stmt_free_result</code>	N/A	Frees stored result memory for the given statement handle

MySQL_STMT			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>mysqli_stmt::get_result()</code>	<code>mysqli_stmt_get_result()</code>	N/A	Gets a result set from a prepared statement. Available only with mysqli .
<code>mysqli_stmt::get_warnings()</code>	<code>mysqli_stmt_get_warnings()</code>	N/A	NOT DOCUMENTED
<code>\$mysqli_stmt::more_results()</code>	<code>mysqli_stmt_more_results()</code>	N/A	NOT DOCUMENTED Available only with mysqli .
<code>\$mysqli_stmt::next_result()</code>	<code>mysqli_stmt_next_result()</code>	N/A	NOT DOCUMENTED Available only with mysqli .
<code>mysqli_stmt::num_rows</code>	<code>mysqli_stmt_num_rows</code>	N/A	See also property \$mysqli_stmt::num_rows
<code>mysqli_stmt::prepare()</code>	<code>mysqli_stmt_prepare()</code>	N/A	Prepare an SQL statement for execution
<code>mysqli_stmt::reset</code>	<code>mysqli_stmt_reset</code>	N/A	Resets a prepared statement
<code>mysqli_stmt::result_metadata</code>	<code>mysqli_stmt_result_metadata</code>	<code>mysqli_get_metadata</code>	Returns result set metadata from a prepared statement
<code>mysqli_stmt::send_long_data()</code>	<code>mysqli_stmt_send_long_data()</code>	<code>mysqli_send_long_data()</code>	Send data in blocks
<code>mysqli_stmt::store_result()</code>	<code>mysqli_stmt_store_result()</code>	N/A	Transfers a result set from a prepared statement

Table 22.42. Summary of `mysqli_result` methods

mysqli_result			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Properties</i>			
<code>\$mysqli_result::current_field</code>	<code>mysqli_field_tell</code>	N/A	Get current field offset of a result pointer
<code>\$mysqli_result::field_count</code>	<code>mysqli_num_fields</code>	N/A	Get the number of fields in a result
<code>\$mysqli_result::lengths</code>	<code>mysqli_fetch_lengths</code>	N/A	Returns the lengths of the columns of the current row in the result set
<code>\$mysqli_result::num_rows</code>	<code>mysqli_num_rows</code>	N/A	Gets the number of rows in a result
<i>Methods</i>			
<code>mysqli_result::data_seek()</code>	<code>mysqli_data_seek</code>	N/A	Adjusts the result pointer to an arbitrary row in the result

mysqli_result			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_all</code>	N/A	Fetches all result rows and returns the result set as an associative array, a numeric array, or both. Available only with mysqlind .
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_array</code>	N/A	Fetch a result row as an associative, a numeric array, or both
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_assoc</code>	N/A	Fetch a result row as an associative array
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_field_direct</code>	N/A	Fetch meta-data for a single field
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_field</code>	N/A	Returns the next field in the result set
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_fields</code>	N/A	Returns an array of objects representing the fields in a result set
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_object</code>	N/A	Returns the current row of a result set as an object
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_row</code>	N/A	Get a result row as an enumerated array
<code>mysqli_result::field</code>	<code>mysqli_field_seek</code>	N/A	Set result pointer to a specified field offset
<code>mysqli_result::free</code> , <code>mysqli_result::close</code> , <code>mysqli_result::free_result</code>	<code>mysqli_free_result</code>	N/A	Frees the memory associated with a result

Table 22.43. Summary of `mysqli_driver` methods

MySQL_Driver			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Properties</i>			
N/A			
<i>Methods</i>			
<code>mysqli_driver::embed</code>	<code>mysqli_embedded_server_end</code>	N/A	NOT DOCUMENTED
<code>mysqli_driver::embed</code>	<code>mysqli_embedded_server_start</code>	N/A	NOT DOCUMENTED



Note

Alias functions are provided for backward compatibility purposes only. Do not use them in new projects.

22.9.3.9. The `mysqli` class ([mysqli](#))

Represents a connection between PHP and a MySQL database.

```
mysqli {
mysqli

    Properties

    int
        mysqli->affected_rows ;

    string
        mysqli->client_info ;

    int
        mysqli->client_version ;

    string
        mysqli->connect_errno ;

    string
        mysqli->connect_error ;

    int
        mysqli->errno ;

    array
        mysqli->error_list ;

    string
        mysqli->error ;

    int
        mysqli->field_count ;

    int
        mysqli->client_version ;

    string
        mysqli->host_info ;

    string
        mysqli->protocol_version ;

    string
        mysqli->server_info ;

    int
        mysqli->server_version ;

    string
        mysqli->info ;

    mixed
        mysqli->insert_id ;

    string
        mysqli->sqlstate ;

    int
        mysqli->thread_id ;

    int
        mysqli->warning_count ;
```

Methods

```
mysqli::__construct(
    string host
        = =ini_get("mysqli.default_host"),
    string username
        = =ini_get("mysqli.default_user"),
    string passwd
        = =ini_get("mysqli.default_pw"),
    string dbname
        = = "",
    int port
        = =ini_get("mysqli.default_port"),
    string socket
        = =ini_get("mysqli.default_socket"));

bool mysqli::autocommit(
    bool mode);

bool mysqli::change_user(
    string user,
    string password,
    string database);

string mysqli::character_set_name();

bool mysqli::close();

bool mysqli::commit();

bool mysqli::debug(
    string message);

bool mysqli::dump_debug_info();

object mysqli::get_charset();

string mysqli::get_client_info();

bool mysqli::get_connection_stats();

mysqli_warning mysqli::get_warnings();

mysqli mysqli::init();

bool mysqli::kill(
    int processid);

bool mysqli::more_results();

bool mysqli::multi_query(
    string query);

bool mysqli::next_result();

bool mysqli::options(
    int option,
    mixed value);

bool mysqli::ping();

public int mysqli::poll(
    array read,
    array error,
    array reject,
```

```
int sec,
int usec);

mysqli_stmt mysqli::prepare(
    string query);

mixed mysqli::query(
    string query,
    int resultmode
    = =MYSQLI_STORE_RESULT);

bool mysqli::real_connect(
    string host,
    string username,
    string passwd,
    string dbname,
    int port,
    string socket,
    int flags);

string mysqli::escape_string(
    string escapestr);

bool mysqli::real_query(
    string query);

public mysqli_result mysqli::reap_async_query();

public bool mysqli::refresh(
    int options);

bool mysqli::rollback();

int mysqli::rpl_query_type(
    string query);

bool mysqli::select_db(
    string dbname);

bool mysqli::send_query(
    string query);

bool mysqli::set_charset(
    string charset);

bool mysqli::set_local_infile_handler(
    mysqli link,
    callable read_func);

bool mysqli::ssl_set(
    string key,
    string cert,
    string ca,
    string capath,
    string cipher);

string mysqli::stat();

mysqli_stmt mysqli::stmt_init();

mysqli_result mysqli::store_result();

mysqli_result mysqli::use_result();
}
```

22.9.3.9.1. *mysqli::\$affected_rows*, *mysqli_affected_rows*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysqli::\$affected_rows*

mysqli_affected_rows

Gets the number of affected rows in a previous MySQL operation

Description

Object oriented style

```
int
mysqli->affected_rows ;
```

Procedural style

```
int mysqli_affected_rows(
mysqli link);
```

Returns the number of rows affected by the last *INSERT*, *UPDATE*, *REPLACE* or *DELETE* query.For *SELECT* statements *mysqli_affected_rows* works like *mysqli_num_rows*.**Parameters**

link Procedural style only: A link identifier returned by *mysqli_connect* or *mysqli_init*

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an *UPDATE* statement, no rows matched the *WHERE* clause in the query or that no query has yet been executed. -1 indicates that the query returned an error.

**Note**

If the number of affected rows is greater than maximal int value, the number of affected rows will be returned as a string.

Examples**Example 22.106. *\$mysqli->affected_rows* example**

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Insert rows */
```

```
$mysqli->query("CREATE TABLE Language SELECT * from CountryLanguage");
printf("Affected rows (INSERT): %d\n", $mysqli->affected_rows);

$mysqli->query("ALTER TABLE Language ADD Status int default 0");

/* update rows */
$mysqli->query("UPDATE Language SET Status=1 WHERE Percentage > 50");
printf("Affected rows (UPDATE): %d\n", $mysqli->affected_rows);

/* delete rows */
$mysqli->query("DELETE FROM Language WHERE Percentage < 50");
printf("Affected rows (DELETE): %d\n", $mysqli->affected_rows);

/* select all rows */
$result = $mysqli->query("SELECT CountryCode FROM Language");
printf("Affected rows (SELECT): %d\n", $mysqli->affected_rows);

$result->close();

/* Delete table Language */
$mysqli->query("DROP TABLE Language");

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

if (!$link) {
    printf("Can't connect to localhost. Error: %s\n", mysqli_connect_error());
    exit();
}

/* Insert rows */
mysqli_query($link, "CREATE TABLE Language SELECT * from CountryLanguage");
printf("Affected rows (INSERT): %d\n", mysqli_affected_rows($link));

mysqli_query($link, "ALTER TABLE Language ADD Status int default 0");

/* update rows */
mysqli_query($link, "UPDATE Language SET Status=1 WHERE Percentage > 50");
printf("Affected rows (UPDATE): %d\n", mysqli_affected_rows($link));

/* delete rows */
mysqli_query($link, "DELETE FROM Language WHERE Percentage < 50");
printf("Affected rows (DELETE): %d\n", mysqli_affected_rows($link));

/* select all rows */
$result = mysqli_query($link, "SELECT CountryCode FROM Language");
printf("Affected rows (SELECT): %d\n", mysqli_affected_rows($link));

mysqli_free_result($result);

/* Delete table Language */
mysqli_query($link, "DROP TABLE Language");

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Affected rows (INSERT): 984
Affected rows (UPDATE): 168
Affected rows (DELETE): 815
Affected rows (SELECT): 169
```

See Also

[mysqli_num_rows](#)
[mysqli_info](#)

22.9.3.9.2. [mysqli::autocommit](#), [mysqli_autocommit](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::autocommit](#)

[mysqli_autocommit](#)

Turns on or off auto-committing database modifications

Description

Object oriented style

```
bool mysqli::autocommit(
    bool mode);
```

Procedural style

```
bool mysqli_autocommit(
    mysqli link,
    bool mode);
```

Turns on or off auto-commit mode on queries for the database connection.

To determine the current state of autocommit use the SQL command [SELECT @@autocommit](#).

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

mode Whether to turn on auto-commit or not.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Notes



Note

This function doesn't work with non transactional table types (like MyISAM or ISAM).

Examples

Example 22.107. `mysqli::autocommit` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* turn autocommit on */
$mysqli->autocommit(TRUE);

if ($result = $mysqli->query("SELECT @@autocommit")) {
    $row = $result->fetch_row();
    printf("Autocommit is %s\n", $row[0]);
    $result->free();
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

if (!$link) {
    printf("Can't connect to localhost. Error: %s\n", mysqli_connect_error());
    exit();
}

/* turn autocommit on */
mysqli_autocommit($link, TRUE);

if ($result = mysqli_query($link, "SELECT @@autocommit")) {
    $row = mysqli_fetch_row($result);
    printf("Autocommit is %s\n", $row[0]);
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Autocommit is 1
```

See Also

[mysqli_commit](#)

`mysqli_rollback`

22.9.3.9.3. `mysqli::change_user`, `mysqli_change_user`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::change_user`

`mysqli_change_user`

Changes the user of the specified database connection

Description

Object oriented style

```
bool mysqli::change_user(  
    string user,  
    string password,  
    string database);
```

Procedural style

```
bool mysqli_change_user(  
    mysqli link,  
    string user,  
    string password,  
    string database);
```

Changes the user of the specified database connection and sets the current database.

In order to successfully change users a valid *username* and *password* parameters must be provided and that user must have sufficient permissions to access the desired database. If for any reason authorization fails, the current user authentication will remain.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by <code>mysqli_connect</code> or <code>mysqli_init</code>
<i>user</i>	The MySQL user name.
<i>password</i>	The MySQL password.
<i>database</i>	The database to change to.

If desired, the `NULL` value may be passed resulting in only changing the user and not selecting a database. To select a database in this case use the `mysqli_select_db` function.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Notes



Note

Using this command will always cause the current database connection to behave as if was a completely new database connection, regardless of if the operation was

completed successfully. This reset includes performing a rollback on any active transactions, closing all temporary tables, and unlocking all locked tables.

Examples

Example 22.108. `mysqli::change_user` example

Object oriented style

```
<?php

/* connect database test */
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Set Variable a */
$mysqli->query("SET @a:=1");

/* reset all and select a new database */
$mysqli->change_user("my_user", "my_password", "world");

if ($result = $mysqli->query("SELECT DATABASE()")) {
    $row = $result->fetch_row();
    printf("Default database: %s\n", $row[0]);
    $result->close();
}

if ($result = $mysqli->query("SELECT @a")) {
    $row = $result->fetch_row();
    if ($row[0] === NULL) {
        printf("Value of variable a is NULL\n");
    }
    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php

/* connect database test */
$link = mysqli_connect("localhost", "my_user", "my_password", "test");

/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Set Variable a */
mysqli_query($link, "SET @a:=1");

/* reset all and select a new database */
mysqli_change_user($link, "my_user", "my_password", "world");
```

```
if ($result = mysqli_query($link, "SELECT DATABASE()")) {
    $row = mysqli_fetch_row($result);
    printf("Default database: %s\n", $row[0]);
    mysqli_free_result($result);
}

if ($result = mysqli_query($link, "SELECT @a")) {
    $row = mysqli_fetch_row($result);
    if ($row[0] === NULL) {
        printf("Value of variable a is NULL\n");
    }
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Default database: world
Value of variable a is NULL
```

See Also

[mysqli_connect](#)
[mysqli_select_db](#)

22.9.3.9.4. [mysqli::character_set_name](#), [mysqli_character_set_name](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::character_set_name](#)
[mysqli_character_set_name](#)

Returns the default character set for the database connection

Description

Object oriented style

```
string mysqli::character_set_name();
```

Procedural style

```
string mysqli_character_set_name(
    mysqli link);
```

Returns the current character set for the database connection.

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

The default character set for the current connection

Examples

Example 22.109. [mysqli::character_set_name](#) example

Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Print current character set */
$charset = $mysqli->character_set_name();
printf ("Current character set is %s\n", $charset);

$mysqli->close();
?>
```

Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Print current character set */
$charset = mysqli_character_set_name($link);
printf ("Current character set is %s\n", $charset);

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Current character set is latin1_swedish_ci
```

See Also

[mysqli_set_charset](#)

```
mysqli_client_encoding  
mysqli_real_escape_string
```

22.9.3.9.5. [mysqli::\\$client_info](#), [mysqli_get_client_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$client_info](#)

```
mysqli_get_client_info
```

Get MySQL client info

Description

Object oriented style

```
string  
mysqli->client_info ;
```

Procedural style

```
string mysqli_get_client_info(  
    mysqli link);
```

Returns a string that represents the MySQL client library version.

Return Values

A string that represents the MySQL client library version

Examples

Example 22.110. [mysqli_get_client_info](#)

```
<?php  
  
/* We don't need a connection to determine  
   the version of mysql client library */  
  
printf("Client library version: %s\n", mysqli_get_client_info());  
?>
```

See Also

```
mysqli_get_client_version  
mysqli_get_server_info  
mysqli_get_server_version
```

22.9.3.9.6. [mysqli::\\$client_version](#), [mysqli_get_client_version](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$client_version](#)

```
mysqli_get_client_version
```

Returns the MySQL client version as a string

Description

Object oriented style

```
int  
mysqli->client_version ;
```

Procedural style

```
int mysqli_get_client_version(  
mysqli link);
```

Returns client version number as an integer.

Return Values

A number that represents the MySQL client library version in format: `main_version*10000 + minor_version *100 + sub_version`. For example, 4.1.0 is returned as 40100.

This is useful to quickly determine the version of the client library to know if some capability exists.

Examples

Example 22.111. `mysqli_get_client_version`

```
<?php  
  
/* We don't need a connection to determine  
   the version of mysql client library */  
  
printf("Client library version: %d\n", mysqli_get_client_version());  
?>
```

See Also

[mysqli_get_client_info](#)
[mysqli_get_server_info](#)
[mysqli_get_server_version](#)

22.9.3.9.7. `mysqli::close`, `mysqli_close`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::close](#)

[mysqli_close](#)

Closes a previously opened database connection

Description

Object oriented style

```
bool mysqli::close();
```

Procedural style

```
bool mysqli_close(  
    mysqli link);
```

Closes a previously opened database connection.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

See `mysqli_connect`.

See Also

```
mysqli::__construct  
mysqli_init  
mysqli_real_connect
```

22.9.3.9.8. `mysqli::commit`, `mysqli_commit`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::commit`

```
mysqli_commit
```

Commits the current transaction

Description

Object oriented style

```
bool mysqli::commit();
```

Procedural style

```
bool mysqli_commit(  
    mysqli link);
```

Commits the current transaction for the database connection.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.112. `mysqli::commit` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$mysqli->query("CREATE TABLE Language LIKE CountryLanguage");

/* set autocommit to off */
$mysqli->autocommit(FALSE);

/* Insert some values */
$mysqli->query("INSERT INTO Language VALUES ('DEU', 'Bavarian', 'F', 11.2)");
$mysqli->query("INSERT INTO Language VALUES ('DEU', 'Swabian', 'F', 9.4)");

/* commit transaction */
$mysqli->commit();

/* drop table */
$mysqli->query("DROP TABLE Language");

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");

/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* set autocommit to off */
mysqli_autocommit($link, FALSE);

mysqli_query($link, "CREATE TABLE Language LIKE CountryLanguage");

/* Insert some values */
mysqli_query($link, "INSERT INTO Language VALUES ('DEU', 'Bavarian', 'F', 11.2)");
mysqli_query($link, "INSERT INTO Language VALUES ('DEU', 'Swabian', 'F', 9.4)");

/* commit transaction */
mysqli_commit($link);

/* close connection */
mysqli_close($link);
?>
```

See Also

```
mysqli_autocommit  
mysqli_rollback
```

22.9.3.9.9. [mysqli::\\$connect_errno](#), [mysqli_connect_errno](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$connect_errno](#)

```
mysqli_connect_errno
```

Returns the error code from last connect call

Description

Object oriented style

```
string  
mysqli->connect_errno ;
```

Procedural style

```
int mysqli_connect_errno();
```

Returns the last error code number from the last call to [mysqli_connect](#).



Note

Client error message numbers are listed in the MySQL [errmsg.h](#) header file, server error message numbers are listed in [mysqld_error.h](#). In the MySQL source distribution you can find a complete list of error messages and error numbers in the file [Docs/mysqld_error.txt](#).

Return Values

An error code value for the last call to [mysqli_connect](#), if it failed. zero means no error occurred.

Examples

Example 22.113. [\\$mysqli->connect_errno](#) example

Object oriented style

```
<?php  
$mysqli = @new mysqli('localhost', 'fake_user', 'my_password', 'my_db');  
  
if ($mysqli->connect_errno) {  
    die('Connect Error: ' . $mysqli->connect_errno);  
}  
?>
```

Procedural style

```
<?php  
$link = @mysqli_connect('localhost', 'fake_user', 'my_password', 'my_db');
```

```
if (!$link) {
    die('Connect Error: ' . mysqli_connect_errno());
}
?>
```

The above examples will output:

```
Connect Error: 1045
```

See Also

[mysqli_connect](#)
[mysqli_connect_error](#)
[mysqli_errno](#)
[mysqli_error](#)
[mysqli_sqlstate](#)

22.9.3.9.10. [mysqli::\\$connect_error](#), [mysqli_connect_error](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$connect_error](#)

[mysqli_connect_error](#)

Returns a string description of the last connect error

Description

Object oriented style

```
string
mysqli->connect_error ;
```

Procedural style

```
string mysqli_connect_error();
```

Returns the last error message string from the last call to [mysqli_connect](#).

Return Values

A string that describes the error. [NULL](#) is returned if no error occurred.

Examples

Example 22.114. [\\$mysqli->connect_error](#) example

Object oriented style

```
<?php
$mysqli = @new mysqli('localhost', 'fake_user', 'my_password', 'my_db');

// Works as of PHP 5.2.9 and 5.3.0.
```

```
if ($mysqli->connect_error) {
    die('Connect Error: ' . $mysqli->connect_error);
}
?>
```

Procedural style

```
<?php
$link = @mysqli_connect('localhost', 'fake_user', 'my_password', 'my_db');

if (!$link) {
    die('Connect Error: ' . mysqli_connect_error());
}
?>
```

The above examples will output:

```
Connect Error: Access denied for user 'fake_user'@'localhost' (using password: YES)
```

Notes



Warning

The `mysqli->connect_error` property only works properly as of PHP versions 5.2.9 and 5.3.0. Use the [`mysqli_connect_error`](#) function if compatibility with earlier PHP versions is required.

See Also

[`mysqli_connect`](#)
[`mysqli_connect_errno`](#)
[`mysqli_errno`](#)
[`mysqli_error`](#)
[`mysqli_sqlstate`](#)

22.9.3.9.11. [`mysqli::__construct`](#), [`mysqli_connect`](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [`mysqli::__construct`](#)

[`mysqli_connect`](#)

Open a new connection to the MySQL server

Description

Object oriented style

```
mysqli::__construct(
    string host
        = =ini_get("mysqli.default_host"),
    string username
```

```

    = =ini_get("mysqli.default_user"),
string passwd
    = =ini_get("mysqli.default_pw"),
string dbname
    = "=",
int port
    = =ini_get("mysqli.default_port"),
string socket
    = =ini_get("mysqli.default_socket");

```

Procedural style

```

mysqli mysqli_connect(
    string host
        = =ini_get("mysqli.default_host"),
    string username
        = =ini_get("mysqli.default_user"),
    string passwd
        = =ini_get("mysqli.default_pw"),
    string dbname
        = "=",
    int port
        = =ini_get("mysqli.default_port"),
    string socket
        = =ini_get("mysqli.default_socket"));

```

Opens a connection to the MySQL Server running on.

Parameters

host

Can be either a host name or an IP address. Passing the [NULL](#) value or the string "localhost" to this parameter, the local host is assumed. When possible, pipes will be used instead of the TCP/IP protocol.

Prepending host by [p:](#) opens a persistent connection.

[mysqli_change_user](#) is automatically called on connections opened from the connection pool.

username

The MySQL user name.

passwd

If not provided or [NULL](#), the MySQL server will attempt to authenticate the user against those user records which have no password only. This allows one username to be used with different permissions (depending on if a password as provided or not).

dbname

If provided will specify the default database to be used when performing queries.

port

Specifies the port number to attempt to connect to the MySQL server.

socket

Specifies the socket or named pipe that should be used.



Note

Specifying the [socket](#) parameter will not explicitly determine the type of connection to be used when connecting to the MySQL server. How the connection is made to the MySQL database is determined by the [host](#) parameter.

Return Values

Returns an object which represents the connection to a MySQL Server.

Changelog

Version	Description
5.3.0	Added the ability of persistent connections.

Examples

Example 22.115. `mysqli::__construct` example

Object oriented style

```
<?php
$mysqli = new mysqli('localhost', 'my_user', 'my_password', 'my_db');

/*
 * This is the "official" OO way to do it,
 * BUT $connect_error was broken until PHP 5.2.9 and 5.3.0.
 */
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
        . $mysqli->connect_error);
}

/*
 * Use this instead of $connect_error if you need to ensure
 * compatibility with PHP versions prior to 5.2.9 and 5.3.0.
 */
if (mysqli_connect_error()) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}

echo 'Success... ' . $mysqli->host_info . "\n";

$mysqli->close();
?>
```

Object oriented style when extending mysqli class

```
<?php

class foo_mysqli extends mysqli {
    public function __construct($host, $user, $pass, $db) {
        parent::__construct($host, $user, $pass, $db);

        if (mysqli_connect_error()) {
            die('Connect Error (' . mysqli_connect_errno() . ') '
                . mysqli_connect_error());
        }
    }
}

$db = new foo_mysqli('localhost', 'my_user', 'my_password', 'my_db');

echo 'Success... ' . $db->host_info . "\n";

$db->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'my_db');

if (!$link) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}

echo 'Success... ' . mysqli_get_host_info($link) . "\n";

mysqli_close($link);
?>
```

The above examples will output:

```
Success... MySQL host info: localhost via TCP/IP
```

Notes

**Note**

MySQLnd always assumes the server default charset. This charset is sent during connection hand-shake/authentication, which mysqlnd will use.

Libmysql uses the default charset set in the [my.cnf](#) or by an explicit call to [mysqli_options](#) prior to calling [mysqli_real_connect](#), but after [mysqli_init](#).

**Note**

OO syntax only: If a connection fails an object is still returned. To check if the connection failed then use either the [mysqli_connect_error](#) function or the [mysqli->connect_error](#) property as in the preceding examples.

**Note**

If it is necessary to set options, such as the connection timeout, [mysqli_real_connect](#) must be used instead.

**Note**

Calling the constructor with no parameters is the same as calling [mysqli_init](#).

**Note**

Error "Can't create TCP/IP socket (10106)" usually means that the [variables_order](#) configure directive doesn't contain character **E**. On Windows, if the environment is not copied the [SYSTEMROOT](#) environment variable won't be available and PHP will have problems loading Winsock.

See Also

[mysqli_real_connect](#)
[mysqli_options](#)
[mysqli_connect_errno](#)
[mysqli_connect_error](#)
[mysqli_close](#)

22.9.3.9.12. [mysqli::debug](#), [mysqli_debug](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::debug](#)

[mysqli_debug](#)

Performs debugging operations

Description

Object oriented style

```
bool mysqli::debug(  
    string message);
```

Procedural style

```
bool mysqli_debug(  
    string message);
```

Performs debugging operations using the Fred Fish debugging library.

Parameters

message A string representing the debugging operation to perform

Return Values

Returns [TRUE](#) .

Notes



Note

To use the [mysqli_debug](#) function you must compile the MySQL client library to support debugging.

Examples

Example 22.116. Generating a Trace File

```
<?php  
  
/* Create a trace file in '/tmp/client.trace' on the local (client) machine: */  
mysqli_debug("d:t:o,/tmp/client.trace");  
  
?>
```

See Also

[mysqli_dump_debug_info](#)
[mysqli_report](#)

22.9.3.9.13. [mysqli::dump_debug_info](#), [mysqli_dump_debug_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::dump_debug_info](#)
[mysqli_dump_debug_info](#)

Dump debugging information into the log

Description

Object oriented style

```
bool mysqli::dump_debug_info();
```

Procedural style

```
bool mysqli_dump_debug_info(  
    mysqli link);
```

This function is designed to be executed by a user with the SUPER privilege and is used to dump debugging information into the log for the MySQL Server relating to the connection.

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

See Also

[mysqli_debug](#)

22.9.3.9.14. [mysqli::\\$errno](#), [mysqli_errno](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$errno](#)
[mysqli_errno](#)

Returns the error code for the most recent function call

Description

Object oriented style

```
int  
mysqli->errno ;
```

Procedural style

```
int mysqli_errno(
    mysqli link);
```

Returns the last error code for the most recent MySQLi function call that can succeed or fail.

Client error message numbers are listed in the MySQL [errmsg.h](#) header file, server error message numbers are listed in [mysqld_error.h](#). In the MySQL source distribution you can find a complete list of error messages and error numbers in the file [Docs/mysqld_error.txt](#).

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

An error code value for the last call, if it failed. zero means no error occurred.

Examples

Example 22.117. [\\$mysqli->errno](#) example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}

if (!$mysqli->query("SET a=1")) {
    printf("Errorcode: %d\n", $mysqli->errno);
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if (!mysqli_query($link, "SET a=1")) {
    printf("Errorcode: %d\n", mysqli_errno($link));
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Errorcode: 1193
```

See Also

```
mysqli_connect_errno  
mysqli_connect_error  
mysqli_error  
mysqli_sqlstate
```

22.9.3.9.15. `mysqli::$error_list`, `mysqli_error_list`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::$error_list`

```
mysqli_error_list
```

Returns a list of errors from the last command executed

Description

Object oriented style

```
array  
mysqli->error_list ;
```

Procedural style

```
array mysqli_error_list(  
mysqli link);
```

Returns a array of errors for the most recent MySQLi function call that can succeed or fail.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

A list of errors, each as an associative array containing the errno, error, and sqlstate.

Examples

Example 22.118. `$mysqli->error_list` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "nobody", "");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if (!$mysqli->query("SET a=1")) {
    print_r($mysqli->error_list);
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if (!mysqli_query($link, "SET a=1")) {
    print_r(mysqli_error_list($link));
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Array
(
    [0] => Array
        (
            [errno] => 1193
            [sqlstate] => HY000
            [error] => Unknown system variable 'a'
        )
)
```

See Also

[mysqli_connect_errno](#)
[mysqli_connect_error](#)
[mysqli_error](#)
[mysqli_sqlstate](#)

22.9.3.9.16. [mysqli::\\$error](#), [mysqli_error](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::$error`

`mysqli_error`

Returns a string description of the last error

Description

Object oriented style

```
string  
mysqli->error ;
```

Procedural style

```
string mysqli_error(  
mysqli link);
```

Returns the last error message for the most recent MySQLi function call that can succeed or fail.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

A string that describes the error. An empty string if no error occurred.

Examples

Example 22.119. `$mysqli->error` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if ($mysqli->connect_errno) {  
    printf("Connect failed: %s\n", $mysqli->connect_error);  
    exit();  
}  
  
if (!$mysqli->query("SET a=1")) {  
    printf("Error message: %s\n", $mysqli->error);  
}  
  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if (!mysqli_query($link, "SET a=1")) {
    printf("Errormessage: %s\n", mysqli_error($link));
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Errormessage: Unknown system variable 'a'
```

See Also

[mysqli_connect_errno](#)
[mysqli_connect_error](#)
[mysqli_errno](#)
[mysqli_sqlstate](#)

22.9.3.9.17. [mysqli::\\$field_count](#), [mysqli_field_count](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$field_count](#)

[mysqli_field_count](#)

Returns the number of columns for the most recent query

Description

Object oriented style

```
int
mysqli->field_count ;
```

Procedural style

```
int mysqli_field_count(
mysqli link);
```

Returns the number of columns for the most recent query on the connection represented by the [link](#) parameter. This function can be useful when using the [mysqli_store_result](#) function to determine if the query should have produced a non-empty result set or not without knowing the nature of the query.

Parameters

[link](#)

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

An integer representing the number of fields in a result set.

Examples

Example 22.120. `$mysqli->field_count` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");

$mysqli->query( "DROP TABLE IF EXISTS friends");
$mysqli->query( "CREATE TABLE friends (id int, name varchar(20))");

$mysqli->query( "INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");

$mysqli->real_query("SELECT * FROM friends");

if ($mysqli->field_count) {
    /* this was a select/show or describe query */
    $result = $mysqli->store_result();

    /* process resultset */
    $row = $result->fetch_row();

    /* free resultset */
    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");

mysqli_query($link, "DROP TABLE IF EXISTS friends");
mysqli_query($link, "CREATE TABLE friends (id int, name varchar(20))");

mysqli_query($link, "INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");

mysqli_real_query($link, "SELECT * FROM friends");

if (mysqli_field_count($link)) {
    /* this was a select/show or describe query */
    $result = mysqli_store_result($link);

    /* process resultset */
    $row = mysqli_fetch_row($result);

    /* free resultset */
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
```

```
?>
```

22.9.3.9.18. *mysqli::get_charset*, *mysqli_get_charset*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysqli::get_charset*

mysqli_get_charset

Returns a character set object

Description

Object oriented style

```
object mysqli::get_charset();
```

Procedural style

```
object mysqli_get_charset(
    mysqli link);
```

Returns a character set object providing several properties of the current active character set.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by <i>mysqli_connect</i> or <i>mysqli_init</i>
-------------	--

Return Values

The function returns a character set object with the following properties:

<i>charset</i>	Character set name
<i>collation</i>	Collation name
<i>dir</i>	Directory the charset description was fetched from (?) or "" for built-in character sets
<i>min_length</i>	Minimum character length in bytes
<i>max_length</i>	Maximum character length in bytes
<i>number</i>	Internal character set number
<i>state</i>	Character set status (?)

Examples

Example 22.121. *mysqli::get_charset* example

Object oriented style

```
<?php
$db = mysqli_init();
$db->real_connect("localhost","root","","test");
var_dump($db->get_charset());
?>
```

Procedural style

```
<?php
$db = mysqli_init();
mysqli_real_connect($db, "localhost","root","","test");
var_dump($db->get_charset());
?>
```

The above examples will output:

```
object(stdClass)#2 (7) {
  ["charset"]=>
  string(6) "latin1"
  ["collation"]=>
  string(17) "latin1_swedish_ci"
  ["dir"]=>
  string(0) ""
  ["min_length"]=>
  int(1)
  ["max_length"]=>
  int(1)
  ["number"]=>
  int(8)
  ["state"]=>
  int(801)
}
```

See Also

[mysqli_character_set_name](#)
[mysqli_set_charset](#)

22.9.3.9.19. [mysqli::get_client_info](#), [mysqli_get_client_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::get_client_info](#)

[mysqli_get_client_info](#)

Get MySQL client info

Description

Object oriented style

```
string mysqli::get_client_info();
```

Procedural style

```
string mysqli_get_client_info(  
    mysqli link);
```

Returns a string that represents the MySQL client library version.

Return Values

A string that represents the MySQL client library version

Examples

Example 22.122. `mysqli_get_client_info`

```
<?php  
  
/* We don't need a connection to determine  
   the version of mysql client library */  
  
printf("Client library version: %s\n", mysqli_get_client_info());  
?>
```

See Also

[mysqli_get_client_version](#)
[mysqli_get_server_info](#)
[mysqli_get_server_version](#)

22.9.3.9.20. `mysqli_get_client_stats`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_get_client_stats](#)

Returns client per-process statistics

Description

```
array mysqli_get_client_stats();
```

Returns client per-process statistics. Available only with [mysqlnd](#).

Parameters

Return Values

Returns an array with client stats if success, [FALSE](#) otherwise.

Examples

Example 22.123. A `mysqli_get_client_stats` example

```
<?php  
$link = mysqli_connect();  
print_r(mysqli_get_client_stats());  
?>
```

The above example will output something similar to:

```
Array
(
    [bytes_sent] => 43
    [bytes_received] => 80
    [packets_sent] => 1
    [packets_received] => 2
    [protocol_overhead_in] => 8
    [protocol_overhead_out] => 4
    [bytes_received_ok_packet] => 11
    [bytes_received_eof_packet] => 0
    [bytes_received_rset_header_packet] => 0
    [bytes_received_rset_field_meta_packet] => 0
    [bytes_received_rset_row_packet] => 0
    [bytes_received_prepare_response_packet] => 0
    [bytes_received_change_user_packet] => 0
    [packets_sent_command] => 0
    [packets_received_ok] => 1
    [packets_received_eof] => 0
    [packets_received_rset_header] => 0
    [packets_received_rset_field_meta] => 0
    [packets_received_rset_row] => 0
    [packets_received_prepare_response] => 0
    [packets_received_change_user] => 0
    [result_set_queries] => 0
    [non_result_set_queries] => 0
    [no_index_used] => 0
    [bad_index_used] => 0
    [slow_queries] => 0
    [buffered_sets] => 0
    [unbuffered_sets] => 0
    [ps_buffered_sets] => 0
    [ps_unbuffered_sets] => 0
    [flushed_normal_sets] => 0
    [flushed_ps_sets] => 0
    [ps_prepared_never_executed] => 0
    [ps_prepared_once_executed] => 0
    [rows_fetched_from_server_normal] => 0
    [rows_fetched_from_server_ps] => 0
    [rows_buffered_from_client_normal] => 0
    [rows_buffered_from_client_ps] => 0
    [rows_fetched_from_client_normal_buffered] => 0
    [rows_fetched_from_client_normal_unbuffered] => 0
    [rows_fetched_from_client_ps_buffered] => 0
    [rows_fetched_from_client_ps_unbuffered] => 0
    [rows_fetched_from_client_ps_cursor] => 0
    [rows_skipped_normal] => 0
    [rows_skipped_ps] => 0
    [copy_on_write_saved] => 0
    [copy_on_write_performed] => 0
    [command_buffer_too_small] => 0
    [connect_success] => 1
    [connect_failure] => 0
    [connection_reused] => 0
    [reconnect] => 0
    [pconnect_success] => 0
    [active_connections] => 1
    [active_persistent_connections] => 0
    [explicit_close] => 0
    [implicit_close] => 0
    [disconnect_close] => 0
    [in_middle_of_command_close] => 0
)
```

```
[explicit_free_result] => 0
[implicit_free_result] => 0
[explicit_stmt_close] => 0
[implicit_stmt_close] => 0
[mem_emalloc_count] => 0
[mem_emalloc_ammount] => 0
[mem_ecalloc_count] => 0
[mem_ecalloc_ammount] => 0
[mem_erealloc_count] => 0
[mem_erealloc_ammount] => 0
[mem_efree_count] => 0
[mem_malloc_count] => 0
[mem_malloc_ammount] => 0
[mem_calloc_count] => 0
[mem_calloc_ammount] => 0
[mem_realloc_count] => 0
[mem_realloc_ammount] => 0
[mem_free_count] => 0
[proto_text_fetched_null] => 0
[proto_text_fetched_bit] => 0
[proto_text_fetched_tinyint] => 0
[proto_text_fetched_short] => 0
[proto_text_fetched_int24] => 0
[proto_text_fetched_int] => 0
[proto_text_fetched_bigint] => 0
[proto_text_fetched_decimal] => 0
[proto_text_fetched_float] => 0
[proto_text_fetched_double] => 0
[proto_text_fetched_date] => 0
[proto_text_fetched_year] => 0
[proto_text_fetched_time] => 0
[proto_text_fetched_datetime] => 0
[proto_text_fetched_timestamp] => 0
[proto_text_fetched_string] => 0
[proto_text_fetched_blob] => 0
[proto_text_fetched_enum] => 0
[proto_text_fetched_set] => 0
[proto_text_fetched_geometry] => 0
[proto_text_fetched_other] => 0
[proto_binary_fetched_null] => 0
[proto_binary_fetched_bit] => 0
[proto_binary_fetched_tinyint] => 0
[proto_binary_fetched_short] => 0
[proto_binary_fetched_int24] => 0
[proto_binary_fetched_int] => 0
[proto_binary_fetched_bigint] => 0
[proto_binary_fetched_decimal] => 0
[proto_binary_fetched_float] => 0
[proto_binary_fetched_double] => 0
[proto_binary_fetched_date] => 0
[proto_binary_fetched_year] => 0
[proto_binary_fetched_time] => 0
[proto_binary_fetched_datetime] => 0
[proto_binary_fetched_timestamp] => 0
[proto_binary_fetched_string] => 0
[proto_binary_fetched_blob] => 0
[proto_binary_fetched_enum] => 0
[proto_binary_fetched_set] => 0
[proto_binary_fetched_geometry] => 0
[proto_binary_fetched_other] => 0
)
```

See Also[Stats description](#)

22.9.3.9.21. `mysqli_get_client_version`, `mysqli::$client_version`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_get_client_version`

`mysqli::$client_version`

Returns the MySQL client version as a string

Description

Object oriented style

```
int
mysqli->client_version ;
```

Procedural style

```
int mysqli_get_client_version(
mysqli link);
```

Returns client version number as an integer.

Return Values

A number that represents the MySQL client library version in format: `main_version*10000 + minor_version *100 + sub_version`. For example, 4.1.0 is returned as 40100.

This is useful to quickly determine the version of the client library to know if some capability exists.

Examples

Example 22.124. `mysqli_get_client_version`

```
<?php
/* We don't need a connection to determine
   the version of mysql client library */
printf("Client library version: %d\n", mysqli_get_client_version());
?>
```

See Also

`mysqli_get_client_info`
`mysqli_get_server_info`
`mysqli_get_server_version`

22.9.3.9.22. `mysqli::get_connection_stats`, `mysqli_get_connection_stats`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::get_connection_stats`

`mysqli_get_connection_stats`

Returns statistics about the client connection

Description

Object oriented style

```
bool mysqli::get_connection_stats();
```

Procedural style

```
array mysqli_get_connection_stats(  
    mysqli link);
```

Returns statistics about the client connection. Available only with [mysqlnd](#).

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns an array with connection stats if success, [FALSE](#) otherwise.

Examples

Example 22.125. A [mysqli_get_connection_stats](#) example

```
<?php  
$link = mysqli_connect();  
print_r(mysqli_get_connection_stats($link));  
?>
```

The above example will output something similar to:

```
Array  
(  
    [bytes_sent] => 43  
    [bytes_received] => 80  
    [packets_sent] => 1  
    [packets_received] => 2  
    [protocol_overhead_in] => 8  
    [protocol_overhead_out] => 4  
    [bytes_received_ok_packet] => 11  
    [bytes_received_eof_packet] => 0  
    [bytes_received_rset_header_packet] => 0  
    [bytes_received_rset_field_meta_packet] => 0  
    [bytes_received_rset_row_packet] => 0  
    [bytes_received_prepare_response_packet] => 0  
    [bytes_received_change_user_packet] => 0  
    [packets_sent_command] => 0  
    [packets_received_ok] => 1  
    [packets_received_eof] => 0  
    [packets_received_rset_header] => 0  
    [packets_received_rset_field_meta] => 0  
    [packets_received_rset_row] => 0  
    [packets_received_prepare_response] => 0  
    [packets_received_change_user] => 0  
    [result_set_queries] => 0  
    [non_result_set_queries] => 0  
    [no_index_used] => 0  
    [bad_index_used] => 0  
    [slow_queries] => 0  
    [buffered_sets] => 0  
    [unbuffered_sets] => 0  
    [ps_buffered_sets] => 0
```

```
[ps_unbuffered_sets] => 0
[flushed_normal_sets] => 0
[flushed_ps_sets] => 0
[ps_prepared_never_executed] => 0
[ps_prepared_once_executed] => 0
[rows_fetched_from_server_normal] => 0
[rows_fetched_from_server_ps] => 0
[rows_buffered_from_client_normal] => 0
[rows_buffered_from_client_ps] => 0
[rows_fetched_from_client_normal_buffered] => 0
[rows_fetched_from_client_normal_unbuffered] => 0
[rows_fetched_from_client_ps_buffered] => 0
[rows_fetched_from_client_ps_unbuffered] => 0
[rows_fetched_from_client_ps_cursor] => 0
[rows_skipped_normal] => 0
[rows_skipped_ps] => 0
[copy_on_write_saved] => 0
[copy_on_write_performed] => 0
[command_buffer_too_small] => 0
[connect_success] => 1
[connect_failure] => 0
[connection_reused] => 0
[reconnect] => 0
[pconnect_success] => 0
[active_connections] => 1
[active_persistent_connections] => 0
[explicit_close] => 0
[implicit_close] => 0
[disconnect_close] => 0
[in_middle_of_command_close] => 0
[explicit_free_result] => 0
[implicit_free_result] => 0
[explicit_stmt_close] => 0
[implicit_stmt_close] => 0
[mem_emalloc_count] => 0
[mem_emalloc_ammount] => 0
[mem_ecalloc_count] => 0
[mem_ecalloc_ammount] => 0
[mem_erealloc_count] => 0
[mem_erealloc_ammount] => 0
[mem_efree_count] => 0
[mem_malloc_count] => 0
[mem_malloc_ammount] => 0
[mem_calloc_count] => 0
[mem_calloc_ammount] => 0
[mem_realloc_count] => 0
[mem_realloc_ammount] => 0
[mem_free_count] => 0
[proto_text_fetched_null] => 0
[proto_text_fetched_bit] => 0
[proto_text_fetched_tinyint] => 0
[proto_text_fetched_short] => 0
[proto_text_fetched_int24] => 0
[proto_text_fetched_int] => 0
[proto_text_fetched_bigint] => 0
[proto_text_fetched_decimal] => 0
[proto_text_fetched_float] => 0
[proto_text_fetched_double] => 0
[proto_text_fetched_date] => 0
[proto_text_fetched_year] => 0
[proto_text_fetched_time] => 0
[proto_text_fetched_datetime] => 0
[proto_text_fetched_timestamp] => 0
[proto_text_fetched_string] => 0
[proto_text_fetched_blob] => 0
[proto_text_fetched_enum] => 0
[proto_text_fetched_set] => 0
```

```
[proto_text_fetched_geometry] => 0
[proto_text_fetched_other] => 0
[proto_binary_fetched_null] => 0
[proto_binary_fetched_bit] => 0
[proto_binary_fetched_tinyint] => 0
[proto_binary_fetched_short] => 0
[proto_binary_fetched_int24] => 0
[proto_binary_fetched_int] => 0
[proto_binary_fetched_bigint] => 0
[proto_binary_fetched_decimal] => 0
[proto_binary_fetched_float] => 0
[proto_binary_fetched_double] => 0
[proto_binary_fetched_date] => 0
[proto_binary_fetched_year] => 0
[proto_binary_fetched_time] => 0
[proto_binary_fetched_datetime] => 0
[proto_binary_fetched_timestamp] => 0
[proto_binary_fetched_string] => 0
[proto_binary_fetched_blob] => 0
[proto_binary_fetched_enum] => 0
[proto_binary_fetched_set] => 0
[proto_binary_fetched_geometry] => 0
[proto_binary_fetched_other] => 0
)
```

See Also

[Stats description](#)

22.9.3.9.23. [mysqli::\\$host_info](#), [mysqli_get_host_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$host_info](#)

[mysqli_get_host_info](#)

Returns a string representing the type of connection used

Description

Object oriented style

```
string
mysqli->host_info ;
```

Procedural style

```
string mysqli_get_host_info(
    mysqli link);
```

Returns a string describing the connection represented by the *link* parameter (including the server host name).

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

A character string representing the server hostname and the connection type.

Examples

Example 22.126. `$mysqli->host_info` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* print host information */
printf("Host info: %s\n", $mysqli->host_info);

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* print host information */
printf("Host info: %s\n", mysqli_get_host_info($link));

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Host info: Localhost via UNIX socket
```

See Also

[mysqli_get_proto_info](#)

22.9.3.9.24. [mysqli::\\$protocol_version](#), [mysqli_get_proto_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::$protocol_version`

`mysqli_get_proto_info`

Returns the version of the MySQL protocol used

Description

Object oriented style

```
string
mysqli->protocol_version ;
```

Procedural style

```
int mysqli_get_proto_info(
mysqli link);
```

Returns an integer representing the MySQL protocol version used by the connection represented by the *link* parameter.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns an integer representing the protocol version.

Examples

Example 22.127. `$mysqli->protocol_version` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* print protocol version */
printf("Protocol version: %d\n", $mysqli->protocol_version);

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
```

```
$link = mysqli_connect("localhost", "my_user", "my_password");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* print protocol version */
printf("Protocol version: %d\n", mysqli_get_proto_info($link));

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Protocol version: 10
```

See Also

[mysqli_get_host_info](#)

22.9.3.9.25. [mysqli::\\$server_info](#), [mysqli_get_server_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$server_info](#)

[mysqli_get_server_info](#)

Returns the version of the MySQL server

Description

Object oriented style

```
string
mysqli->server_info ;
```

Procedural style

```
string mysqli_get_server_info(
    mysqli link);
```

Returns a string representing the version of the MySQL server that the MySQLi extension is connected to.

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

A character string representing the server version.

Examples

Example 22.128. `$mysqli->server_info` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* print server version */
printf("Server version: %s\n", $mysqli->server_info);

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* print server version */
printf("Server version: %s\n", mysqli_get_server_info($link));

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Server version: 4.1.2-alpha-debug
```

See Also

[mysqli_get_client_info](#)
[mysqli_get_client_version](#)
[mysqli_get_server_version](#)

22.9.3.9.26. `mysqli::$server_version`, `mysqli_get_server_version`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::$server_version`

`mysqli_get_server_version`

Returns the version of the MySQL server as an integer

Description

Object oriented style

```
int
mysqli->server_version ;
```

Procedural style

```
int mysqli_get_server_version(
mysqli link);
```

The `mysqli_get_server_version` function returns the version of the server connected to (represented by the *link* parameter) as an integer.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

An integer representing the server version.

The form of this version number is `main_version * 10000 + minor_version * 100 + sub_version` (i.e. version 4.1.0 is 40100).

Examples

Example 22.129. `mysqli->server_version` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* print server version */
printf("Server version: %d\n", $mysqli->server_version);

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* print server version */
printf("Server version: %d\n", mysqli_get_server_version($link));

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Server version: 40102
```

See Also

[mysqli_get_client_info](#)
[mysqli_get_client_version](#)
[mysqli_get_server_info](#)

22.9.3.9.27. [mysqli::get_warnings](#), [mysqli_get_warnings](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::get_warnings](#)

[mysqli_get_warnings](#)

Get result of SHOW WARNINGS

Description

Object oriented style

```
mysqli_warning mysqli::get_warnings();
```

Procedural style

```
mysqli_warning mysqli_get_warnings(
    mysqli link);
```



Warning

This function is currently not documented; only its argument list is available.

22.9.3.9.28. [mysqli::\\$info](#), [mysqli_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::$info`

`mysqli_info`

Retrieves information about the most recently executed query

Description

Object oriented style

```
string
mysqli->info ;
```

Procedural style

```
string mysqli_info(
    mysqli link);
```

The `mysqli_info` function returns a string providing information about the last query executed. The nature of this string is provided below:

Table 22.44. Possible `mysqli_info` return values

Query type	Example result string
INSERT INTO...SELECT...	Records: 100 Duplicates: 0 Warnings: 0
INSERT INTO...VALUES (...),(...),(...)	Records: 3 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...	Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE ...	Records: 3 Duplicates: 0 Warnings: 0
UPDATE ...	Rows matched: 40 Changed: 40 Warnings: 0



Note

Queries which do not fall into one of the preceding formats are not supported. In these situations, `mysqli_info` will return an empty string.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

A character string representing additional information about the most recently executed query.

Examples

Example 22.130. `$mysqli->info` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
```

```
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$mysqli->query("CREATE TEMPORARY TABLE t1 LIKE City");

/* INSERT INTO .. SELECT */
$mysqli->query("INSERT INTO t1 SELECT * FROM City ORDER BY ID LIMIT 150");
printf("%s\n", $mysqli->info);

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

mysqli_query($link, "CREATE TEMPORARY TABLE t1 LIKE City");

/* INSERT INTO .. SELECT */
mysqli_query($link, "INSERT INTO t1 SELECT * FROM City ORDER BY ID LIMIT 150");
printf("%s\n", mysqli_info($link));

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Records: 150 Duplicates: 0 Warnings: 0
```

See Also

[mysqli_affected_rows](#)
[mysqli_warning_count](#)
[mysqli_num_rows](#)

22.9.3.9.29. [mysqli::init](#), [mysqli_init](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::init](#)

[mysqli_init](#)

Initializes MySQLi and returns a resource for use with [mysqli_real_connect\(\)](#)

Description

Object oriented style

```
mysqli mysqli::init();
```

Procedural style

```
mysqli mysqli_init();
```

Allocates or initializes a MySQL object suitable for [mysqli_options](#) and [mysqli_real_connect](#).



Note

Any subsequent calls to any mysqli function (except [mysqli_options](#)) will fail until [mysqli_real_connect](#) was called.

Return Values

Returns an object.

Examples

See [mysqli_real_connect](#).

See Also

[mysqli_options](#)
[mysqli_close](#)
[mysqli_real_connect](#)
[mysqli_connect](#)

22.9.3.9.30. [mysqli::\\$insert_id](#), [mysqli_insert_id](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$insert_id](#)

[mysqli_insert_id](#)

Returns the auto generated id used in the last query

Description

Object oriented style

```
mixed  
mysqli->insert_id ;
```

Procedural style

```
mixed mysqli_insert_id(  
    mysqli link);
```

The [mysqli_insert_id](#) function returns the ID generated by a query on a table with a column having the AUTO_INCREMENT attribute. If the last query wasn't an INSERT or UPDATE statement or if the modified table does not have a column with the AUTO_INCREMENT attribute, this function will return zero.

**Note**

Performing an INSERT or UPDATE statement using the LAST_INSERT_ID() function will also modify the value returned by the `mysqli_insert_id` function.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

The value of the `AUTO_INCREMENT` field that was updated by the previous query. Returns zero if there was no previous query on the connection or if the query did not update an `AUTO_INCREMENT` value.

**Note**

If the number is greater than maximal int value, `mysqli_insert_id` will return a string.

Examples**Example 22.131. `$mysqli->insert_id` example****Object oriented style**

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$mysqli->query("CREATE TABLE myCity LIKE City");

$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
$mysqli->query($query);

printf ("New Record has id %d.\n", $mysqli->insert_id);

/* drop table */
$mysqli->query("DROP TABLE myCity");

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
```

```
    exit();
}

mysqli_query($link, "CREATE TABLE myCity LIKE City");

$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
mysqli_query($link, $query);

printf ("New Record has id %d.\n", mysqli_insert_id($link));

/* drop table */
mysqli_query($link, "DROP TABLE myCity");

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
New Record has id 1.
```

22.9.3.9.31. [mysqli::kill](#), [mysqli_kill](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::kill](#)

[mysqli_kill](#)

Asks the server to kill a MySQL thread

Description

Object oriented style

```
bool mysqli::kill(
    int processid);
```

Procedural style

```
bool mysqli_kill(
    mysqli link,
    int processid);
```

This function is used to ask the server to kill a MySQL thread specified by the *processid* parameter. This value must be retrieved by calling the [mysqli_thread_id](#) function.

To stop a running query you should use the SQL command `KILL QUERY processid`.

Parameters

link

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.132. `mysqli::kill` example

Object oriented style

```
<?php
mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* determine our thread id */
$thread_id = $mysqli->thread_id;

/* Kill connection */
mysqli->kill($thread_id);

/* This should produce an error */
if (!$mysqli->query("CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", $mysqli->error);
    exit;
}

/* close connection */
mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* determine our thread id */
$thread_id = mysqli_thread_id($link);

/* Kill connection */
mysqli_kill($link, $thread_id);

/* This should produce an error */
if (!$mysqli_query($link, "CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", mysqli_error($link));
    exit;
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: MySQL server has gone away
```

See Also

[mysqli_thread_id](#)

22.9.3.9.32. [mysqli::more_results](#), [mysqli_more_results](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::more_results](#)

[mysqli_more_results](#)

Check if there are any more query results from a multi query

Description

Object oriented style

```
bool mysqli::more_results();
```

Procedural style

```
bool mysqli_more_results(  
    mysqli link);
```

Indicates if one or more result sets are available from a previous call to [mysqli_multi_query](#).

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

See [mysqli_multi_query](#).

See Also

[mysqli_multi_query](#)
[mysqli_next_result](#)
[mysqli_store_result](#)
[mysqli_use_result](#)

22.9.3.9.33. [mysqli::multi_query](#), [mysqli_multi_query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::multi_query`

`mysqli_multi_query`

Performs a query on the database

Description

Object oriented style

```
bool mysqli::multi_query(  
    string query);
```

Procedural style

```
bool mysqli_multi_query(  
    mysqli link,  
    string query);
```

Executes one or multiple queries which are concatenated by a semicolon.

To retrieve the resultset from the first query you can use `mysqli_use_result` or `mysqli_store_result`. All subsequent query results can be processed using `mysqli_more_results` and `mysqli_next_result`.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by <code>mysqli_connect</code> or <code>mysqli_init</code>
<i>query</i>	The query, as a string. Data inside the query should be properly escaped .

Return Values

Returns `FALSE` if the first statement failed. To retrieve subsequent errors from other statements you have to call `mysqli_next_result` first.

Examples

Example 22.133. `mysqli::multi_query` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
  
$query = "SELECT CURRENT_USER();";  
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
```

```

/* execute multi query */
if ($mysqli->multi_query($query)) {
    do {
        /* store first result set */
        if ($result = $mysqli->store_result()) {
            while ($row = $result->fetch_row()) {
                printf("%s\n", $row[0]);
            }
            $result->free();
        }
        /* print divider */
        if ($mysqli->more_results()) {
            printf("-----\n");
        }
    } while ($mysqli->next_result());
}

/* close connection */
$mysqli->close();
?>

```

Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT CURRENT_USER()";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";

/* execute multi query */
if (mysqli_multi_query($link, $query)) {
    do {
        /* store first result set */
        if ($result = mysqli_store_result($link)) {
            while ($row = mysqli_fetch_row($result)) {
                printf("%s\n", $row[0]);
            }
            mysqli_free_result($result);
        }
        /* print divider */
        if (mysqli_more_results($link)) {
            printf("-----\n");
        }
    } while (mysqli_next_result($link));
}

/* close connection */
mysqli_close($link);
?>

```

The above examples will output something similar to:

```

my_user@localhost
-----
Amersfoort

```

```
Maastricht
Dordrecht
Leiden
Haarlemmermeer
```

See Also

```
mysqli_query
mysqli_use_result
mysqli_store_result
mysqli_next_result
mysqli_more_results
```

22.9.3.9.34. [mysqli::next_result](#), [mysqli_next_result](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::next_result](#)

```
mysqli_next_result
```

Prepare next result from `multi_query`

Description

Object oriented style

```
bool mysqli::next_result();
```

Procedural style

```
bool mysqli_next_result(
    mysqli link);
```

Prepares next result set from a previous call to [mysqli_multi_query](#) which can be retrieved by [mysqli_store_result](#) or [mysqli_use_result](#).

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

See [mysqli_multi_query](#).

See Also

```
mysqli_multi_query
mysqli_more_results
mysqli_store_result
```

`mysqli_use_result`**22.9.3.9.35. `mysqli::options`, `mysqli_options`**

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::options`

`mysqli_options`

Set options

Description

Object oriented style

```
bool mysqli::options(
    int option,
    mixed value);
```

Procedural style

```
bool mysqli_options(
    mysqli link,
    int option,
    mixed value);
```

Used to set extra connect options and affect behavior for a connection.

This function may be called multiple times to set several options.

`mysqli_options` should be called after `mysqli_init` and before `mysqli_real_connect`.**Parameters***link*Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`*option*

The option that you want to set. It can be one of the following values:

Table 22.45. Valid options

Name	Description
<code>MYSQLI_OPT_CONNECT_TIMEOUT</code>	connection timeout in seconds (supported on Windows with TCP/IP since PHP 5.3.1)
<code>MYSQLI_OPT_LOCAL_INFILE</code>	enable/disable use of <code>LOAD LOCAL INFILE</code>
<code>MYSQLI_INIT_COMMAND</code>	command to execute after when connecting to MySQL server
<code>MYSQLI_READ_DEFAULT_FILE</code>	Read options from named option file instead of <code>my.cnf</code>
<code>MYSQLI_READ_DEFAULT_GROUP</code>	Read options from the named group from <code>my.cnf</code> or the file specified with <code>MYSQLI_READ_DEFAULT_FILE</code> .

Name	Description
<code>MYSQLI_SERVER_PUBLIC_KEY</code>	RSA public key file used with the SHA-256 based authentication.

value The value for the option.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Changelog

Version	Description
5.5.0	The <code>MYSQLI_SERVER_PUBLIC_KEY</code> option was added.

Examples

See `mysqli_real_connect`.

Notes



Note

MySQLnd always assumes the server default charset. This charset is sent during connection hand-shake/authentication, which mysqlnd will use.

Libmysql uses the default charset set in the `my.cnf` or by an explicit call to `mysqli_options` prior to calling `mysqli_real_connect`, but after `mysqli_init`.

See Also

`mysqli_init`
`mysqli_real_connect`

22.9.3.9.36. `mysqli::ping`, `mysqli_ping`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::ping`

`mysqli_ping`

Pings a server connection, or tries to reconnect if the connection has gone down

Description

Object oriented style

```
bool mysqli::ping();
```

Procedural style

```
bool mysqli_ping(
```

```
mysqli link);
```

Checks whether the connection to the server is working. If it has gone down, and global option [mysqli.reconnect](#) is enabled an automatic reconnection is attempted.

This function can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.134. [mysqli::ping](#) example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* check if server is alive */
if ($mysqli->ping()) {
    printf ("Our connection is ok!\n");
} else {
    printf ("Error: %s\n", $mysqli->error);
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* check if server is alive */
if (mysqli_ping($link)) {
    printf ("Our connection is ok!\n");
} else {
```

```
    printf ("Error: %s\n", mysqli_error($link));
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Our connection is ok!
```

22.9.3.9.37. [mysqli::poll](#), [mysqli_poll](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::poll](#)

[mysqli_poll](#)

Poll connections

Description

Object oriented style

```
public int mysqli::poll(
    array read,
    array error,
    array reject,
    int sec,
    int usec);
```

Procedural style

```
int mysqli_poll(
    array read,
    array error,
    array reject,
    int sec,
    int usec);
```



Warning

This function is currently not documented; only its argument list is available.

Poll connections. Available only with [mysqlnd](#). The method can be used as [static](#).

Parameters

read

error

reject

sec

Number of seconds to wait, must be non-negative.

[usec](#)

Number of microseconds to wait, must be non-negative.

Return Values

Returns number of ready connections upon success, [FALSE](#) otherwise.

Examples

Example 22.135. A [mysqli_poll](#) example

```
<?php
$link1 = mysqli_connect();
$link1->query("SELECT 'test'", MYSQLI_ASYNC);
$all_links = array($link1);
$processed = 0;
do {
    $links = $errors = $reject = array();
    foreach ($all_links as $link) {
        $links[] = $errors[] = $reject[] = $link;
    }
    if (!mysqli_poll($links, $errors, $reject, 1)) {
        continue;
    }
    foreach ($links as $link) {
        if ($result = $link->reap_async_query()) {
            print_r($result->fetch_row());
            if (is_object($result))
                mysqli_free_result($result);
        } else die(sprintf("MySQLi Error: %s", mysqli_error($link)));
        $processed++;
    }
} while ($processed < count($all_links));
?>
```

The above example will output:

```
Array
(
    [0] => test
)
```

See Also

[mysqli_query](#)[mysqli_reap_async_query](#)

22.9.3.9.38. [mysqli::prepare](#), [mysqli_prepare](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::prepare](#)

[mysqli_prepare](#)

Prepare an SQL statement for execution

Description

Object oriented style

```
mysqli_stmt mysqli::prepare(  
    string query);
```

Procedural style

```
mysqli_stmt mysqli_prepare(  
    mysqli link,  
    string query);
```

Prepares the SQL query, and returns a statement handle to be used for further operations on the statement. The query must consist of a single SQL statement.

The parameter markers must be bound to application variables using [mysqli_stmt_bind_param](#) and/or [mysqli_stmt_bind_result](#) before executing the statement or fetching rows.

Parameters

link

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

query

The query, as a string.



Note

You should not add a terminating semicolon or `\g` to the statement.

This parameter can include one or more parameter markers in the SQL statement by embedding question mark (?) characters at the appropriate positions.



Note

The markers are legal only in certain places in SQL statements. For example, they are allowed in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value.

However, they are not allowed for identifiers (such as table or column names), in the select list that names the columns to be returned by a `SELECT` statement, or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. It's not allowed to compare marker with `NULL` by `? IS NULL` too. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

Return Values

[mysqli_prepare](#) returns a statement object or `FALSE` if an error occurred.

Examples

Example 22.136. `mysqli::prepare` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$city = "Amersfoort";

/* create a prepared statement */
if ($stmt = $mysqli->prepare("SELECT District FROM City WHERE Name=?")) {

    /* bind parameters for markers */
    $stmt->bind_param("s", $city);

    /* execute query */
    $stmt->execute();

    /* bind result variables */
    $stmt->bind_result($district);

    /* fetch value */
    $stmt->fetch();

    printf("%s is in district %s\n", $city, $district);

    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$city = "Amersfoort";

/* create a prepared statement */
if ($stmt = mysqli_prepare($link, "SELECT District FROM City WHERE Name=?")) {

    /* bind parameters for markers */
    mysqli_stmt_bind_param($stmt, "s", $city);

    /* execute query */
```

```
mysqli_stmt_execute($stmt);

/* bind result variables */
mysqli_stmt_bind_result($stmt, $district);

/* fetch value */
mysqli_stmt_fetch($stmt);

printf("%s is in district %s\n", $city, $district);

/* close statement */
mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Amersfoort is in district Utrecht
```

See Also

[mysqli_stmt_execute](#)
[mysqli_stmt_fetch](#)
[mysqli_stmt_bind_param](#)
[mysqli_stmt_bind_result](#)
[mysqli_stmt_close](#)

22.9.3.9.39. [mysqli::query](#), [mysqli_query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::query](#)

[mysqli_query](#)

Performs a query on the database

Description

Object oriented style

```
mixed mysqli::query(
    string query,
    int resultmode
    = =MYSQLI_STORE_RESULT);
```

Procedural style

```
mixed mysqli_query(
    mysqli link,
    string query,
    int resultmode
    = =MYSQLI_STORE_RESULT);
```

Performs a [query](#) against the database.

Functionally, using this function is identical to calling `mysqli_real_query` followed either by `mysqli_use_result` or `mysqli_store_result`.



Note

In the case where you pass a statement to `mysqli_query` that is longer than `max_allowed_packet` of the server, the returned error codes are different depending on whether you are using MySQL Native Driver (`mysqlnd`) or MySQL Client Library (`libmysql`). The behavior is as follows:

- `mysqlnd` on Linux returns an error code of 1153. The error message means “got a packet bigger than `max_allowed_packet` bytes”.
- `mysqlnd` on Windows returns an error code 2006. This error message means “server has gone away”.
- `libmysql` on all platforms returns an error code 2006. This error message means “server has gone away”.

Parameters

- link* Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
- query* The query string.
Data inside the query should be [properly escaped](#).
- resultmode* Either the constant `MYSQLI_USE_RESULT` or `MYSQLI_STORE_RESULT` depending on the desired behavior. By default, `MYSQLI_STORE_RESULT` is used.

If you use `MYSQLI_USE_RESULT` all subsequent calls will return error `Commands out of sync` unless you call `mysqli_free_result`

With `MYSQLI_ASYNC` (available with `mysqlnd`), it is possible to perform query asynchronously. `mysqli_poll` is then used to get results from such queries.

Return Values

Returns `FALSE` on failure. For successful `SELECT`, `SHOW`, `DESCRIBE` or `EXPLAIN` queries `mysqli_query` will return a `mysqli_result` object. For other successful queries `mysqli_query` will return `TRUE`.

Changelog

Version	Description
5.3.0	Added the ability of async queries.

Examples

Example 22.137. `mysqli::query` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}

/* Create table doesn't return a resultset */
if ($mysqli->query("CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
    printf("Table myCity successfully created.\n");
}

/* Select queries return a resultset */
if ($result = $mysqli->query("SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", $result->num_rows);

    /* free result set */
    $result->close();
}

/* If we have to retrieve large amount of data we use MYSQLI_USE_RESULT */
if ($result = $mysqli->query("SELECT * FROM City", MYSQLI_USE_RESULT)) {

    /* Note, that we can't execute any functions which interact with the
       server until result set was closed. All calls will return an
       'out of sync' error */
    if (!$mysqli->query("SET @a:='this will not work'")) {
        printf("Error: %s\n", $mysqli->error);
    }
    $result->close();
}

$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Create table doesn't return a resultset */
if (mysqli_query($link, "CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
    printf("Table myCity successfully created.\n");
}

/* Select queries return a resultset */
if ($result = mysqli_query($link, "SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", mysqli_num_rows($result));

    /* free result set */
    mysqli_free_result($result);
}

/* If we have to retrieve large amount of data we use MYSQLI_USE_RESULT */
if ($result = mysqli_query($link, "SELECT * FROM City", MYSQLI_USE_RESULT)) {
```

```
/* Note, that we can't execute any functions which interact with the
server until result set was closed. All calls will return an
'out of sync' error */
if (!mysqli_query($link, "SET @a:='this will not work'")) {
    printf("Error: %s\n", mysqli_error($link));
}
mysqli_free_result($result);
}

mysqli_close($link);
?>
```

The above examples will output:

```
Table myCity successfully created.
Select returned 10 rows.
Error: Commands out of sync; You can't run this command now
```

See Also

[mysqli_real_query](#)
[mysqli_multi_query](#)
[mysqli_free_result](#)

22.9.3.9.40. [mysqli::real_connect](#), [mysqli_real_connect](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::real_connect](#)

[mysqli_real_connect](#)

Opens a connection to a mysql server

Description

Object oriented style

```
bool mysqli::real_connect(
    string host,
    string username,
    string passwd,
    string dbname,
    int port,
    string socket,
    int flags);
```

Procedural style

```
bool mysqli_real_connect(
    mysqli link,
    string host,
    string username,
    string passwd,
    string dbname,
    int port,
    string socket,
```

```
int flags);
```

Establish a connection to a MySQL database engine.

This function differs from `mysqli_connect`:

- `mysqli_real_connect` needs a valid object which has to be created by function `mysqli_init`.
- With the `mysqli_options` function you can set various options for connection.
- There is a `flags` parameter.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by <code>mysqli_connect</code> or <code>mysqli_init</code>
<i>host</i>	Can be either a host name or an IP address. Passing the <code>NULL</code> value or the string "localhost" to this parameter, the local host is assumed. When possible, pipes will be used instead of the TCP/IP protocol.
<i>username</i>	The MySQL user name.
<i>passwd</i>	If provided or <code>NULL</code> , the MySQL server will attempt to authenticate the user against those user records which have no password only. This allows one username to be used with different permissions (depending on if a password as provided or not).
<i>dbname</i>	If provided will specify the default database to be used when performing queries.
<i>port</i>	Specifies the port number to attempt to connect to the MySQL server.
<i>socket</i>	Specifies the socket or named pipe that should be used.



Note

Specifying the `socket` parameter will not explicitly determine the type of connection to be used when connecting to the MySQL server. How the connection is made to the MySQL database is determined by the `host` parameter.

flags With the parameter `flags` you can set different connection options:

Table 22.46. Supported flags

Name	Description
<code>MYSQLI_CLIENT_COMPRESS</code>	Use compression protocol
<code>MYSQLI_CLIENT_FOUND_ROWS</code>	return number of matched rows, not the number of affected rows
<code>MYSQLI_CLIENT_IGNORE_SPACE</code>	Allow spaces after function names. Makes all function names reserved words.
<code>MYSQLI_CLIENT_INTERACTIVE</code>	Allow <code>interactive_timeout</code> seconds (instead of

Name	Description
	<code>wait_timeout</code> seconds) of inactivity before closing the connection
<code>MYSQLI_CLIENT_SSL</code>	Use SSL (encryption)

**Note**

For security reasons the `MULTI_STATEMENT` flag is not supported in PHP. If you want to execute multiple queries use the `mysqli_multi_query` function.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples**Example 22.138. `mysqli::real_connect` example**

Object oriented style

```
<?php
$mysqli = mysqli_init();
if (!$mysqli) {
    die('mysqli_init failed');
}

if (!$mysqli->options(MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
    die('Setting MYSQLI_INIT_COMMAND failed');
}

if (!$mysqli->options(MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
    die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
}

if (!$mysqli->real_connect('localhost', 'my_user', 'my_password', 'my_db')) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}

echo 'Success... ' . $mysqli->host_info . "\n";

$mysqli->close();
?>
```

Object oriented style when extending mysqli class

```
<?php
class foo_mysqli extends mysqli {
    public function __construct($host, $user, $pass, $db) {
        parent::init();

        if (!$parent::options(MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
```

```
        die('Setting MYSQLI_INIT_COMMAND failed');
    }

    if (!parent::options(MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
        die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
    }

    if (!parent::real_connect($host, $user, $pass, $db)) {
        die('Connect Error (' . mysqli_connect_errno() . ') '
            . mysqli_connect_error());
    }
}

$db = new foo_mysqli('localhost', 'my_user', 'my_password', 'my_db');

echo 'Success... ' . $db->host_info . "\n";

$db->close();
?>
```

Procedural style

```
<?php

$link = mysqli_init();
if (!$link) {
    die('mysqli_init failed');
}

if (!mysqli_options($link, MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
    die('Setting MYSQLI_INIT_COMMAND failed');
}

if (!mysqli_options($link, MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
    die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
}

if (!mysqli_real_connect($link, 'localhost', 'my_user', 'my_password', 'my_db')) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}

echo 'Success... ' . mysqli_get_host_info($link) . "\n";

mysqli_close($link);
?>
```

The above examples will output:

```
Success... MySQL host info: localhost via TCP/IP
```

Notes



Note

MySQLnd always assumes the server default charset. This charset is sent during connection hand-shake/authentication, which mysqlnd will use.

Libmysql uses the default charset set in the `my.cnf` or by an explicit call to `mysqli_options` prior to calling `mysqli_real_connect`, but after `mysqli_init`.

See Also

`mysqli_connect`
`mysqli_init`
`mysqli_options`
`mysqli_ssl_set`
`mysqli_close`

22.9.3.9.41. `mysqli::real_escape_string`, `mysqli_real_escape_string`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::real_escape_string`

`mysqli_real_escape_string`

Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection

Description

Object oriented style

```
string mysqli::escape_string(  
    string escapestr);
```

```
string mysqli::real_escape_string(  
    string escapestr);
```

Procedural style

```
string mysqli_real_escape_string(  
    mysqli link,  
    string escapestr);
```

This function is used to create a legal SQL string that you can use in an SQL statement. The given string is encoded to an escaped SQL string, taking into account the current character set of the connection.



Security: the default character set

The character set must be set either at the server level, or with the API function `mysqli_set_charset` for it to affect `mysqli_real_escape_string`. See the concepts section on [character sets](#) for more information.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

escapestr

The string to be escaped.

Characters encoded are `NUL` (ASCII 0), `\n`, `\r`, `\`, `'`, `"`, and `Control-Z`.

Return Values

Returns an escaped string.

Examples

Example 22.139. `mysqli::real_escape_string` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$mysqli->query("CREATE TEMPORARY TABLE myCity LIKE City");

$city = "'s Hertogenbosch";

/* this query will fail, cause we didn't escape $city */
if (!$mysqli->query("INSERT into myCity (Name) VALUES ('$city')")) {
    printf("Error: %s\n", $mysqli->sqlstate);
}

$city = $mysqli->real_escape_string($city);

/* this query with escaped $city will work */
if ($mysqli->query("INSERT into myCity (Name) VALUES ('$city')")) {
    printf("%d Row inserted.\n", $mysqli->affected_rows);
}

$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

mysqli_query($link, "CREATE TEMPORARY TABLE myCity LIKE City");

$city = "'s Hertogenbosch";

/* this query will fail, cause we didn't escape $city */
if (!mysqli_query($link, "INSERT into myCity (Name) VALUES ('$city')")) {
    printf("Error: %s\n", mysqli_sqlstate($link));
}

$city = mysqli_real_escape_string($link, $city);

/* this query with escaped $city will work */
```

```
if (mysqli_query($link, "INSERT into myCity (Name) VALUES ('$city')")) {
    printf("%d Row inserted.\n", mysqli_affected_rows($link));
}

mysqli_close($link);
?>
```

The above examples will output:

```
Error: 42000
1 Row inserted.
```

Notes



Note

For those accustomed to using `mysql_real_escape_string`, note that the arguments of `mysqli_real_escape_string` differ from what `mysql_real_escape_string` expects. The `link` identifier comes first in `mysqli_real_escape_string`, whereas the string to be escaped comes first in `mysql_real_escape_string`.

See Also

`mysqli_set_charset`
`mysqli_character_set_name`

22.9.3.9.42. `mysqli::real_query`, `mysqli_real_query`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::real_query`

`mysqli_real_query`

Execute an SQL query

Description

Object oriented style

```
bool mysqli::real_query(
    string query);
```

Procedural style

```
bool mysqli_real_query(
    mysqli link,
    string query);
```

Executes a single query against the database whose result can then be retrieved or stored using the `mysqli_store_result` or `mysqli_use_result` functions.

In order to determine if a given query should return a result set or not, see `mysqli_field_count`.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
<i>query</i>	The query, as a string. Data inside the query should be properly escaped .

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

See Also

[mysqli_query](#)
[mysqli_store_result](#)
[mysqli_use_result](#)

22.9.3.9.43. [mysqli::reap_async_query](#), [mysqli_reap_async_query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::reap_async_query](#)

[mysqli_reap_async_query](#)

Get result from async query

Description

Object oriented style

```
public mysqli_result mysqli::reap_async_query();
```

Procedural style

```
mysqli_result mysqli_reap_async_query(  
    mysql link);
```



Warning

This function is currently not documented; only its argument list is available.

Get result from async query. Available only with [mysqlnd](#).

Parameters

<i>link</i>	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
-------------	--

Return Values

Returns [mysqli_result](#) in success, [FALSE](#) otherwise.

See Also

`mysqli_poll`

22.9.3.9.44. `mysqli::refresh`, `mysqli_refresh`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::refresh`

`mysqli_refresh`

Refreshes

Description

Object oriented style

```
public bool mysqli::refresh(
    int options);
```

Procedural style

```
int mysqli_refresh(
    resource link,
    int options);
```

Flushes tables or caches, or resets the replication server information.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

options

The options to refresh, using the `MYSQLI_REFRESH_*` constants as documented within the [MySQLi constants](#) documentation.

See also the official [MySQL Refresh](#) documentation.

Return Values

`TRUE` if the refresh was a success, otherwise `FALSE`

See Also

`mysqli_poll`

22.9.3.9.45. `mysqli::rollback`, `mysqli_rollback`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::rollback`

`mysqli_rollback`

Rolls back current transaction

Description

Object oriented style

```
bool mysqli::rollback();
```

Procedural style

```
bool mysqli_rollback(  
    mysqli link);
```

Rollbacks the current transaction for the database.

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.140. [mysqli::rollback](#) example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
  
/* disable autocommit */  
$mysqli->autocommit(FALSE);  
  
$mysqli->query("CREATE TABLE myCity LIKE City");  
$mysqli->query("ALTER TABLE myCity Type=InnoDB");  
$mysqli->query("INSERT INTO myCity SELECT * FROM City LIMIT 50");  
  
/* commit insert */  
$mysqli->commit();  
  
/* delete all rows */  
$mysqli->query("DELETE FROM myCity");  
  
if ($result = $mysqli->query("SELECT COUNT(*) FROM myCity")) {  
    $row = $result->fetch_row();  
    printf("%d rows in table myCity.\n", $row[0]);  
    /* Free result */  
    $result->close();  
}  
  
/* Rollback */  
$mysqli->rollback();  
  
if ($result = $mysqli->query("SELECT COUNT(*) FROM myCity")) {  
    $row = $result->fetch_row();  
    printf("%d rows in table myCity (after rollback).\n", $row[0]);  
    /* Free result */  
    $result->close();
```

```

}

/* Drop table myCity */
$mysqli->query("DROP TABLE myCity");

$mysqli->close();
?>

```

Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* disable autocommit */
mysqli_autocommit($link, FALSE);

mysqli_query($link, "CREATE TABLE myCity LIKE City");
mysqli_query($link, "ALTER TABLE myCity Type=InnoDB");
mysqli_query($link, "INSERT INTO myCity SELECT * FROM City LIMIT 50");

/* commit insert */
mysqli_commit($link);

/* delete all rows */
mysqli_query($link, "DELETE FROM myCity");

if ($result = mysqli_query($link, "SELECT COUNT(*) FROM myCity")) {
    $row = mysqli_fetch_row($result);
    printf("%d rows in table myCity.\n", $row[0]);
    /* Free result */
    mysqli_free_result($result);
}

/* Rollback */
mysqli_rollback($link);

if ($result = mysqli_query($link, "SELECT COUNT(*) FROM myCity")) {
    $row = mysqli_fetch_row($result);
    printf("%d rows in table myCity (after rollback).\n", $row[0]);
    /* Free result */
    mysqli_free_result($result);
}

/* Drop table myCity */
mysqli_query($link, "DROP TABLE myCity");

mysqli_close($link);
?>

```

The above examples will output:

```

0 rows in table myCity.
50 rows in table myCity (after rollback).

```

See Also

```
mysqli_commit  
mysqli_autocommit
```

22.9.3.9.46. `mysqli::rpl_query_type`, `mysqli_rpl_query_type`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::rpl_query_type`

```
mysqli_rpl_query_type
```

Returns RPL query type

Description

Object oriented style

```
int mysqli::rpl_query_type(  
    string query);
```

Procedural style

```
int mysqli_rpl_query_type(  
    mysqli link,  
    string query);
```

Returns `MYSQLI_RPL_MASTER`, `MYSQLI_RPL_SLAVE` or `MYSQLI_RPL_ADMIN` depending on a query type. `INSERT`, `UPDATE` and similar are *master* queries, `SELECT` is *slave*, and `FLUSH`, `REPAIR` and similar are *admin*.

**Warning**

This function is currently not documented; only its argument list is available.

**Warning**

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.9.47. `mysqli::select_db`, `mysqli_select_db`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::select_db`

```
mysqli_select_db
```

Selects the default database for database queries

Description

Object oriented style

```
bool mysqli::select_db(  
    string dbname);
```

Procedural style

```
bool mysqli_select_db(
    mysqli link,
    string dbname);
```

Selects the default database to be used when performing queries against the database connection.



Note

This function should only be used to change the default database for the connection. You can select the default database with 4th parameter in [mysqli_connect](#).

Parameters

<i>link</i>	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
<i>dbname</i>	The database name.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.141. [mysqli::select_db](#) example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* return name of current default database */
if ($result = $mysqli->query("SELECT DATABASE()")) {
    $row = $result->fetch_row();
    printf("Default database is %s.\n", $row[0]);
    $result->close();
}

/* change db to world db */
$mysqli->select_db("world");

/* return name of current default database */
if ($result = $mysqli->query("SELECT DATABASE()")) {
    $row = $result->fetch_row();
    printf("Default database is %s.\n", $row[0]);
    $result->close();
}

$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* return name of current default database */
if ($result = mysqli_query($link, "SELECT DATABASE())) {
    $row = mysqli_fetch_row($result);
    printf("Default database is %s.\n", $row[0]);
    mysqli_free_result($result);
}

/* change db to world db */
mysqli_select_db($link, "world");

/* return name of current default database */
if ($result = mysqli_query($link, "SELECT DATABASE())) {
    $row = mysqli_fetch_row($result);
    printf("Default database is %s.\n", $row[0]);
    mysqli_free_result($result);
}

mysqli_close($link);
?>
```

The above examples will output:

```
Default database is test.
Default database is world.
```

See Also

[mysqli_connect](#)
[mysqli_real_connect](#)

22.9.3.9.48. [mysqli::send_query](#), [mysqli_send_query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::send_query](#)

[mysqli_send_query](#)

Send the query and return

Description

Object oriented style

```
bool mysqli::send_query(
```

```
string query);
```

Procedural style

```
bool mysqli_send_query(
    mysqli link,
    string query);
```



Warning

This function is currently not documented; only its argument list is available.



Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.9.49. *mysqli::set_charset*, *mysqli_set_charset*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysqli::set_charset*

mysqli_set_charset

Sets the default client character set

Description

Object oriented style

```
bool mysqli::set_charset(
    string charset);
```

Procedural style

```
bool mysqli_set_charset(
    mysqli link,
    string charset);
```

Sets the default character set to be used when sending data from and to the database server.

Parameters

link Procedural style only: A link identifier returned by *mysqli_connect* or *mysqli_init*

charset The charset to be set as default.

Return Values

Returns *TRUE* on success or *FALSE* on failure.

Notes



Note

To use this function on a Windows platform you need MySQL client library version 4.1.11 or above (for MySQL 5.0 you need 5.0.6 or above).

**Note**

This is the preferred way to change the charset. Using `mysqli_query` to set it (such as `SET NAMES utf8`) is not recommended. See the [MySQL character set concepts](#) section for more information.

Examples**Example 22.142. `mysqli::set_charset` example**

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* change character set to utf8 */
if (!$mysqli->set_charset("utf8")) {
    printf("Error loading character set utf8: %s\n", $mysqli->error);
} else {
    printf("Current character set: %s\n", $mysqli->character_set_name());
}

$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'test');

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* change character set to utf8 */
if (!mysqli_set_charset($link, "utf8")) {
    printf("Error loading character set utf8: %s\n", mysqli_error($link));
} else {
    printf("Current character set: %s\n", mysqli_character_set_name($link));
}

mysqli_close($link);
?>
```

The above examples will output:

```
Current character set: utf8
```

See Also

`mysqli_character_set_name`
`mysqli_real_escape_string`
[List of character sets that MySQL supports](#)

22.9.3.9.50. `mysqli::set_local_infile_default`, `mysqli_set_local_infile_default`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::set_local_infile_default`

`mysqli_set_local_infile_default`

Unsets user defined handler for load local infile command

Description

```
void mysqli_set_local_infile_default(
    mysqli link);
```

Deactivates a `LOAD DATA INFILE LOCAL` handler previously set with `mysqli_set_local_infile_handler`.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

No value is returned.

Examples

See `mysqli_set_local_infile_handler` examples

See Also

`mysqli_set_local_infile_handler`

22.9.3.9.51. `mysqli::set_local_infile_handler`, `mysqli_set_local_infile_handler`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::set_local_infile_handler`

`mysqli_set_local_infile_handler`

Set callback function for `LOAD DATA LOCAL INFILE` command

Description

Object oriented style

```
bool mysqli::set_local_infile_handler(
    mysqli link,
```

```
callable read_func);
```

Procedural style

```
bool mysqli_set_local_infile_handler(
    mysqli link,
    callable read_func);
```

Set callback function for LOAD DATA LOCAL INFILE command

The callback's task is to read input from the file specified in the `LOAD DATA LOCAL INFILE` and to reformat it into the format understood by `LOAD DATA INFILE`.

The returned data needs to match the format specified in the `LOAD DATA`

Parameters

<i>link</i>	Procedural style only: A link identifier returned by <code>mysqli_connect</code> or <code>mysqli_init</code>
<i>read_func</i>	A callback function or object method taking the following parameters:
<i>stream</i>	A PHP stream associated with the SQL commands INFILE
<i>&buffer</i>	A string buffer to store the rewritten input into
<i>buflen</i>	The maximum number of characters to be stored in the buffer
<i>&errmsg</i>	If an error occurs you can store an error message in here

The callback function should return the number of characters stored in the *buffer* or a negative value if an error occurred.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.143. `mysqli::set_local_infile_handler` example

Object oriented style

```
<?php
$db = mysqli_init();
$db->real_connect("localhost","root","","test");

function callme($stream, &$buffer, $buflen, &$errmsg)
{
    $buffer = fgets($stream);

    echo $buffer;

    // convert to upper case and replace "," delimiter with [TAB]
```

```
$buffer = strtoupper(str_replace(",","\\t", $buffer));

return strlen($buffer);
}

echo "Input:\\n";

$db->set_local_infile_handler("callme");
$db->query("LOAD DATA LOCAL INFILE 'input.txt' INTO TABLE t1");
$db->set_local_infile_default();

$res = $db->query("SELECT * FROM t1");

echo "\\nResult:\\n";
while ($row = $res->fetch_assoc()) {
    echo join(",",$row)."\\n";
}
?>
```

Procedural style

```
<?php
$db = mysqli_init();
mysqli_real_connect($db, "localhost","root","","test");

function callme($stream, &$buffer, $buflen, &$errormsg)
{
    $buffer = fgets($stream);

    echo $buffer;

    // convert to upper case and replace "," delimiter with [TAB]
    $buffer = strtoupper(str_replace(",","\\t", $buffer));

    return strlen($buffer);
}

echo "Input:\\n";

mysqli_set_local_infile_handler($db, "callme");
mysqli_query($db, "LOAD DATA LOCAL INFILE 'input.txt' INTO TABLE t1");
mysqli_set_local_infile_default($db);

$res = mysqli_query($db, "SELECT * FROM t1");

echo "\\nResult:\\n";
while ($row = mysqli_fetch_assoc($res)) {
    echo join(",",$row)."\\n";
}
?>
```

The above examples will output:

```
Input:
23,foo
42,bar

Output:
```

```
23,FOO
42,BAR
```

See Also

[mysqli_set_local_infile_default](#)

22.9.3.9.52. [mysqli::\\$sqlstate](#), [mysqli_sqlstate](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$sqlstate](#)

[mysqli_sqlstate](#)

Returns the SQLSTATE error from previous MySQL operation

Description

Object oriented style

```
string
mysqli->sqlstate ;
```

Procedural style

```
string mysqli_sqlstate(
mysqli link);
```

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. '00000' means no error. The values are specified by ANSI SQL and ODBC. For a list of possible values, see <http://dev.mysql.com/doc/mysql/en/error-handling.html>.



Note

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value [HY000](#) (general error) is used for unmapped errors.

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. '00000' means no error.

Examples

Example 22.144. [\\$mysqli->sqlstate](#) example

Object oriented style

```
<?php
```

```
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Table City already exists, so we should get an error */
if (!$mysqli->query("CREATE TABLE City (ID INT, Name VARCHAR(30))")) {
    printf("Error - SQLSTATE %s.\n", $mysqli->sqlstate);
}

$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Table City already exists, so we should get an error */
if (!mysqli_query($link, "CREATE TABLE City (ID INT, Name VARCHAR(30))")) {
    printf("Error - SQLSTATE %s.\n", mysqli_sqlstate($link));
}

mysqli_close($link);
?>
```

The above examples will output:

```
Error - SQLSTATE 42S01.
```

See Also

[mysqli_errno](#)
[mysqli_error](#)

22.9.3.9.53. [mysqli::ssl_set](#), [mysqli_ssl_set](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::ssl_set](#)

[mysqli_ssl_set](#)

Used for establishing secure connections using SSL

Description

Object oriented style

```
bool mysqli::ssl_set(  
    string key,  
    string cert,  
    string ca,  
    string capath,  
    string cipher);
```

Procedural style

```
bool mysqli_ssl_set(  
    mysqli link,  
    string key,  
    string cert,  
    string ca,  
    string capath,  
    string cipher);
```

Used for establishing secure connections using SSL. It must be called before [mysqli_real_connect](#). This function does nothing unless OpenSSL support is enabled.

Note that MySQL Native Driver does not support SSL before PHP 5.3.3, so calling this function when using MySQL Native Driver will result in an error. MySQL Native Driver is enabled by default on Microsoft Windows from PHP version 5.3 onwards.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
<i>key</i>	The path name to the key file.
<i>cert</i>	The path name to the certificate file.
<i>ca</i>	The path name to the certificate authority file.
<i>capath</i>	The pathname to a directory that contains trusted SSL CA certificates in PEM format.
<i>cipher</i>	A list of allowable ciphers to use for SSL encryption.

Any unused SSL parameters may be given as [NULL](#)

Return Values

This function always returns [TRUE](#) value. If SSL setup is incorrect [mysqli_real_connect](#) will return an error when you attempt to connect.

See Also

[mysqli_options](#)
[mysqli_real_connect](#)

22.9.3.9.54. [mysqli::stat](#), [mysqli_stat](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::stat](#)

`mysqli_stat`

Gets the current system status

Description

Object oriented style

```
string mysqli::stat();
```

Procedural style

```
string mysqli_stat(  
    mysqli link);
```

`mysqli_stat` returns a string containing information similar to that provided by the 'mysqladmin status' command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

A string describing the server status. `FALSE` if an error occurred.

Examples

Example 22.145. `mysqli::stat` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
  
printf ("System status: %s\n", $mysqli->stat());  
  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {
```

```
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

printf("System status: %s\n", mysqli_stat($link));

mysqli_close($link);
?>
```

The above examples will output:

```
System status: Uptime: 272  Threads: 1  Questions: 5340  Slow queries: 0
Opens: 13  Flush tables: 1  Open tables: 0  Queries per second avg: 19.632
Memory in use: 8496K  Max memory used: 8560K
```

See Also

[mysqli_get_server_info](#)

22.9.3.9.55. [mysqli::stmt_init](#), [mysqli_stmt_init](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::stmt_init](#)

[mysqli_stmt_init](#)

Initializes a statement and returns an object for use with [mysqli_stmt_prepare](#)

Description

Object oriented style

```
mysqli_stmt mysqli::stmt_init();
```

Procedural style

```
mysqli_stmt mysqli_stmt_init(
    mysqli link);
```

Allocates and initializes a statement object suitable for [mysqli_stmt_prepare](#).



Note

Any subsequent calls to any [mysqli_stmt](#) function will fail until [mysqli_stmt_prepare](#) was called.

Parameters

link

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns an object.

See Also

[mysqli_stmt_prepare](#)

22.9.3.9.56. [mysqli::store_result](#), [mysqli_store_result](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::store_result](#)

[mysqli_store_result](#)

Transfers a result set from the last query

Description

Object oriented style

```
mysqli_result mysqli::store_result();
```

Procedural style

```
mysqli_result mysqli_store_result(  
    mysqli link);
```

Transfers the result set from the last query on the database connection represented by the [link](#) parameter to be used with the [mysqli_data_seek](#) function.

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns a buffered result object or [FALSE](#) if an error occurred.



Note

[mysqli_store_result](#) returns [FALSE](#) in case the query didn't return a result set (if the query was, for example an INSERT statement). This function also returns [FALSE](#) if the reading of the result set failed. You can check if you have got an error by checking if [mysqli_error](#) doesn't return an empty string, if [mysqli_errno](#) returns a non zero value, or if [mysqli_field_count](#) returns a non zero value. Also possible reason for this function returning [FALSE](#) after successful call to [mysqli_query](#) can be too large result set (memory for it cannot be allocated). If [mysqli_field_count](#) returns a non-zero value, the statement should have produced a non-empty result set.

Notes



Note

Although it is always good practice to free the memory used by the result of a query using the [mysqli_free_result](#) function, when transferring large result sets using the [mysqli_store_result](#) this becomes particularly important.

Examples

See `mysqli_multi_query`.

See Also

`mysqli_real_query`
`mysqli_use_result`

22.9.3.9.57. `mysqli::$thread_id`, `mysqli_thread_id`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::$thread_id`

`mysqli_thread_id`

Returns the thread ID for the current connection

Description

Object oriented style

```
int  
mysqli->thread_id ;
```

Procedural style

```
int mysqli_thread_id(  
mysqli link);
```

The `mysqli_thread_id` function returns the thread ID for the current connection which can then be killed using the `mysqli_kill` function. If the connection is lost and you reconnect with `mysqli_ping`, the thread ID will be other. Therefore you should get the thread ID only when you need it.



Note

The thread ID is assigned on a connection-by-connection basis. Hence, if the connection is broken and then re-established a new thread ID will be assigned.

To kill a running query you can use the SQL command `KILL QUERY processid`.

Parameters

`link` Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns the Thread ID for the current connection.

Examples

Example 22.146. `$mysqli->thread_id` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* determine our thread id */
$thread_id = $mysqli->thread_id;

/* Kill connection */
$mysqli->kill($thread_id);

/* This should produce an error */
if (!$mysqli->query("CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", $mysqli->error);
    exit;
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* determine our thread id */
$thread_id = mysqli_thread_id($link);

/* Kill connection */
mysqli_kill($link, $thread_id);

/* This should produce an error */
if (!mysqli_query($link, "CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", mysqli_error($link));
    exit;
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: MySQL server has gone away
```

See Also

`mysqli_kill`

22.9.3.9.58. [mysqli::thread_safe](#), [mysqli_thread_safe](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::thread_safe`

`mysqli_thread_safe`

Returns whether thread safety is given or not

Description

Procedural style

```
bool mysqli_thread_safe();
```

Tells whether the client library is compiled as thread-safe.

Return Values

`TRUE` if the client library is thread-safe, otherwise `FALSE`.

22.9.3.9.59. [mysqli::use_result](#), [mysqli_use_result](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::use_result`

`mysqli_use_result`

Initiate a result set retrieval

Description

Object oriented style

```
mysqli_result mysqli::use_result();
```

Procedural style

```
mysqli_result mysqli_use_result(  
    mysqli link);
```

Used to initiate the retrieval of a result set from the last query executed using the `mysqli_real_query` function on the database connection.

Either this or the `mysqli_store_result` function must be called before the results of a query can be retrieved, and one or the other must be called to prevent the next query on that database connection from failing.



Note

The `mysqli_use_result` function does not transfer the entire result set from the database and hence cannot be used functions such as `mysqli_data_seek` to move to a particular row within the set. To use this functionality, the result

set must be stored using `mysqli_store_result`. One should not use `mysqli_use_result` if a lot of processing on the client side is performed, since this will tie up the server and prevent other threads from updating any tables from which the data is being fetched.

Return Values

Returns an unbuffered result object or `FALSE` if an error occurred.

Examples

Example 22.147. `mysqli::use_result` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT CURRENT_USER()";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";

/* execute multi query */
if ($mysqli->multi_query($query)) {
    do {
        /* store first result set */
        if ($result = $mysqli->use_result()) {
            while ($row = $result->fetch_row()) {
                printf("%s\n", $row[0]);
            }
            $result->close();
        }
        /* print divider */
        if ($mysqli->more_results()) {
            printf("-----\n");
        }
    } while ($mysqli->next_result());
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT CURRENT_USER()";
```

```
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";

/* execute multi query */
if (mysqli_multi_query($link, $query)) {
    do {
        /* store first result set */
        if ($result = mysqli_use_result($link)) {
            while ($row = mysqli_fetch_row($result)) {
                printf("%s\n", $row[0]);
            }
            mysqli_free_result($result);
        }
        /* print divider */
        if (mysqli_more_results($link)) {
            printf("-----\n");
        }
    } while (mysqli_next_result($link));
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
my_user@localhost
-----
Amersfoort
Maastricht
Dordrecht
Leiden
Haarlemmermeer
```

See Also

[mysqli_real_query](#)
[mysqli_store_result](#)

22.9.3.9.60. [mysqli::\\$warning_count](#), [mysqli_warning_count](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli::\\$warning_count](#)

[mysqli_warning_count](#)

Returns the number of warnings from the last query for the given link

Description

Object oriented style

```
int
mysqli->warning_count ;
```

Procedural style

```
int mysqli_warning_count(
```

```
mysqli link);
```

Returns the number of warnings from the last query in the connection.



Note

For retrieving warning messages you can use the SQL command `SHOW WARNINGS [limit row_count]`.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Number of warnings or zero if there are no warnings.

Examples

Example 22.148. `$mysqli->warning_count` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$mysqli->query("CREATE TABLE myCity LIKE City");

/* a remarkable city in Wales */
$query = "INSERT INTO myCity (CountryCode, Name) VALUES ('GBR',
    'Llanfairpwllgwyngyllgogerychwyrndrobwlllandysiliogogoch')";

$mysqli->query($query);

if ($mysqli->warning_count) {
    if ($result = $mysqli->query("SHOW WARNINGS")) {
        $row = $result->fetch_row();
        printf("%s (%d): %s\n", $row[0], $row[1], $row[2]);
        $result->close();
    }
}

/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

mysqli_query($link, "CREATE TABLE myCity LIKE City");

/* a remarkable long city name in Wales */
$query = "INSERT INTO myCity (CountryCode, Name) VALUES('GBR',
    'Llanfairpwllgwyngyllgogerychwyrndrobwlllllantysiliogogoch')";

mysqli_query($link, $query);

if (mysqli_warning_count($link)) {
    if ($result = mysqli_query($link, "SHOW WARNINGS")) {
        $row = mysqli_fetch_row($result);
        printf("%s (%d): %s\n", $row[0], $row[1], $row[2]);
        mysqli_free_result($result);
    }
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Warning (1264): Data truncated for column 'Name' at row 1
```

See Also

[mysqli_errno](#)
[mysqli_error](#)
[mysqli_sqlstate](#)

22.9.3.10. The `mysqli_stmt` class ([mysqli_stmt](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

Represents a prepared statement.

```
mysqli_stmt {
    mysqli_stmt

    Properties

    int
        mysqli_stmt->affected_rows ;

    int
        mysqli_stmt->errno ;

    array
        mysqli_stmt->error_list ;

    string
```

```
mysqli_stmt->error ;

int
mysqli_stmt->field_count ;

int
mysqli_stmt->insert_id ;

int
mysqli_stmt->num_rows ;

int
mysqli_stmt->param_count ;

string
mysqli_stmt->sqlstate ;

Methods

int mysqli_stmt::attr_get(
    int attr);

bool mysqli_stmt::attr_set(
    int attr,
    int mode);

bool mysqli_stmt::bind_param(
    string types,
    mixed var1,
    mixed ...);

bool mysqli_stmt::bind_result(
    mixed var1,
    mixed ...);

bool mysqli_stmt::close();

void mysqli_stmt::data_seek(
    int offset);

bool mysqli_stmt::execute();

bool mysqli_stmt::fetch();

void mysqli_stmt::free_result();

mysqli_result mysqli_stmt::get_result();

object mysqli_stmt::get_warnings(
    mysqli_stmt stmt);

mixed mysqli_stmt::prepare(
    string query);

bool mysqli_stmt::reset();

mysqli_result mysqli_stmt::result_metadata();

bool mysqli_stmt::send_long_data(
    int param_nr,
    string data);

bool mysqli_stmt::store_result();
}
```

22.9.3.10.1. `mysqli_stmt::$affected_rows`, `mysqli_stmt_affected_rows`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::$affected_rows`

`mysqli_stmt_affected_rows`

Returns the total number of rows changed, deleted, or inserted by the last executed statement

Description

Object oriented style

```
int
mysqli_stmt->affected_rows ;
```

Procedural style

```
int mysqli_stmt_affected_rows(
mysqli_stmt stmt);
```

Returns the number of rows affected by `INSERT`, `UPDATE`, or `DELETE` query.This function only works with queries which update a table. In order to get the number of rows from a `SELECT` query, use `mysqli_stmt_num_rows` instead.**Parameters**

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an `UPDATE/DELETE` statement, no rows matched the `WHERE` clause in the query or that no query has yet been executed. -1 indicates that the query has returned an error. `NULL` indicates an invalid argument was supplied to the function.

**Note**

If the number of affected rows is greater than maximal PHP int value, the number of affected rows will be returned as a string value.

Examples**Example 22.149. Object oriented style**

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* create temp table */
```

```

$mysqli->query("CREATE TEMPORARY TABLE myCountry LIKE Country");

$query = "INSERT INTO myCountry SELECT * FROM Country WHERE Code LIKE ?";

/* prepare statement */
if ($stmt = $mysqli->prepare($query)) {

    /* Bind variable for placeholder */
    $code = 'A%';
    $stmt->bind_param("s", $code);

    /* execute statement */
    $stmt->execute();

    printf("rows inserted: %d\n", $stmt->affected_rows);

    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
?>

```

Example 22.150. Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* create temp table */
mysqli_query($link, "CREATE TEMPORARY TABLE myCountry LIKE Country");

$query = "INSERT INTO myCountry SELECT * FROM Country WHERE Code LIKE ?";

/* prepare statement */
if ($stmt = mysqli_prepare($link, $query)) {

    /* Bind variable for placeholder */
    $code = 'A%';
    mysqli_stmt_bind_param($stmt, "s", $code);

    /* execute statement */
    mysqli_stmt_execute($stmt);

    printf("rows inserted: %d\n", mysqli_stmt_affected_rows($stmt));

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```
rows inserted: 17
```

See Also

`mysqli_stmt_num_rows`
`mysqli_prepare`

22.9.3.10.2. `mysqli_stmt::attr_get`, `mysqli_stmt_attr_get`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::attr_get`

`mysqli_stmt_attr_get`

Used to get the current value of a statement attribute

Description

Object oriented style

```
int mysqli_stmt::attr_get(  
    int attr);
```

Procedural style

```
int mysqli_stmt_attr_get(  
    mysqli_stmt stmt,  
    int attr);
```

Gets the current value of a statement attribute.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

attr The attribute that you want to get.

Return Values

Returns `FALSE` if the attribute is not found, otherwise returns the value of the attribute.

22.9.3.10.3. `mysqli_stmt::attr_set`, `mysqli_stmt_attr_set`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::attr_set`

`mysqli_stmt_attr_set`

Used to modify the behavior of a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::attr_set(
    int attr,
    int mode);
```

Procedural style

```
bool mysqli_stmt_attr_set(
    mysqli_stmt stmt,
    int attr,
    int mode);
```

Used to modify the behavior of a prepared statement. This function may be called multiple times to set several attributes.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

attr

The attribute that you want to set. It can have one of the following values:

Table 22.47. Attribute values

Character	Description
<code>MYSQLI_STMT_ATTR_UPDATE_MAX_LENGTH</code>	Maximum value of <code>MYSQL_FIELD->max_length</code> to update the metadata.
<code>MYSQLI_STMT_ATTR_CURSOR_TYPE</code>	TYPE of cursor to open for statement when <code>mysqli_stmt_execute</code> is invoked. <i>mode</i> can be <code>MYSQLI_CURSOR_TYPE_NO_CURSOR</code> (the default) or <code>MYSQLI_CURSOR_TYPE_READ_ONLY</code> .
<code>MYSQLI_STMT_ATTR_PREFETCH_ROWS</code>	Number of rows to fetch from server at a time when using a cursor. <i>mode</i> can be in the range from 1 to the maximum value of unsigned long. The default is 1.

If you use the `MYSQLI_STMT_ATTR_CURSOR_TYPE` option with `MYSQLI_CURSOR_TYPE_READ_ONLY`, a cursor is opened for the statement when you invoke `mysqli_stmt_execute`. If there is already an open cursor from a previous `mysqli_stmt_execute` call, it closes the cursor before opening a new one. `mysqli_stmt_reset` also closes any open cursor before preparing the statement for re-execution. `mysqli_stmt_free_result` closes any open cursor.

If you open a cursor for a prepared statement, `mysqli_stmt_store_result` is unnecessary.

mode

The value to assign to the attribute.

22.9.3.10.4. `mysqli_stmt::bind_param`, `mysqli_stmt_bind_param`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::bind_param`

`mysqli_stmt_bind_param`

Binds variables to a prepared statement as parameters

Description

Object oriented style

```
bool mysqli_stmt::bind_param(
    string types,
    mixed var1,
    mixed ...);
```

Procedural style

```
bool mysqli_stmt_bind_param(
    mysqli_stmt stmt,
    string types,
    mixed var1,
    mixed ...);
```

Bind variables for the parameter markers in the SQL statement that was passed to `mysqli_prepare`.**Note**

If data size of a variable exceeds `max_allowed_packet`, you have to specify `b` in `types` and use `mysqli_stmt_send_long_data` to send the data in packets.

**Note**

Care must be taken when using `mysqli_stmt_bind_param` in conjunction with `call_user_func_array`. Note that `mysqli_stmt_bind_param` requires parameters to be passed by reference, whereas `call_user_func_array` can accept as a parameter a list of variables that can represent references or values.

Parameters*stmt*Procedural style only: A statement identifier returned by `mysqli_stmt_init`.*types*

A string that contains one or more characters which specify the types for the corresponding bind variables:

Table 22.48. Type specification chars

Character	Description
i	corresponding variable has type integer
d	corresponding variable has type double

Character	Description
s	corresponding variable has type string
b	corresponding variable is a blob and will be sent in packets

var1

The number of variables and length of string *types* must match the parameters in the statement.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example 22.151. Object oriented style

```
<?php
$mysqli = new mysqli('localhost', 'my_user', 'my_password', 'world');

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$stmt = $mysqli->prepare("INSERT INTO CountryLanguage VALUES (?, ?, ?, ?)");
$stmt->bind_param('sssd', $code, $language, $official, $percent);

$code = 'DEU';
$language = 'Bavarian';
$official = "F";
$percent = 11.2;

/* execute prepared statement */
$stmt->execute();

printf("%d Row inserted.\n", $stmt->affected_rows);

/* close statement and connection */
$stmt->close();

/* Clean up table CountryLanguage */
$mysqli->query("DELETE FROM CountryLanguage WHERE Language='Bavarian'");
printf("%d Row deleted.\n", $mysqli->affected_rows);

/* close connection */
$mysqli->close();
?>
```

Example 22.152. Procedural style

```
<?php
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'world');

/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
```

```
        exit();
    }

    $stmt = mysqli_prepare($link, "INSERT INTO CountryLanguage VALUES (?, ?, ?, ?)");
    mysqli_stmt_bind_param($stmt, 'sssd', $code, $language, $official, $percent);

    $code = 'DEU';
    $language = 'Bavarian';
    $official = "F";
    $percent = 11.2;

    /* execute prepared statement */
    mysqli_stmt_execute($stmt);

    printf("%d Row inserted.\n", mysqli_stmt_affected_rows($stmt));

    /* close statement and connection */
    mysqli_stmt_close($stmt);

    /* Clean up table CountryLanguage */
    mysqli_query($link, "DELETE FROM CountryLanguage WHERE Language='Bavarian'");
    printf("%d Row deleted.\n", mysqli_affected_rows($link));

    /* close connection */
    mysqli_close($link);
    ?>
```

The above examples will output:

```
1 Row inserted.
1 Row deleted.
```

See Also

[mysqli_stmt_bind_result](#)
[mysqli_stmt_execute](#)
[mysqli_stmt_fetch](#)
[mysqli_prepare](#)
[mysqli_stmt_send_long_data](#)
[mysqli_stmt_errno](#)
[mysqli_stmt_error](#)

22.9.3.10.5. [mysqli_stmt::bind_result](#), [mysqli_stmt_bind_result](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::bind_result](#)

[mysqli_stmt_bind_result](#)

Binds variables to a prepared statement for result storage

Description

Object oriented style

```
bool mysqli_stmt::bind_result(
```

```
mixed var1,
mixed ...);
```

Procedural style

```
bool mysqli_stmt_bind_result(
    mysqli_stmt stmt,
    mixed var1,
    mixed ...);
```

Binds columns in the result set to variables.

When `mysqli_stmt_fetch` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified variables `var1`,



Note

Note that all columns must be bound after `mysqli_stmt_execute` and prior to calling `mysqli_stmt_fetch`. Depending on column types bound variables can silently change to the corresponding PHP type.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysqli_stmt_fetch` is called.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

var1 The variable to be bound.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.153. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* prepare statement */
if ($stmt = $mysqli->prepare("SELECT Code, Name FROM Country ORDER BY Name LIMIT 5")) {
    $stmt->execute();

    /* bind variables to prepared statement */
    $stmt->bind_result($col1, $col2);

    /* fetch values */
    while ($stmt->fetch()) {
        printf("%s %s\n", $col1, $col2);
    }
}
```

```
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();

?>
```

Example 22.154. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* prepare statement */
if ($stmt = mysqli_prepare($link, "SELECT Code, Name FROM Country ORDER BY Name LIMIT 5")) {
    mysqli_stmt_execute($stmt);

    /* bind variables to prepared statement */
    mysqli_stmt_bind_result($stmt, $col1, $col2);

    /* fetch values */
    while (mysqli_stmt_fetch($stmt)) {
        printf("%s %s\n", $col1, $col2);
    }

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);

?>
```

The above examples will output:

```
AFG Afghanistan
ALB Albania
DZA Algeria
ASM American Samoa
AND Andorra
```

See Also

[mysqli_stmt_bind_param](#)
[mysqli_stmt_execute](#)
[mysqli_stmt_fetch](#)
[mysqli_prepare](#)
[mysqli_stmt_prepare](#)
[mysqli_stmt_init](#)

```
mysqli_stmt_errno  
mysqli_stmt_error
```

22.9.3.10.6. `mysqli_stmt::close`, `mysqli_stmt_close`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::close`

```
mysqli_stmt_close
```

Closes a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::close();
```

Procedural style

```
bool mysqli_stmt_close(  
    mysqli_stmt stmt);
```

Closes a prepared statement. `mysqli_stmt_close` also deallocates the statement handle. If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

See Also

`mysqli_prepare`

22.9.3.10.7. `mysqli_stmt::data_seek`, `mysqli_stmt_data_seek`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::data_seek`

```
mysqli_stmt_data_seek
```

Seeks to an arbitrary row in statement result set

Description

Object oriented style

```
void mysqli_stmt::data_seek(  
    int offset);
```

Procedural style

```
void mysqli_stmt_data_seek(
    mysqli_stmt stmt,
    int offset);
```

Seeks to an arbitrary result pointer in the statement result set.

[mysqli_stmt_store_result](#) must be called prior to [mysqli_stmt_data_seek](#).

Parameters

stmt

Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

offset

Must be between zero and the total number of rows minus one (0.. [mysqli_stmt_num_rows](#) - 1).

Return Values

No value is returned.

Examples

Example 22.155. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {

    /* execute query */
    $stmt->execute();

    /* bind result variables */
    $stmt->bind_result($name, $code);

    /* store result */
    $stmt->store_result();

    /* seek to row no. 400 */
    $stmt->data_seek(399);

    /* fetch values */
    $stmt->fetch();

    printf ("City: %s Countrycode: %s\n", $name, $code);

    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.156. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {

    /* execute query */
    mysqli_stmt_execute($stmt);

    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $name, $code);

    /* store result */
    mysqli_stmt_store_result($stmt);

    /* seek to row no. 400 */
    mysqli_stmt_data_seek($stmt, 399);

    /* fetch values */
    mysqli_stmt_fetch($stmt);

    printf ("City: %s Countrycode: %s\n", $name, $code);

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
City: Benin City Countrycode: NGA
```

See Also

[mysqli_prepare](#)

22.9.3.10.8. [mysqli_stmt::\\$errno](#), [mysqli_stmt_errno](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::\\$errno](#)

[mysqli_stmt_errno](#)

Returns the error code for the most recent statement call

Description

Object oriented style

```
int
mysqli_stmt->errno ;
```

Procedural style

```
int mysqli_stmt_errno(
    mysqli_stmt stmt);
```

Returns the error code for the most recently invoked statement function that can succeed or fail.

Client error message numbers are listed in the MySQL [errmsg.h](#) header file, server error message numbers are listed in [mysqld_error.h](#). In the MySQL source distribution you can find a complete list of error messages and error numbers in the file [Docs/mysqld_error.txt](#).

Parameters

stmt Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

An error code value. Zero means no error occurred.

Examples

Example 22.157. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");

$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {

    /* drop table */
    $mysqli->query("DROP TABLE myCountry");

    /* execute query */
    $stmt->execute();

    printf("Error: %d.\n", $stmt->errno);

    /* close statement */
```

```
$stmt->close();
}

/* close connection */
mysqli->close();
?>
```

Example 22.158. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");

$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {

    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");

    /* execute query */
    mysqli_stmt_execute($stmt);

    printf("Error: %d.\n", mysqli_stmt_errno($stmt));

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: 1146.
```

See Also

[mysqli_stmt_error](#)
[mysqli_stmt_sqlstate](#)

22.9.3.10.9. [mysqli_stmt::\\$error_list](#), [mysqli_stmt_error_list](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::\\$error_list](#)

`mysqli_stmt_error_list`

Returns a list of errors from the last statement executed

Description

Object oriented style

```
array  
mysqli_stmt->error_list ;
```

Procedural style

```
array mysqli_stmt_error_list(  
mysqli_stmt stmt);
```

Returns an array of errors for the most recently invoked statement function that can succeed or fail.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

A list of errors, each as an associative array containing the `errno`, `error`, and `sqlstate`.

Examples

Example 22.159. Object oriented style

```
<?php  
/* Open a connection */  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
  
$mysqli->query("CREATE TABLE myCountry LIKE Country");  
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");  
  
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";  
if ($stmt = $mysqli->prepare($query)) {  
  
    /* drop table */  
    $mysqli->query("DROP TABLE myCountry");  
  
    /* execute query */  
    $stmt->execute();  
  
    echo "Error:\n";  
    print_r($stmt->error_list);  
  
    /* close statement */  
    $stmt->close();  
}
```

```
/* close connection */
mysqli->close();
?>
```

Example 22.160. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");

$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {

    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");

    /* execute query */
    mysqli_stmt_execute($stmt);

    echo "Error:\n";
    print_r(mysqli_stmt_error_list($stmt));

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Array
(
    [0] => Array
        (
            [errno] => 1146
            [sqlstate] => 42S02
            [error] => Table 'world.myCountry' doesn't exist
        )
)
```

See Also

[mysqli_stmt_error](#)

```
mysqli_stmt_errno  
mysqli_stmt_sqlstate
```

22.9.3.10.10. *mysqli_stmt::\$error*, *mysqli_stmt_error*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysqli_stmt::\$error*

```
mysqli_stmt_error
```

Returns a string description for last statement error

Description

Object oriented style

```
string  
mysqli_stmt->error ;
```

Procedural style

```
string mysqli_stmt_error(  
mysqli_stmt stmt);
```

Returns a containing the error message for the most recently invoked statement function that can succeed or fail.

Parameters

stmt Procedural style only: A statement identifier returned by *mysqli_stmt_init*.

Return Values

A string that describes the error. An empty string if no error occurred.

Examples

Example 22.161. Object oriented style

```
<?php  
/* Open a connection */  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
  
$mysqli->query("CREATE TABLE myCountry LIKE Country");  
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");  
  
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";  
if ($stmt = $mysqli->prepare($query)) {  
  
    /* drop table */
```

```
$mysqli->query("DROP TABLE myCountry");

/* execute query */
$stmt->execute();

printf("Error: %s.\n", $stmt->error);

/* close statement */
$stmt->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.162. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");

$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {

    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");

    /* execute query */
    mysqli_stmt_execute($stmt);

    printf("Error: %s.\n", mysqli_stmt_error($stmt));

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: Table 'world.myCountry' doesn't exist.
```

See Also

[mysqli_stmt_errno](#)

[mysqli_stmt_sqlstate](#)

22.9.3.10.11. [mysqli_stmt::execute](#), [mysqli_stmt_execute](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::execute](#)

[mysqli_stmt_execute](#)

Executes a prepared Query

Description

Object oriented style

```
bool mysqli_stmt::execute();
```

Procedural style

```
bool mysqli_stmt_execute(  
    mysqli_stmt stmt);
```

Executes a query that has been previously prepared using the [mysqli_prepare](#) function. When executed any parameter markers which exist will automatically be replaced with the appropriate data.

If the statement is [UPDATE](#), [DELETE](#), or [INSERT](#), the total number of affected rows can be determined by using the [mysqli_stmt_affected_rows](#) function. Likewise, if the query yields a result set the [mysqli_stmt_fetch](#) function is used.



Note

When using [mysqli_stmt_execute](#), the [mysqli_stmt_fetch](#) function must be used to fetch the data prior to performing any additional queries.

Parameters

stmt Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.163. Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}
```

```

$mysqli->query("CREATE TABLE myCity LIKE City");

/* Prepare an insert statement */
$query = "INSERT INTO myCity (Name, CountryCode, District) VALUES (?, ?, ?)";
$stmt = $mysqli->prepare($query);

$stmt->bind_param("sss", $val1, $val2, $val3);

$val1 = 'Stuttgart';
$val2 = 'DEU';
$val3 = 'Baden-Wuerttemberg';

/* Execute the statement */
$stmt->execute();

$val1 = 'Bordeaux';
$val2 = 'FRA';
$val3 = 'Aquitaine';

/* Execute the statement */
$stmt->execute();

/* close statement */
$stmt->close();

/* retrieve all rows from myCity */
$query = "SELECT Name, CountryCode, District FROM myCity";
if ($result = $mysqli->query($query)) {
    while ($row = $result->fetch_row()) {
        printf("%s (%s,%s)\n", $row[0], $row[1], $row[2]);
    }
    /* free result set */
    $result->close();
}

/* remove table */
$mysqli->query("DROP TABLE myCity");

/* close connection */
$mysqli->close();
?>

```

Example 22.164. Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

mysqli_query($link, "CREATE TABLE myCity LIKE City");

/* Prepare an insert statement */
$query = "INSERT INTO myCity (Name, CountryCode, District) VALUES (?, ?, ?)";
$stmt = mysqli_prepare($link, $query);

mysqli_stmt_bind_param($stmt, "sss", $val1, $val2, $val3);

$val1 = 'Stuttgart';
$val2 = 'DEU';
$val3 = 'Baden-Wuerttemberg';

```

```
/* Execute the statement */
mysqli_stmt_execute($stmt);

$val1 = 'Bordeaux';
$val2 = 'FRA';
$val3 = 'Aquitaine';

/* Execute the statement */
mysqli_stmt_execute($stmt);

/* close statement */
mysqli_stmt_close($stmt);

/* retrieve all rows from myCity */
$query = "SELECT Name, CountryCode, District FROM myCity";
if ($result = mysqli_query($link, $query)) {
    while ($row = mysqli_fetch_row($result)) {
        printf("%s (%s,%s)\n", $row[0], $row[1], $row[2]);
    }
    /* free result set */
    mysqli_free_result($result);
}

/* remove table */
mysqli_query($link, "DROP TABLE myCity");

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Stuttgart (DEU,Baden-Wuerttemberg)
Bordeaux (FRA,Aquitaine)
```

See Also

[mysqli_prepare](#)
[mysqli_stmt_bind_param](#)

22.9.3.10.12. [mysqli_stmt::fetch](#), [mysqli_stmt_fetch](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::fetch](#)

[mysqli_stmt_fetch](#)

Fetch results from a prepared statement into the bound variables

Description

Object oriented style

```
bool mysqli_stmt::fetch();
```

Procedural style

```
bool mysqli_stmt_fetch(
    mysqli_stmt stmt);
```

Fetch the result from a prepared statement into the variables bound by `mysqli_stmt_bind_result`.



Note

Note that all columns must be bound by the application before calling `mysqli_stmt_fetch`.



Note

Data are transferred unbuffered without calling `mysqli_stmt_store_result` which can decrease performance (but reduces memory cost).

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Table 22.49. Return Values

Value	Description
TRUE	Success. Data has been fetched
FALSE	Error occurred
NULL	No more rows/data exists or data truncation occurred

Examples

Example 22.165. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 150,5";

if ($stmt = $mysqli->prepare($query)) {

    /* execute statement */
    $stmt->execute();

    /* bind result variables */
    $stmt->bind_result($name, $code);

    /* fetch values */
    while ($stmt->fetch()) {
        printf ("%s (%s)\n", $name, $code);
    }
}
```

```
    /* close statement */
    $stmt->close();
}

/* close connection */
mysqli->close();
?>
```

Example 22.166. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 150,5";

if ($stmt = mysqli_prepare($link, $query)) {

    /* execute statement */
    mysqli_stmt_execute($stmt);

    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $name, $code);

    /* fetch values */
    while (mysqli_stmt_fetch($stmt)) {
        printf ("%s (%s)\n", $name, $code);
    }

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Rockford (USA)
Tallahassee (USA)
Salinas (USA)
Santa Clarita (USA)
Springfield (USA)
```

See Also

[mysqli_prepare](#)
[mysqli_stmt_errno](#)
[mysqli_stmt_error](#)

`mysqli_stmt_bind_result`

22.9.3.10.13. `mysqli_stmt::$field_count`, `mysqli_stmt_field_count`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::$field_count`

`mysqli_stmt_field_count`

Returns the number of field in the given statement

Description

Object oriented style

```
int
mysqli_stmt->field_count ;
```

Procedural style

```
int mysqli_stmt_field_count(
mysqli_stmt stmt);
```



Warning

This function is currently not documented; only its argument list is available.

22.9.3.10.14. `mysqli_stmt::free_result`, `mysqli_stmt_free_result`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::free_result`

`mysqli_stmt_free_result`

Frees stored result memory for the given statement handle

Description

Object oriented style

```
void mysqli_stmt::free_result();
```

Procedural style

```
void mysqli_stmt_free_result(
mysqli_stmt stmt);
```

Frees the result memory associated with the statement, which was allocated by `mysqli_stmt_store_result`.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

No value is returned.

See Also

[mysqli_stmt_store_result](#)

22.9.3.10.15. [mysqli_stmt::get_result](#), [mysqli_stmt_get_result](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::get_result](#)

[mysqli_stmt_get_result](#)

Gets a result set from a prepared statement

Description

Object oriented style

```
mysqli_result mysqli_stmt::get_result();
```

Procedural style

```
mysqli_result mysqli_stmt_get_result(
    mysqli_stmt stmt);
```

Call to return a result set from a prepared statement query.

Parameters

stmt Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

Returns a resultset or `FALSE` on failure.

MySQL Native Driver Only

Available only with [mysqlnd](#).

Examples

Example 22.167. Object oriented style

```
<?php
$mysqli = new mysqli("127.0.0.1", "user", "password", "world");

if($mysqli->connect_error)
{
    die("$mysqli->connect_errno: $mysqli->connect_error");
}

$query = "SELECT Name, Population, Continent FROM Country WHERE Continent=? ORDER BY Name LIMIT 1";
$stmt = $mysqli->stmt_init();
```

```

if(!$stmt->prepare($query))
{
    print "Failed to prepare statement\n";
}
else
{
    $stmt->bind_param("s", $continent);

    $continent_array = array('Europe','Africa','Asia','North America');

    foreach($continent_array as $continent)
    {
        $stmt->execute();
        $result = $stmt->get_result();
        while ($row = $result->fetch_array(MYSQLI_NUM))
        {
            foreach ($row as $r)
            {
                print "$r ";
            }
            print "\n";
        }
    }

    $stmt->close();
    $mysqli->close();
?>

```

Example 22.168. Procedural style

```

<?php

$link = mysqli_connect("127.0.0.1", "user", "password", "world");

if (!$link)
{
    $error = mysqli_connect_error();
    $errno = mysqli_connect_errno();
    print "$errno: $error\n";
    exit();
}

$query = "SELECT Name, Population, Continent FROM Country WHERE Continent=? ORDER BY Name LIMIT 1";

$stmt = mysqli_stmt_init($link);
if(!mysqli_stmt_prepare($stmt, $query))
{
    print "Failed to prepare statement\n";
}
else
{
    mysqli_stmt_bind_param($stmt, "s", $continent);

    $continent_array = array('Europe','Africa','Asia','North America');

    foreach($continent_array as $continent)
    {
        mysqli_stmt_execute($stmt);
        $result = mysqli_stmt_get_result($stmt);
        while ($row = mysqli_fetch_array($result, MYSQLI_NUM))
        {
            foreach ($row as $r)
            {

```

```
        print "$r ";
    }
    print "\n";
}
}
}
mysqli_stmt_close($stmt);
mysqli_close($link);
?>
```

The above examples will output:

```
Albania 3401200 Europe
Algeria 31471000 Africa
Afghanistan 22720000 Asia
Anguilla 8000 North America
```

See Also

[mysqli_prepare](#)
[mysqli_stmt_result_metadata](#)
[mysqli_stmt_fetch](#)
[mysqli_fetch_array](#)
[mysqli_stmt_store_result](#)

22.9.3.10.16. [mysqli_stmt::get_warnings](#), [mysqli_stmt_get_warnings](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::get_warnings](#)

[mysqli_stmt_get_warnings](#)

Get result of SHOW WARNINGS

Description

Object oriented style

```
object mysqli_stmt::get_warnings(
    mysqli_stmt $stmt);
```

Procedural style

```
object mysqli_stmt_get_warnings(
    mysqli_stmt $stmt);
```



Warning

This function is currently not documented; only its argument list is available.

22.9.3.10.17. [mysqli_stmt::\\$insert_id](#), [mysqli_stmt_insert_id](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::$insert_id`

`mysqli_stmt_insert_id`

Get the ID generated from the previous INSERT operation

Description

Object oriented style

```
int
mysqli_stmt->insert_id ;
```

Procedural style

```
mixed mysqli_stmt_insert_id(
mysqli_stmt stmt);
```



Warning

This function is currently not documented; only its argument list is available.

22.9.3.10.18. `mysqli_stmt::more_results`, `mysqli_stmt_more_results`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::more_results`

`mysqli_stmt_more_results`

Check if there are more query results from a multiple query

Description

Object oriented style (method):

```
public bool mysqli_stmt::more_results();
```

Procedural style:

```
bool mysqli_stmt_more_results(
mysqli_stmt stmt);
```

Checks if there are more query results from a multiple query.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns `TRUE` if more results exist, otherwise `FALSE`.

See Also

`mysqli_stmt::next_result`

```
mysqli::multi_query
```

22.9.3.10.19. `mysqli_stmt::next_result`, `mysqli_stmt_next_result`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::next_result`

```
mysqli_stmt_next_result
```

Reads the next result from a multiple query

Description

Object oriented style (method):

```
public bool mysqli_stmt::next_result();
```

Procedural style:

```
bool mysqli_stmt_next_result(  
    mysqli_stmt stmt);
```

Reads the next result from a multiple query.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Errors/Exceptions

Emits an `E_STRICT` level error if a result set does not exist, and suggests using `mysqli_stmt::more_results` in these cases, before calling `mysqli_stmt::next_result`.

See Also

```
mysqli_stmt::more_results  
mysqli::multi_query
```

22.9.3.10.20. `mysqli_stmt::$num_rows`, `mysqli_stmt_num_rows`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::$num_rows`

```
mysqli_stmt_num_rows
```

Return the number of rows in statements result set

Description

Object oriented style

```
int
mysqli_stmt->num_rows ;
```

Procedural style

```
int mysqli_stmt_num_rows(
mysqli_stmt stmt);
```

Returns the number of rows in the result set. The use of `mysqli_stmt_num_rows` depends on whether or not you used `mysqli_stmt_store_result` to buffer the entire result set in the statement handle.

If you use `mysqli_stmt_store_result`, `mysqli_stmt_num_rows` may be called immediately.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

An integer representing the number of rows in result set.

Examples

Example 22.169. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = $mysqli->prepare($query)) {

    /* execute query */
    $stmt->execute();

    /* store result */
    $stmt->store_result();

    printf("Number of rows: %d.\n", $stmt->num_rows);

    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.170. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = mysqli_prepare($link, $query)) {

    /* execute query */
    mysqli_stmt_execute($stmt);

    /* store result */
    mysqli_stmt_store_result($stmt);

    printf("Number of rows: %d.\n", mysqli_stmt_num_rows($stmt));

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Number of rows: 20.
```

See Also

[mysqli_stmt_affected_rows](#)
[mysqli_prepare](#)
[mysqli_stmt_store_result](#)

22.9.3.10.21. [mysqli_stmt::\\$param_count](#), [mysqli_stmt_param_count](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::\\$param_count](#)

[mysqli_stmt_param_count](#)

Returns the number of parameter for the given statement

Description

Object oriented style

```
int
mysqli_stmt->param_count ;
```

Procedural style

```
int mysqli_stmt_param_count(
    mysqli_stmt stmt);
```

Returns the number of parameter markers present in the prepared statement.

Parameters

stmt Procedural style only: A statement identifier returned by *mysqli_stmt_init*.

Return Values

Returns an integer representing the number of parameters.

Examples

Example 22.171. Object oriented style

```
<?php
$link = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if ($stmt = $link->prepare("SELECT Name FROM Country WHERE Name=? OR Code=?")) {

    $marker = $stmt->param_count;
    printf("Statement has %d markers.\n", $marker);

    /* close statement */
    $stmt->close();
}

/* close connection */
$link->close();
?>
```

Example 22.172. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if ($stmt = mysqli_prepare($link, "SELECT Name FROM Country WHERE Name=? OR Code=?")) {

    $marker = mysqli_stmt_param_count($stmt);
    printf("Statement has %d markers.\n", $marker);

    /* close statement */
    mysqli_stmt_close($stmt);
}
```

```
}  
  
/* close connection */  
mysqli_close($link);  
?>
```

The above examples will output:

```
Statement has 2 markers.
```

See Also

[mysqli_prepare](#)

22.9.3.10.22. [mysqli_stmt::prepare](#), [mysqli_stmt_prepare](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::prepare](#)

[mysqli_stmt_prepare](#)

Prepare an SQL statement for execution

Description

Object oriented style

```
mixed mysqli_stmt::prepare(  
    string query);
```

Procedural style

```
bool mysqli_stmt_prepare(  
    mysqli_stmt stmt,  
    string query);
```

Prepares the SQL query pointed to by the null-terminated string query.

The parameter markers must be bound to application variables using [mysqli_stmt_bind_param](#) and/or [mysqli_stmt_bind_result](#) before executing the statement or fetching rows.



Note

In the case where you pass a statement to [mysqli_stmt_prepare](#) that is longer than [max_allowed_packet](#) of the server, the returned error codes are different depending on whether you are using MySQL Native Driver ([mysqlnd](#)) or MySQL Client Library ([libmysql](#)). The behavior is as follows:

- [mysqlnd](#) on Linux returns an error code of 1153. The error message means “got a packet bigger than [max_allowed_packet](#) bytes”.
- [mysqlnd](#) on Windows returns an error code 2006. This error message means “server has gone away”.

- `libmysql` on all platforms returns an error code 2006. This error message means “server has gone away”.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

query

The query, as a string. It must consist of a single SQL statement.

You can include one or more parameter markers in the SQL statement by embedding question mark (?) characters at the appropriate positions.



Note

You should not add a terminating semicolon or `\g` to the statement.



Note

The markers are legal only in certain places in SQL statements. For example, they are allowed in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value.

However, they are not allowed for identifiers (such as table or column names), in the select list that names the columns to be returned by a `SELECT` statement), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.173. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
```

```
$city = "Amersfoort";

/* create a prepared statement */
$stmt = $mysqli->stmt_init();
if ($stmt->prepare("SELECT District FROM City WHERE Name=?")) {

    /* bind parameters for markers */
    $stmt->bind_param("s", $city);

    /* execute query */
    $stmt->execute();

    /* bind result variables */
    $stmt->bind_result($district);

    /* fetch value */
    $stmt->fetch();

    printf("%s is in district %s\n", $city, $district);

    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.174. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$city = "Amersfoort";

/* create a prepared statement */
$stmt = mysqli_stmt_init($link);
if (mysqli_stmt_prepare($stmt, 'SELECT District FROM City WHERE Name=?')) {

    /* bind parameters for markers */
    mysqli_stmt_bind_param($stmt, "s", $city);

    /* execute query */
    mysqli_stmt_execute($stmt);

    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $district);

    /* fetch value */
    mysqli_stmt_fetch($stmt);

    printf("%s is in district %s\n", $city, $district);

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
```

```
mysqli_close($link);  
?>
```

The above examples will output:

```
Amersfoort is in district Utrecht
```

See Also

```
mysqli_stmt_init  
mysqli_stmt_execute  
mysqli_stmt_fetch  
mysqli_stmt_bind_param  
mysqli_stmt_bind_result  
mysqli_stmt_close
```

22.9.3.10.23. *mysqli_stmt::reset*, *mysqli_stmt_reset*

Copyright 1997-2012 the PHP Documentation Group. [3186]

- *mysqli_stmt::reset*

```
mysqli_stmt_reset
```

Resets a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::reset();
```

Procedural style

```
bool mysqli_stmt_reset(  
    mysqli_stmt stmt);
```

Resets a prepared statement on client and server to state after prepare.

It resets the statement on the server, data sent using *mysqli_stmt_send_long_data*, unbuffered result sets and current errors. It does not clear bindings or stored result sets. Stored result sets will be cleared when executing the prepared statement (or closing it).

To prepare a statement with another query use function *mysqli_stmt_prepare*.

Parameters

stmt

Procedural style only: A statement identifier returned by *mysqli_stmt_init*.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also[mysqli_prepare](#)**22.9.3.10.24. [mysqli_stmt::result_metadata](#), [mysqli_stmt_result_metadata](#)**

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::result_metadata](#)

[mysqli_stmt_result_metadata](#)

Returns result set metadata from a prepared statement

Description

Object oriented style

```
mysqli_result mysqli_stmt::result_metadata();
```

Procedural style

```
mysqli_result mysqli_stmt_result_metadata(  
    mysqli_stmt stmt);
```

If a statement passed to [mysqli_prepare](#) is one that produces a result set, [mysqli_stmt_result_metadata](#) returns the result object that can be used to process the meta information such as total number of fields and individual field information.

**Note**

This result set pointer can be passed as an argument to any of the field-based functions that process result set metadata, such as:

- [mysqli_num_fields](#)
- [mysqli_fetch_field](#)
- [mysqli_fetch_field_direct](#)
- [mysqli_fetch_fields](#)
- [mysqli_field_count](#)
- [mysqli_field_seek](#)
- [mysqli_field_tell](#)
- [mysqli_free_result](#)

The result set structure should be freed when you are done with it, which you can do by passing it to [mysqli_free_result](#)

**Note**

The result set returned by [mysqli_stmt_result_metadata](#) contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with [mysqli_stmt_fetch](#).

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns a result object or `FALSE` if an error occurred.

Examples

Example 22.175. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");

$mysqli->query("DROP TABLE IF EXISTS friends");
$mysqli->query("CREATE TABLE friends (id int, name varchar(20))");

$mysqli->query("INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");

$stmt = $mysqli->prepare("SELECT id, name FROM friends");
$stmt->execute();

/* get resultset for metadata */
$result = $stmt->result_metadata();

/* retrieve field information from metadata result set */
$field = $result->fetch_field();

printf("Fieldname: %s\n", $field->name);

/* close resultset */
$result->close();

/* close connection */
$mysqli->close();
?>
```

Example 22.176. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");

mysqli_query($link, "DROP TABLE IF EXISTS friends");
mysqli_query($link, "CREATE TABLE friends (id int, name varchar(20))");

mysqli_query($link, "INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");

$stmt = mysqli_prepare($link, "SELECT id, name FROM friends");
mysqli_stmt_execute($stmt);

/* get resultset for metadata */
$result = mysqli_stmt_result_metadata($stmt);

/* retrieve field information from metadata result set */
$field = mysqli_fetch_field($result);

printf("Fieldname: %s\n", $field->name);
```

```
/* close resultset */
mysqli_free_result($result);

/* close connection */
mysqli_close($link);
?>
```

See Also

[mysqli_prepare](#)
[mysqli_free_result](#)

22.9.3.10.25. [mysqli_stmt::send_long_data](#), [mysqli_stmt_send_long_data](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::send_long_data](#)

[mysqli_stmt_send_long_data](#)

Send data in blocks

Description

Object oriented style

```
bool mysqli_stmt::send_long_data(
    int param_nr,
    string data);
```

Procedural style

```
bool mysqli_stmt_send_long_data(
    mysqli_stmt stmt,
    int param_nr,
    string data);
```

Allows to send parameter data to the server in pieces (or chunks), e.g. if the size of a blob exceeds the size of [max_allowed_packet](#). This function can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the TEXT or BLOB datatypes.

Parameters

<i>stmt</i>	Procedural style only: A statement identifier returned by mysqli_stmt_init .
<i>param_nr</i>	Indicates which parameter to associate the data with. Parameters are numbered beginning with 0.
<i>data</i>	A string containing data to be sent.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 22.177. Object oriented style

```
<?php
$stmt = $mysqli->prepare("INSERT INTO messages (message) VALUES (?)");
>null = NULL;
$stmt->bind_param("b", $null);
$fp = fopen("messages.txt", "r");
while (!feof($fp)) {
    $stmt->send_long_data(0, fread($fp, 8192));
}
fclose($fp);
$stmt->execute();
?>
```

See Also

[mysqli_prepare](#)
[mysqli_stmt_bind_param](#)

22.9.3.10.26. `mysqli_stmt::$sqlstate`, `mysqli_stmt_sqlstate`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_stmt::$sqlstate`

`mysqli_stmt_sqlstate`

Returns SQLSTATE error from previous statement operation

Description

Object oriented style

```
string
mysqli_stmt->sqlstate ;
```

Procedural style

```
string mysqli_stmt_sqlstate(
    mysqli_stmt stmt);
```

Returns a string containing the SQLSTATE error code for the most recently invoked prepared statement function that can succeed or fail. The error code consists of five characters. '00000' means no error. The values are specified by ANSI SQL and ODBC. For a list of possible values, see <http://dev.mysql.com/doc/mysql/en/error-handling.html>.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. '00000' means no error.

Notes

**Note**

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value `HY000` (general error) is used for unmapped errors.

Examples**Example 22.178. Object oriented style**

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");

$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {

    /* drop table */
    $mysqli->query("DROP TABLE myCountry");

    /* execute query */
    $stmt->execute();

    printf("Error: %s.\n", $stmt->sqlstate);

    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.179. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");

$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
```

```
/* drop table */
mysqli_query($link, "DROP TABLE myCountry");

/* execute query */
mysqli_stmt_execute($stmt);

printf("Error: %s.\n", mysqli_stmt_sqlstate($stmt));

/* close statement */
mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: 42S02.
```

See Also

[mysqli_stmt_errno](#)
[mysqli_stmt_error](#)

22.9.3.10.27. [mysqli_stmt::store_result](#), [mysqli_stmt_store_result](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_stmt::store_result](#)

[mysqli_stmt_store_result](#)

Transfers a result set from a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::store_result();
```

Procedural style

```
bool mysqli_stmt_store_result(
    mysqli_stmt stmt);
```

You must call [mysqli_stmt_store_result](#) for every query that successfully produces a result set ([SELECT](#), [SHOW](#), [DESCRIBE](#), [EXPLAIN](#)), and only if you want to buffer the complete result set by the client, so that the subsequent [mysqli_stmt_fetch](#) call returns buffered data.



Note

It is unnecessary to call [mysqli_stmt_store_result](#) for other queries, but if you do, it will not harm or cause any notable performance in all cases.

You can detect whether the query produced a result set by checking if `mysqli_stmt_result_metadata` returns NULL.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.180. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = $mysqli->prepare($query)) {

    /* execute query */
    $stmt->execute();

    /* store result */
    $stmt->store_result();

    printf("Number of rows: %d.\n", $stmt->num_rows);

    /* free result */
    $stmt->free_result();

    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.181. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
```

```
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = mysqli_prepare($link, $query)) {

    /* execute query */
    mysqli_stmt_execute($stmt);

    /* store result */
    mysqli_stmt_store_result($stmt);

    printf("Number of rows: %d.\n", mysqli_stmt_num_rows($stmt));

    /* free result */
    mysqli_stmt_free_result($stmt);

    /* close statement */
    mysqli_stmt_close($stmt);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Number of rows: 20.
```

See Also

[mysqli_prepare](#)
[mysqli_stmt_result_metadata](#)
[mysqli_stmt_fetch](#)

22.9.3.11. The `mysqli_result` class ([mysqli_result](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

Represents the result set obtained from a query against the database.

Changelog

Table 22.50. Changelog

Version	Description
5.4.0	Iterator support was added, as <code>mysqli_result</code> now implements Traversable .

```
mysqli_result {
    mysqli_result

        Traversable

        Properties

    int
```

```
mysqli_result->current_field ;

int
mysqli_result->field_count ;

array
mysqli_result->lengths ;

int
mysqli_result->num_rows ;

Methods

bool mysqli_result::data_seek(
    int offset);

mixed mysqli_result::fetch_all(
    int resulttype
    = =MYSQLI_NUM);

mixed mysqli_result::fetch_array(
    int resulttype
    = =MYSQLI_BOTH);

array mysqli_result::fetch_assoc();

object mysqli_result::fetch_field_direct(
    int fieldnr);

object mysqli_result::fetch_field();

array mysqli_result::fetch_fields();

object mysqli_result::fetch_object(
    string class_name,
    array params);

mixed mysqli_result::fetch_row();

bool mysqli_result::field_seek(
    int fieldnr);

void mysqli_result::free();
}
```

22.9.3.11.1. `mysqli_result::$current_field`, `mysqli_field_tell`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_result::$current_field`

`mysqli_field_tell`

Get current field offset of a result pointer

Description

Object oriented style

```
int
mysqli_result->current_field ;
```

Procedural style

```
int mysqli_field_tell(
    mysqli_result result);
```

Returns the position of the field cursor used for the last `mysqli_fetch_field` call. This value can be used as an argument to `mysqli_field_seek`.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns current offset of field cursor.

Examples

Example 22.182. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";

if ($result = $mysqli->query($query)) {

    /* Get field information for all columns */
    while ($finfo = $result->fetch_field()) {

        /* get fieldpointer offset */
        $currentfield = $result->current_field;

        printf("Column %d:\n", $currentfield);
        printf("Name:      %s\n", $finfo->name);
        printf("Table:     %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.183. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";

if ($result = mysqli_query($link, $query)) {

    /* Get field information for all fields */
    while ($finfo = mysqli_fetch_field($result)) {

        /* get fieldpointer offset */
        $currentfield = mysqli_field_tell($result);

        printf("Column %d:\n", $currentfield);
        printf("Name:      %s\n", $finfo->name);
        printf("Table:     %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Column 1:
Name:      Name
Table:     Country
max. Len:  11
Flags:     1
Type:      254

Column 2:
Name:      SurfaceArea
Table:     Country
max. Len:  10
Flags:     32769
Type:      4
```

See Also

[mysqli_fetch_field](#)
[mysqli_field_seek](#)

22.9.3.11.2. [mysqli_result::data_seek](#), [mysqli_data_seek](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_result::data_seek](#)
[mysqli_data_seek](#)

Adjusts the result pointer to an arbitrary row in the result

Description

Object oriented style

```
bool mysqli_result::data_seek(
    int offset);
```

Procedural style

```
bool mysqli_data_seek(
    mysqli_result result,
    int offset);
```

The `mysqli_data_seek` function seeks to an arbitrary result pointer specified by the `offset` in the result set.

Parameters

<i>result</i>	Procedural style only: A result set identifier returned by <code>mysqli_query</code> , <code>mysqli_store_result</code> or <code>mysqli_use_result</code> .
<i>offset</i>	The field offset. Must be between zero and the total number of rows minus one (<code>0..mysqli_num_rows - 1</code>).

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Notes



Note

This function can only be used with buffered results attained from the use of the `mysqli_store_result` or `mysqli_query` functions.

Examples

Example 22.184. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($result = $mysqli->query($query)) {

    /* seek to row no. 400 */
    $result->data_seek(399);

    /* fetch row */
```

```
$row = $result->fetch_row();

printf ("City: %s  Countrycode: %s\n", $row[0], $row[1]);

/* free result set*/
$result->close();
}

/* close connection */
mysqli->close();
?>
```

Example 22.185. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER BY Name";

if ($result = mysqli_query($link, $query)) {

    /* seek to row no. 400 */
    mysqli_data_seek($result, 399);

    /* fetch row */
    $row = mysqli_fetch_row($result);

    printf ("City: %s  Countrycode: %s\n", $row[0], $row[1]);

    /* free result set*/
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
City: Benin City  Countrycode: NGA
```

See Also

[mysqli_store_result](#)
[mysqli_fetch_row](#)
[mysqli_fetch_array](#)
[mysqli_fetch_assoc](#)
[mysqli_fetch_object](#)
[mysqli_query](#)

[mysqli_num_rows](#)

22.9.3.11.3. [mysqli_result::fetch_all](#), [mysqli_fetch_all](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_result::fetch_all](#)

[mysqli_fetch_all](#)

Fetches all result rows as an associative array, a numeric array, or both

Description

Object oriented style

```
mixed mysqli_result::fetch_all(  
    int resulttype  
    = =MYSQLI_NUM);
```

Procedural style

```
mixed mysqli_fetch_all(  
    mysqli_result result,  
    int resulttype  
    = =MYSQLI_NUM);
```

[mysqli_fetch_all](#) fetches all result rows and returns the result set as an associative array, a numeric array, or both.

Parameters

result

Procedural style only: A result set identifier returned by [mysqli_query](#), [mysqli_store_result](#) or [mysqli_use_result](#).

resulttype

This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants [MYSQLI_ASSOC](#) , [MYSQLI_NUM](#) , or [MYSQLI_BOTH](#) .

Return Values

Returns an array of associative or numeric arrays holding result rows.

MySQL Native Driver Only

Available only with [mysqlnd](#).

As [mysqli_fetch_all](#) returns all the rows as an array in a single step, it may consume more memory than some similar functions such as [mysqli_fetch_array](#), which only returns one row at a time from the result set. Further, if you need to iterate over the result set, you will need a looping construct that will further impact performance. For these reasons [mysqli_fetch_all](#) should only be used in those situations where the fetched result set will be sent to another layer for processing.

See Also

[mysqli_fetch_array](#)

`mysqli_query`

22.9.3.11.4. `mysqli_result::fetch_array`, `mysqli_fetch_array`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_result::fetch_array`

`mysqli_fetch_array`

Fetch a result row as an associative, a numeric array, or both

Description

Object oriented style

```
mixed mysqli_result::fetch_array(  
    int resulttype  
    = =MYSQLI_BOTH);
```

Procedural style

```
mixed mysqli_fetch_array(  
    mysqli_result result,  
    int resulttype  
    = =MYSQLI_BOTH);
```

Returns an array that corresponds to the fetched row or `NULL` if there are no more rows for the resultset represented by the `result` parameter.

`mysqli_fetch_array` is an extended version of the `mysqli_fetch_row` function. In addition to storing the data in the numeric indices of the result array, the `mysqli_fetch_array` function can also store the data in associative indices, using the field names of the result set as keys.



Note

Field names returned by this function are *case-sensitive*.



Note

This function sets `NULL` fields to the PHP `NULL` value.

If two or more columns of the result have the same field names, the last column will take precedence and overwrite the earlier data. In order to access multiple columns with the same name, the numerically indexed version of the row must be used.

Parameters

`result`

Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

`resulttype`

This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants `MYSQLI_ASSOC`, `MYSQLI_NUM`, or `MYSQLI_BOTH`.

By using the `MYSQLI_ASSOC` constant this function will behave identically to the `mysqli_fetch_assoc`, while `MYSQLI_NUM` will

behave identically to the `mysqli_fetch_row` function. The final option `MYSQLI_BOTH` will create a single array with the attributes of both.

Return Values

Returns an array of strings that corresponds to the fetched row or `NULL` if there are no more rows in resultset.

Examples

Example 22.186. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID LIMIT 3";
$result = $mysqli->query($query);

/* numeric array */
$row = $result->fetch_array(MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);

/* associative array */
$row = $result->fetch_array(MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);

/* associative and numeric array */
$row = $result->fetch_array(MYSQLI_BOTH);
printf ("%s (%s)\n", $row[0], $row["CountryCode"]);

/* free result set */
$result->free();

/* close connection */
$mysqli->close();
?>
```

Example 22.187. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID LIMIT 3";
$result = mysqli_query($link, $query);

/* numeric array */
$row = mysqli_fetch_array($result, MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);
```

```
/* associative array */
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);

/* associative and numeric array */
$row = mysqli_fetch_array($result, MYSQLI_BOTH);
printf ("%s (%s)\n", $row[0], $row["CountryCode"]);

/* free result set */
mysqli_free_result($result);

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Kabul (AFG)
Qandahar (AFG)
Herat (AFG)
```

See Also

[mysqli_fetch_assoc](#)
[mysqli_fetch_row](#)
[mysqli_fetch_object](#)
[mysqli_query](#)
[mysqli_data_seek](#)

22.9.3.11.5. [mysqli_result::fetch_assoc](#), [mysqli_fetch_assoc](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_result::fetch_assoc](#)

[mysqli_fetch_assoc](#)

Fetch a result row as an associative array

Description

Object oriented style

```
array mysqli_result::fetch_assoc();
```

Procedural style

```
array mysqli_fetch_assoc(
    mysqli_result result);
```

Returns an associative array that corresponds to the fetched row or [NULL](#) if there are no more rows.



Note

Field names returned by this function are *case-sensitive*.

**Note**

This function sets NULL fields to the PHP `NULL` value.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns an associative array of strings representing the fetched row in the result set, where each key in the array represents the name of one of the result set's columns or `NULL` if there are no more rows in resultset.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you either need to access the result with numeric indices by using `mysqli_fetch_row` or add alias names.

Examples**Example 22.188. Object oriented style**

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";

if ($result = $mysqli->query($query)) {

    /* fetch associative array */
    while ($row = $result->fetch_assoc()) {
        printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
    }

    /* free result set */
    $result->free();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.189. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
```

```

}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";

if ($result = mysqli_query($link, $query)) {

    /* fetch associative array */
    while ($row = mysqli_fetch_assoc($result)) {
        printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
    }

    /* free result set */
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```

Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)

```

Example 22.190. A `mysqli_result` example comparing `iterator` usage

```

<?php
$c = mysqli_connect('127.0.0.1','user', 'pass');

// Using iterators (support was added with PHP 5.4)
foreach ( $c->query('SELECT user,host FROM mysql.user') as $row ) {
    printf("%s'%s'\n", $row['user'], $row['host']);
}

echo "\n=====\n";

// Not using iterators
$result = $c->query('SELECT user,host FROM mysql.user');
while ($row = $result->fetch_assoc()) {
    printf("%s'%s'\n", $row['user'], $row['host']);
}

?>

```

The above example will output something similar to:

```

'root'@'192.168.1.1'
'root'@'127.0.0.1'
'dude'@'localhost'
'lebowski'@'localhost'

=====

```

```
'root'@'192.168.1.1'
'root'@'127.0.0.1'
'dude'@'localhost'
'lebowski'@'localhost'
```

See Also

```
mysqli_fetch_array
mysqli_fetch_row
mysqli_fetch_object
mysqli_query
mysqli_data_seek
```

22.9.3.11.6. `mysqli_result::fetch_field_direct`, `mysqli_fetch_field_direct`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_result::fetch_field_direct`

```
mysqli_fetch_field_direct
```

Fetch meta-data for a single field

Description

Object oriented style

```
object mysqli_result::fetch_field_direct(
    int fieldnr);
```

Procedural style

```
object mysqli_fetch_field_direct(
    mysqli_result result,
    int fieldnr);
```

Returns an object which contains field definition information from the specified result set.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

fieldnr The field number. This value must be in the range from 0 to `number of fields - 1`.

Return Values

Returns an object which contains field definition information or `FALSE` if no field information for specified `fieldnr` is available.

Table 22.51. Object attributes

Attribute	Description
name	The name of the column

Attribute	Description
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
def	The default value for this field, represented as a string
max_length	The maximum width of the field for the result set.
length	The width of the field, as specified in the table definition.
charsetnr	The character set number for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for integer fields)

Examples

Example 22.191. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Name LIMIT 5";

if ($result = $mysqli->query($query)) {

    /* Get field information for column 'SurfaceArea' */
    $finfo = $result->fetch_field_direct(1);

    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len:  %d\n", $finfo->max_length);
    printf("Flags:     %d\n", $finfo->flags);
    printf("Type:      %d\n", $finfo->type);

    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.192. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Name LIMIT 5";

if ($result = mysqli_query($link, $query)) {

    /* Get field information for column 'SurfaceArea' */
    $finfo = mysqli_fetch_field_direct($result, 1);

    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len:  %d\n", $finfo->max_length);
    printf("Flags:    %d\n", $finfo->flags);
    printf("Type:     %d\n", $finfo->type);

    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:      SurfaceArea
Table:     Country
max. Len:  10
Flags:    32769
Type:     4
```

See Also

[mysqli_num_fields](#)
[mysqli_fetch_field](#)
[mysqli_fetch_fields](#)

22.9.3.11.7. [mysqli_result::fetch_field](#), [mysqli_fetch_field](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_result::fetch_field](#)

[mysqli_fetch_field](#)

Returns the next field in the result set

Description

Object oriented style

```
object mysqli_result::fetch_field();
```

Procedural style

```
object mysqli_fetch_field(
    mysqli_result result);
```

Returns the definition of one column of a result set as an object. Call this function repeatedly to retrieve information about all columns in the result set.

Parameters

result Procedural style only: A result set identifier returned by *mysqli_query*, *mysqli_store_result* or *mysqli_use_result*.

Return Values

Returns an object which contains field definition information or `FALSE` if no field information is available.

Table 22.52. Object properties

Property	Description
name	The name of the column
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
def	Reserved for default value, currently always ""
db	Database (since PHP 5.3.6)
catalog	The catalog name, always "def" (since PHP 5.3.6)
max_length	The maximum width of the field for the result set.
length	The width of the field, as specified in the table definition.
charsetnr	The character set number for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for integer fields)

Examples

Example 22.193. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";

if ($result = $mysqli->query($query)) {

    /* Get field information for all columns */
```

```

while ($finfo = $result->fetch_field()) {

    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len:  %d\n", $finfo->max_length);
    printf("Flags:    %d\n", $finfo->flags);
    printf("Type:     %d\n\n", $finfo->type);
}
$result->close();
}

/* close connection */
mysqli->close();
?>

```

Example 22.194. Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";

if ($result = mysqli_query($link, $query)) {

    /* Get field information for all fields */
    while ($finfo = mysqli_fetch_field($result)) {

        printf("Name:      %s\n", $finfo->name);
        printf("Table:     %s\n", $finfo->table);
        printf("max. Len:  %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```

Name:      Name
Table:     Country
max. Len:  11
Flags:     1
Type:      254

Name:      SurfaceArea
Table:     Country
max. Len:  10
Flags:     32769
Type:      4

```

See Also

[mysqli_num_fields](#)
[mysqli_fetch_field_direct](#)
[mysqli_fetch_fields](#)
[mysqli_field_seek](#)

22.9.3.11.8. [mysqli_result::fetch_fields](#), [mysqli_fetch_fields](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_result::fetch_fields](#)

[mysqli_fetch_fields](#)

Returns an array of objects representing the fields in a result set

Description

Object oriented style

```
array mysqli_result::fetch_fields();
```

Procedural style

```
array mysqli_fetch_fields(
    mysqli_result result);
```

This function serves an identical purpose to the [mysqli_fetch_field](#) function with the single difference that, instead of returning one object at a time for each field, the columns are returned as an array of objects.

Parameters

result Procedural style only: A result set identifier returned by [mysqli_query](#), [mysqli_store_result](#) or [mysqli_use_result](#).

Return Values

Returns an array of objects which contains field definition information or [FALSE](#) if no field information is available.

Table 22.53. Object properties

Property	Description
name	The name of the column
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
max_length	The maximum width of the field for the result set.

Property	Description
length	The width of the field, as specified in the table definition.
charsetnr	The character set number for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for integer fields)

Examples

Example 22.195. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";

if ($result = $mysqli->query($query)) {

    /* Get field information for all columns */
    $finfo = $result->fetch_fields();

    foreach ($finfo as $val) {
        printf("Name:      %s\n", $val->name);
        printf("Table:      %s\n", $val->table);
        printf("max. Len:  %d\n", $val->max_length);
        printf("Flags:      %d\n", $val->flags);
        printf("Type:       %d\n\n", $val->type);
    }
    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.196. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";

if ($result = mysqli_query($link, $query)) {
```

```
/* Get field information for all columns */
$finfo = mysqli_fetch_fields($result);

foreach ($finfo as $val) {
    printf("Name:      %s\n", $val->name);
    printf("Table:     %s\n", $val->table);
    printf("max. Len:  %d\n", $val->max_length);
    printf("Flags:     %d\n", $val->flags);
    printf("Type:      %d\n\n", $val->type);
}
mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:      Name
Table:     Country
max. Len:  11
Flags:     1
Type:      254

Name:      SurfaceArea
Table:     Country
max. Len:  10
Flags:     32769
Type:      4
```

See Also

[mysqli_num_fields](#)
[mysqli_fetch_field_direct](#)
[mysqli_fetch_field](#)

22.9.3.11.9. [mysqli_result::fetch_object](#), [mysqli_fetch_object](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_result::fetch_object](#)

[mysqli_fetch_object](#)

Returns the current row of a result set as an object

Description

Object oriented style

```
object mysqli_result::fetch_object(
    string class_name,
    array params);
```

Procedural style

```
object mysqli_fetch_object(
    mysqli_result result,
    string class_name,
    array params);
```

The `mysqli_fetch_object` will return the current row result set as an object where the attributes of the object represent the names of the fields found within the result set.

Note that `mysqli_fetch_object` sets the properties of the object before calling the object constructor.

Parameters

<code>result</code>	Procedural style only: A result set identifier returned by <code>mysqli_query</code> , <code>mysqli_store_result</code> or <code>mysqli_use_result</code> .
<code>class_name</code>	The name of the class to instantiate, set the properties of and return. If not specified, a <code>stdClass</code> object is returned.
<code>params</code>	An optional array of parameters to pass to the constructor for <code>class_name</code> objects.

Return Values

Returns an object with string properties that corresponds to the fetched row or `NULL` if there are no more rows in resultset.



Note

Field names returned by this function are *case-sensitive*.



Note

This function sets NULL fields to the PHP `NULL` value.

Changelog

Version	Description
5.0.0	Added the ability to return as a different object.

Examples

Example 22.197. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";

if ($result = $mysqli->query($query)) {
```

```
/* fetch object array */
while ($obj = $result->fetch_object()) {
    printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
}

/* free result set */
$result->close();
}

/* close connection */
mysqli->close();
?>
```

Example 22.198. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";

if ($result = mysqli_query($link, $query)) {

    /* fetch associative array */
    while ($obj = mysqli_fetch_object($result)) {
        printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
    }

    /* free result set */
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)
```

See Also

[mysqli_fetch_array](#)
[mysqli_fetch_assoc](#)
[mysqli_fetch_row](#)
[mysqli_query](#)

`mysqli_data_seek`

22.9.3.11.10. `mysqli_result::fetch_row`, `mysqli_fetch_row`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_result::fetch_row`

`mysqli_fetch_row`

Get a result row as an enumerated array

Description

Object oriented style

```
mixed mysqli_result::fetch_row();
```

Procedural style

```
mixed mysqli_fetch_row(  
    mysqli_result result);
```

Fetches one row of data from the result set and returns it as an enumerated array, where each column is stored in an array offset starting from 0 (zero). Each subsequent call to this function will return the next row within the result set, or `NULL` if there are no more rows.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

`mysqli_fetch_row` returns an array of strings that corresponds to the fetched row or `NULL` if there are no more rows in result set.



Note

This function sets NULL fields to the PHP `NULL` value.

Examples

Example 22.199. Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
  
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
```

```
if ($result = $mysqli->query($query)) {  
  
    /* fetch object array */  
    while ($row = $result->fetch_row()) {  
        printf ("%s (%s)\n", $row[0], $row[1]);  
    }  
  
    /* free result set */  
    $result->close();  
}  
  
/* close connection */  
$mysqli->close();  
?>
```

Example 22.200. Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
  
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";  
  
if ($result = mysqli_query($link, $query)) {  
  
    /* fetch associative array */  
    while ($row = mysqli_fetch_row($result)) {  
        printf ("%s (%s)\n", $row[0], $row[1]);  
    }  
  
    /* free result set */  
    mysqli_free_result($result);  
}  
  
/* close connection */  
mysqli_close($link);  
?>
```

The above examples will output:

```
Pueblo (USA)  
Arvada (USA)  
Cape Coral (USA)  
Green Bay (USA)  
Santa Clara (USA)
```

See Also

[mysqli_fetch_array](#)
[mysqli_fetch_assoc](#)
[mysqli_fetch_object](#)

```
mysqli_query  
mysqli_data_seek
```

22.9.3.11.11. `mysqli_result::$field_count`, `mysqli_num_fields`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_result::$field_count`

```
mysqli_num_fields
```

Get the number of fields in a result

Description

Object oriented style

```
int  
mysqli_result->field_count ;
```

Procedural style

```
int mysqli_num_fields(  
mysqli_result result);
```

Returns the number of fields from specified result set.

Parameters

result

Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

The number of fields from a result set.

Examples

Example 22.201. Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
  
if ($result = $mysqli->query("SELECT * FROM City ORDER BY ID LIMIT 1")) {  
  
    /* determine number of fields in result set */  
    $field_cnt = $result->field_count;  
  
    printf("Result set has %d fields.\n", $field_cnt);  
  
    /* close result set */  
    $result->close();  
}
```

```
/* close connection */
mysqli->close();
?>
```

Example 22.202. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if ($result = mysqli_query($link, "SELECT * FROM City ORDER BY ID LIMIT 1")) {

    /* determine number of fields in result set */
    $field_cnt = mysqli_num_fields($result);

    printf("Result set has %d fields.\n", $field_cnt);

    /* close result set */
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Result set has 5 fields.
```

See Also

[mysqli_fetch_field](#)

22.9.3.11.12. [mysqli_result::field_seek](#), [mysqli_field_seek](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_result::field_seek](#)

[mysqli_field_seek](#)

Set result pointer to a specified field offset

Description

Object oriented style

```
bool mysqli_result::field_seek(
```

```
int fieldnr);
```

Procedural style

```
bool mysqli_field_seek(
    mysqli_result result,
    int fieldnr);
```

Sets the field cursor to the given offset. The next call to `mysqli_fetch_field` will retrieve the field definition of the column associated with that offset.



Note

To seek to the beginning of a row, pass an offset value of zero.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

fieldnr The field number. This value must be in the range from 0 to `number of fields - 1`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.203. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";

if ($result = $mysqli->query($query)) {

    /* Get field information for 2nd column */
    $result->field_seek(1);
    $finfo = $result->fetch_field();

    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len:  %d\n", $finfo->max_length);
    printf("Flags:     %d\n", $finfo->flags);
    printf("Type:      %d\n\n", $finfo->type);

    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.204. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";

if ($result = mysqli_query($link, $query)) {

    /* Get field information for 2nd column */
    mysqli_field_seek($result, 1);
    $finfo = mysqli_fetch_field($result);

    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len:  %d\n", $finfo->max_length);
    printf("Flags:     %d\n", $finfo->flags);
    printf("Type:      %d\n\n", $finfo->type);

    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:      SurfaceArea
Table:     Country
max. Len:  10
Flags:     32769
Type:      4
```

See Also

[mysqli_fetch_field](#)

22.9.3.11.13. [mysqli_result::free](#), [mysqli_free_result](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_result::free](#)

[mysqli_free_result](#)

Frees the memory associated with a result

Description

Object oriented style

```
void mysqli_result::free();
```

```
void mysqli_result::close();
```

```
void mysqli_result::free_result();
```

Procedural style

```
void mysqli_free_result(  
    mysqli_result result);
```

Frees the memory associated with the result.



Note

You should always free your result with `mysqli_free_result`, when your result object is not needed anymore.

Parameters

result

Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

No value is returned.

See Also

`mysqli_query`
`mysqli_stmt_store_result`
`mysqli_store_result`
`mysqli_use_result`

22.9.3.11.14. `mysqli_result::$lengths`, `mysqli_fetch_lengths`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_result::$lengths`

`mysqli_fetch_lengths`

Returns the lengths of the columns of the current row in the result set

Description

Object oriented style

```
array  
    mysqli_result->lengths ;
```

Procedural style

```
array mysqli_fetch_lengths(  
    mysqli_result result);
```

```
mysqli_result result);
```

The [mysqli_fetch_lengths](#) function returns an array containing the lengths of every column of the current row within the result set.

Parameters

result Procedural style only: A result set identifier returned by [mysqli_query](#), [mysqli_store_result](#) or [mysqli_use_result](#).

Return Values

An array of integers representing the size of each column (not including any terminating null characters). [FALSE](#) if an error occurred.

[mysqli_fetch_lengths](#) is valid only for the current row of the result set. It returns [FALSE](#) if you call it before calling [mysqli_fetch_row/array/object](#) or after retrieving all rows in the result.

Examples

Example 22.205. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT * from Country ORDER BY Code LIMIT 1";

if ($result = $mysqli->query($query)) {

    $row = $result->fetch_row();

    /* display column lengths */
    foreach ($result->lengths as $i => $val) {
        printf("Field %2d has Length %2d\n", $i+1, $val);
    }
    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.206. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
```

```
$query = "SELECT * from Country ORDER BY Code LIMIT 1";

if ($result = mysqli_query($link, $query)) {

    $row = mysqli_fetch_row($result);

    /* display column lengths */
    foreach (mysqli_fetch_lengths($result) as $i => $val) {
        printf("Field %2d has Length %2d\n", $i+1, $val);
    }
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Field 1 has Length 3
Field 2 has Length 5
Field 3 has Length 13
Field 4 has Length 9
Field 5 has Length 6
Field 6 has Length 1
Field 7 has Length 6
Field 8 has Length 4
Field 9 has Length 6
Field 10 has Length 6
Field 11 has Length 5
Field 12 has Length 44
Field 13 has Length 7
Field 14 has Length 3
Field 15 has Length 2
```

22.9.3.11.15. `mysqli_result::$num_rows`, `mysqli_num_rows`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_result::$num_rows`

`mysqli_num_rows`

Gets the number of rows in a result

Description

Object oriented style

```
int
mysqli_result->num_rows ;
```

Procedural style

```
int mysqli_num_rows(
    mysqli_result result);
```

Returns the number of rows in the result set.

The behaviour of `mysqli_num_rows` depends on whether buffered or unbuffered result sets are being used. For unbuffered result sets, `mysqli_num_rows` will not return the correct number of rows until all the rows in the result have been retrieved.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns number of rows in the result set.



Note

If the number of rows is greater than `MAXINT`, the number will be returned as a string.

Examples

Example 22.207. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if ($result = $mysqli->query("SELECT Code, Name FROM Country ORDER BY Name")) {

    /* determine number of rows result set */
    $row_cnt = $result->num_rows;

    printf("Result set has %d rows.\n", $row_cnt);

    /* close result set */
    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 22.208. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
```

```
if ($result = mysqli_query($link, "SELECT Code, Name FROM Country ORDER BY Name")) {  
  
    /* determine number of rows result set */  
    $row_cnt = mysqli_num_rows($result);  
  
    printf("Result set has %d rows.\n", $row_cnt);  
  
    /* close result set */  
    mysqli_free_result($result);  
}  
  
/* close connection */  
mysqli_close($link);  
?>
```

The above examples will output:

```
Result set has 239 rows.
```

See Also

[mysqli_affected_rows](#)
[mysqli_store_result](#)
[mysqli_use_result](#)
[mysqli_query](#)

22.9.3.12. The `mysqli_driver` class ([mysqli_driver](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

MySQLi Driver.

```
mysqli_driver {  
    mysqli_driver  
  
    Properties  
  
    public readonly string  
        client_info ;  
  
    public readonly string  
        client_version ;  
  
    public readonly string  
        driver_version ;  
  
    public readonly string  
        embedded ;  
  
    public bool  
        reconnect ;  
  
    public int  
        report_mode ;  
  
    Methods
```

```

void mysqli_driver::embedded_server_end();

bool mysqli_driver::embedded_server_start(
    bool start,
    array arguments,
    array groups);
}

```

<code>client_info</code>	The Client API header version
<code>client_version</code>	The Client version
<code>driver_version</code>	The MySQLi Driver version
<code>embedded</code>	Whether MySQLi Embedded support is enabled
<code>reconnect</code>	Allow or prevent reconnect (see the <code>mysqli.reconnect</code> INI directive)
<code>report_mode</code>	Set to <code>MYSQLI_REPORT_OFF</code> , <code>MYSQLI_REPORT_ALL</code> or any combination of <code>MYSQLI_REPORT_STRICT</code> (throw Exceptions for errors), <code>MYSQLI_REPORT_ERROR</code> (report errors) and <code>MYSQLI_REPORT_INDEX</code> (errors regarding indexes). See also mysqli_report .

22.9.3.12.1. `mysqli_driver::embedded_server_end`, `mysqli_embedded_server_end`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_driver::embedded_server_end`

`mysqli_embedded_server_end`

Stop embedded server

Description

Object oriented style

```
void mysqli_driver::embedded_server_end();
```

Procedural style

```
void mysqli_embedded_server_end();
```



Warning

This function is currently not documented; only its argument list is available.

22.9.3.12.2. `mysqli_driver::embedded_server_start`, `mysqli_embedded_server_start`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_driver::embedded_server_start`

`mysqli_embedded_server_start`

Initialize and start embedded server

Description

Object oriented style

```
bool mysqli_driver::embedded_server_start(
    bool start,
    array arguments,
    array groups);
```

Procedural style

```
bool mysqli_embedded_server_start(
    bool start,
    array arguments,
    array groups);
```

**Warning**

This function is currently not documented; only its argument list is available.

22.9.3.12.3. `mysqli_driver::$report_mode`, `mysqli_report`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_driver::$report_mode`

`mysqli_report`

Enables or disables internal report functions

Description

Object oriented style

```
int
mysqli_driver->report_mode ;
```

Procedural style

```
bool mysqli_report(
    int flags);
```

A function helpful in improving queries during code development and testing. Depending on the flags, it reports errors from `mysqli` function calls or queries that don't use an index (or use a bad index).

Parameters*flags***Table 22.54. Supported flags**

Name	Description
<code>MYSQLI_REPORT_OFF</code>	Turns reporting off
<code>MYSQLI_REPORT_ERROR</code>	Report errors from <code>mysqli</code> function calls
<code>MYSQLI_REPORT_STRICT</code>	Throw <code>mysqli_sql_exception</code> for errors instead of warnings
<code>MYSQLI_REPORT_INDEX</code>	Report if no index or bad index was used in a query

Name	Description
MYSQLI_REPORT_ALL	Set all options (report all)

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Changelog

Version	Description
5.3.4	Changing the reporting mode is now be per-request, rather than per-process.
5.2.15	Changing the reporting mode is now be per-request, rather than per-process.

Examples

Example 22.209. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* activate reporting */
$driver = new mysqli_driver();
$driver->report_mode = MYSQLI_REPORT_ALL;

try {

    /* this query should report an error */
    $result = $mysqli->query("SELECT Name FROM Nonexistingtable WHERE population > 50000");

    /* this query should report a bad index */
    $result = $mysqli->query("SELECT Name FROM City WHERE population > 50000");

    $result->close();

    $mysqli->close();

} catch (mysqli_sql_exception $e) {

    echo $e->__toString();
}
?>
```

Example 22.210. Procedural style

```
<?php
/* activate reporting */
```

```
mysqli_report(MYSQLI_REPORT_ALL);

$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* this query should report an error */
$result = mysqli_query("SELECT Name FROM Nonexistingtable WHERE population > 50000");

/* this query should report a bad index */
$result = mysqli_query("SELECT Name FROM City WHERE population > 50000");

mysqli_free_result($result);

mysqli_close($link);
?>
```

See Also

[mysqli_debug](#)
[mysqli_dump_debug_info](#)
[mysqli_sql_exception](#)
[set_exception_handler](#)
[error_reporting](#)

22.9.3.13. The `mysqli_warning` class ([mysqli_warning](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

Represents a MySQL warning.

```
mysqli_warning {
    mysqli_warning

    Properties

    public
        message ;

    public
        sqlstate ;

    public
        errno ;

    Methods

    public mysqli_warning::__construct();

    public void mysqli_warning::next();
}
```

[message](#)

Message string

<code>sqlstate</code>	SQL state
<code>errno</code>	Error number

22.9.3.13.1. `mysqli_warning::__construct`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_warning::__construct`

The `__construct` purpose

Description

```
public mysqli_warning::__construct();
```



Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

22.9.3.13.2. `mysqli_warning::next`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_warning::next`

The `next` purpose

Description

```
public void mysqli_warning::next();
```



Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

22.9.3.14. The `mysqli_sql_exception` class (`mysqli_sql_exception`)

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqli` exception handling class.

```
mysqli_sql_exception {
    mysqli_sql_exception extends RuntimeException

    Properties

    protected
        code ;

    protected
        sqlstate ;
}
```

<code>message</code>	The error message.
<code>file</code>	The file with the error.
<code>line</code>	The line with the error.
<code>code</code>	The code causing the error.
<code>sqlstate</code>	The sql state with the error.

22.9.3.15. Aliases and deprecated Mysqli Functions

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.3.15.1. `mysqli_bind_param`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_bind_param`

Alias for `mysqli_stmt_bind_param`

Description

This function is an alias of `mysqli_stmt_bind_param`.



Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_bind_param`

22.9.3.15.2. `mysqli_bind_result`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_bind_result`

Alias for `mysqli_stmt_bind_result`

Description

This function is an alias of `mysqli_stmt_bind_result`.



Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_bind_result`

22.9.3.15.3. `mysqli_client_encoding`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_client_encoding`

Alias of `mysqli_character_set_name`

Description

This function is an alias of `mysqli_character_set_name`.



Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_real_escape_string`

22.9.3.15.4. `mysqli_connect`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_connect`

Alias of `mysqli::__construct`

Description

This function is an alias of: `mysqli::__construct`

22.9.3.15.5. `mysqli::disable_reads_from_master`, `mysqli_disable_reads_from_master`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli::disable_reads_from_master`

`mysqli_disable_reads_from_master`

Disable reads from master

Description

Object oriented style

```
void mysqli::disable_reads_from_master();
```

Procedural style

```
bool mysqli_disable_reads_from_master(  
    mysqli link);
```



Warning

This function is currently not documented; only its argument list is available.



Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.15.6. [mysqli_disable_rpl_parse](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_disable_rpl_parse](#)

Disable RPL parse

Description

```
bool mysqli_disable_rpl_parse(  
    mysqli link);
```



Warning

This function is currently not documented; only its argument list is available.



Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.15.7. [mysqli_enable_reads_from_master](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_enable_reads_from_master](#)

Enable reads from master

Description

```
bool mysqli_enable_reads_from_master(  
    mysqli link);
```



Warning

This function is currently not documented; only its argument list is available.

**Warning**

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.15.8. [mysqli_enable_rpl_parse](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_enable_rpl_parse](#)

Enable RPL parse

Description

```
bool mysqli_enable_rpl_parse(  
    mysqli link);
```

**Warning**

This function is currently not documented; only its argument list is available.

**Warning**

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.15.9. [mysqli_escape_string](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_escape_string](#)

Alias of [mysqli_real_escape_string](#)

Description

This function is an alias of: [mysqli_real_escape_string](#).

22.9.3.15.10. [mysqli_execute](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_execute](#)

Alias for [mysqli_stmt_execute](#)

Description

This function is an alias of [mysqli_stmt_execute](#).

Notes**Note**

[mysqli_execute](#) is deprecated and will be removed.

See Also

`mysqli_stmt_execute`

22.9.3.15.11. `mysqli_fetch`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_fetch`

Alias for `mysqli_stmt_fetch`

Description

This function is an alias of `mysqli_stmt_fetch`.

**Warning**

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_fetch`

22.9.3.15.12. `mysqli_get_cache_stats`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_get_cache_stats`

Returns client Zval cache statistics

Description

```
array mysqli_get_cache_stats();
```

**Warning**

This function is currently not documented; only its argument list is available.

Returns client Zval cache statistics. Available only with `mysqlnd`.

Parameters**Return Values**

Returns an array with client Zval cache stats if success, `FALSE` otherwise.

Examples**Example 22.211. A `mysqli_get_cache_stats` example**

```
<?php
$link = mysqli_connect();
print_r(mysqli_get_cache_stats());
?>
```

The above example will output something similar to:

```
Array
(
    [bytes_sent] => 43
    [bytes_received] => 80
    [packets_sent] => 1
    [packets_received] => 2
    [protocol_overhead_in] => 8
    [protocol_overhead_out] => 4
    [bytes_received_ok_packet] => 11
    [bytes_received_eof_packet] => 0
    [bytes_received_rset_header_packet] => 0
    [bytes_received_rset_field_meta_packet] => 0
    [bytes_received_rset_row_packet] => 0
    [bytes_received_prepare_response_packet] => 0
    [bytes_received_change_user_packet] => 0
    [packets_sent_command] => 0
    [packets_received_ok] => 1
    [packets_received_eof] => 0
    [packets_received_rset_header] => 0
    [packets_received_rset_field_meta] => 0
    [packets_received_rset_row] => 0
    [packets_received_prepare_response] => 0
    [packets_received_change_user] => 0
    [result_set_queries] => 0
    [non_result_set_queries] => 0
    [no_index_used] => 0
    [bad_index_used] => 0
    [slow_queries] => 0
    [buffered_sets] => 0
    [unbuffered_sets] => 0
    [ps_buffered_sets] => 0
    [ps_unbuffered_sets] => 0
    [flushed_normal_sets] => 0
    [flushed_ps_sets] => 0
    [ps_prepared_never_executed] => 0
    [ps_prepared_once_executed] => 0
    [rows_fetched_from_server_normal] => 0
    [rows_fetched_from_server_ps] => 0
    [rows_buffered_from_client_normal] => 0
    [rows_buffered_from_client_ps] => 0
    [rows_fetched_from_client_normal_buffered] => 0
    [rows_fetched_from_client_normal_unbuffered] => 0
    [rows_fetched_from_client_ps_buffered] => 0
    [rows_fetched_from_client_ps_unbuffered] => 0
    [rows_fetched_from_client_ps_cursor] => 0
    [rows_skipped_normal] => 0
    [rows_skipped_ps] => 0
    [copy_on_write_saved] => 0
    [copy_on_write_performed] => 0
    [command_buffer_too_small] => 0
    [connect_success] => 1
    [connect_failure] => 0
    [connection_reused] => 0
    [reconnect] => 0
    [pconnect_success] => 0
    [active_connections] => 1
    [active_persistent_connections] => 0
    [explicit_close] => 0
    [implicit_close] => 0
    [disconnect_close] => 0
    [in_middle_of_command_close] => 0
)
```

```
[explicit_free_result] => 0
[implicit_free_result] => 0
[explicit_stmt_close] => 0
[implicit_stmt_close] => 0
[mem_emalloc_count] => 0
[mem_emalloc_ammount] => 0
[mem_ecalloc_count] => 0
[mem_ecalloc_ammount] => 0
[mem_erealloc_count] => 0
[mem_erealloc_ammount] => 0
[mem_efree_count] => 0
[mem_malloc_count] => 0
[mem_malloc_ammount] => 0
[mem_calloc_count] => 0
[mem_calloc_ammount] => 0
[mem_realloc_count] => 0
[mem_realloc_ammount] => 0
[mem_free_count] => 0
[proto_text_fetched_null] => 0
[proto_text_fetched_bit] => 0
[proto_text_fetched_tinyint] => 0
[proto_text_fetched_short] => 0
[proto_text_fetched_int24] => 0
[proto_text_fetched_int] => 0
[proto_text_fetched_bigint] => 0
[proto_text_fetched_decimal] => 0
[proto_text_fetched_float] => 0
[proto_text_fetched_double] => 0
[proto_text_fetched_date] => 0
[proto_text_fetched_year] => 0
[proto_text_fetched_time] => 0
[proto_text_fetched_datetime] => 0
[proto_text_fetched_timestamp] => 0
[proto_text_fetched_string] => 0
[proto_text_fetched_blob] => 0
[proto_text_fetched_enum] => 0
[proto_text_fetched_set] => 0
[proto_text_fetched_geometry] => 0
[proto_text_fetched_other] => 0
[proto_binary_fetched_null] => 0
[proto_binary_fetched_bit] => 0
[proto_binary_fetched_tinyint] => 0
[proto_binary_fetched_short] => 0
[proto_binary_fetched_int24] => 0
[proto_binary_fetched_int] => 0
[proto_binary_fetched_bigint] => 0
[proto_binary_fetched_decimal] => 0
[proto_binary_fetched_float] => 0
[proto_binary_fetched_double] => 0
[proto_binary_fetched_date] => 0
[proto_binary_fetched_year] => 0
[proto_binary_fetched_time] => 0
[proto_binary_fetched_datetime] => 0
[proto_binary_fetched_timestamp] => 0
[proto_binary_fetched_string] => 0
[proto_binary_fetched_blob] => 0
[proto_binary_fetched_enum] => 0
[proto_binary_fetched_set] => 0
[proto_binary_fetched_geometry] => 0
[proto_binary_fetched_other] => 0
)
```

See Also

[Stats description](#)

22.9.3.15.13. [mysqli_get_metadata](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_get_metadata](#)

Alias for [mysqli_stmt_result_metadata](#)

Description

This function is an alias of [mysqli_stmt_result_metadata](#).



Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

[mysqli_stmt_result_metadata](#)

22.9.3.15.14. [mysqli_master_query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_master_query](#)

Enforce execution of a query on the master in a master/slave setup

Description

```
bool mysqli_master_query(  
    mysqli link,  
    string query);
```



Warning

This function is currently not documented; only its argument list is available.



Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.15.15. [mysqli_param_count](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqli_param_count](#)

Alias for [mysqli_stmt_param_count](#)

Description

This function is an alias of [mysqli_stmt_param_count](#).



Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_param_count`

22.9.3.15.16. `mysqli_report`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_report`

Alias of `mysqli_driver->report_mode`

Description

This function is an alias of: `mysqli_driver->report_mode`

22.9.3.15.17. `mysqli_rpl_parse_enabled`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_rpl_parse_enabled`

Check if RPL parse is enabled

Description

```
int mysqli_rpl_parse_enabled(
    mysqli link);
```



Warning

This function is currently not documented; only its argument list is available.



Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.15.18. `mysqli_rpl_probe`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_rpl_probe`

RPL probe

Description

```
bool mysqli_rpl_probe(
    mysqli link);
```



Warning

This function is currently not documented; only its argument list is available.



Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.15.19. `mysqli_send_long_data`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_send_long_data`

Alias for `mysqli_stmt_send_long_data`

Description

This function is an alias of `mysqli_stmt_send_long_data`.



Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_send_long_data`

22.9.3.15.20. `mysqli_set_opt`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_set_opt`

Alias of `mysqli_options`

Description

This function is an alias of `mysqli_options`.

22.9.3.15.21. `mysqli_slave_query`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqli_slave_query`

Force execution of a query on a slave in a master/slave setup

Description

```
bool mysqli_slave_query(  
    mysqli link,  
    string query);
```



Warning

This function is currently not documented; only its argument list is available.



Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

22.9.3.16. Changelog

Copyright 1997-2012 the PHP Documentation Group. [3186]

The following changes have been made to classes/functions/methods of this extension.

22.9.4. MySQL Functions (PDO_MYSQL) (MySQL (PDO))

Copyright 1997-2012 the PHP Documentation Group. [3186]

PDO_MYSQL is a driver that implements the [PHP Data Objects \(PDO\) interface](#) to enable access from PHP to MySQL 3.x, 4.x and 5.x databases.

PDO_MYSQL will take advantage of native prepared statement support present in MySQL 4.1 and higher. If you're using an older version of the mysql client libraries, PDO will emulate them for you.



Warning

Beware: Some MySQL table types (storage engines) do not support transactions. When writing transactional database code using a table type that does not support transactions, MySQL will pretend that a transaction was initiated successfully. In addition, any DDL queries issued will implicitly commit any pending transactions.

Use `--with-pdo-mysql[=DIR]` to install the PDO MySQL extension, where the optional `[=DIR]` is the MySQL base install directory. If `mysqlnd` is passed as `[=DIR]`, then the MySQL native driver will be used.

Optionally, the `--with-mysql-sock[=DIR]` sets to location to the MySQL unix socket pointer for all MySQL extensions, including PDO_MYSQL. If unspecified, the default locations are searched.

Optionally, the `--with-zlib-dir[=DIR]` is used to set the path to the libz install prefix.

```
$ ./configure --with-pdo-mysql --with-mysql-sock=/var/mysql/mysql.sock
```

SSL support is enabled using the appropriate [PDO_MySQL constants \[3498\]](#), which is equivalent to calling the [MySQL C API function `mysql_ssl_set\(\)`](#). Also, SSL cannot be enabled with `PDO::setAttribute` because the connection already exists. See also the MySQL documentation about [connecting to MySQL with SSL](#).

Table 22.55. Changelog

Version	Description
5.4.0	MySQL client libraries 4.1 and below are no longer supported.
5.3.9	Added SSL support with <code>mysqlnd</code> and OpenSSL.
5.3.7	Added SSL support with <code>libmysql</code> and OpenSSL.

The constants below are defined by this driver, and will only be available when the extension has been either compiled into PHP or dynamically loaded at runtime. In addition, these driver-specific constants should only be used if you are using this driver. Using driver-specific attributes with another driver may result in unexpected behaviour. `PDO::getAttribute` may be used to obtain the `PDO_ATTR_DRIVER_NAME` attribute to check the driver, if your code can run against multiple drivers.

`PDO::MYSQL_ATTR_USE_BUFFERED_QUERY` (integer) If this attribute is set to `TRUE` on a `PDOStatement`, the MySQL driver will use the buffered versions of the MySQL API. If you're writing portable code, you should use `PDOStatement::fetchAll` instead.

Example 22.212. Forcing queries to be buffered in mysql

```
<?php
if ($db->getAttribute(PDO::ATTR_DRIVER_NAME) == 'mysql') {
    $stmt = $db->prepare('select * from foo',
```

```

        array(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY => true));
    } else {
        die("my application only works with mysql; I should use \${stmt->fetchA
    }
    ?>

```

- `PDO::MYSQL_ATTR_LOCAL_INFILE`** (integer) Enable `LOAD LOCAL INFILE`.
 Note, this constant can only be used in the `driver_options` array when constructing a new database handle.
- `PDO::MYSQL_ATTR_INIT_COMMAND`** (integer) Command to execute when connecting to the MySQL server. Will automatically be re-executed when reconnecting.
 Note, this constant can only be used in the `driver_options` array when constructing a new database handle.
- `PDO::MYSQL_ATTR_READ_DEFAULT_OPTIONS`** (integer) Read options from the named option file instead of from `my.cnf`. This option is not available if `mysqlnd` is used, because `mysqlnd` does not read the `mysql` configuration files.
- `PDO::MYSQL_ATTR_READ_DEFAULT_GROUP`** (integer) Read options from the named group from `my.cnf` or the file specified with `MYSQL_READ_DEFAULT_FILE`. This option is not available if `mysqlnd` is used, because `mysqlnd` does not read the `mysql` configuration files.
- `PDO::MYSQL_ATTR_MAX_BUFFER_SIZE`** (integer) Maximum buffer size. Defaults to 1 MiB. This constant is not supported when compiled against `mysqlnd`.
- `PDO::MYSQL_ATTR_DIRECT_QUERY`** (integer) Perform direct queries, don't use prepared statements.
- `PDO::MYSQL_ATTR_FOUND_ROWS`** (integer) Return the number of found (matched) rows, not the number of changed rows.
- `PDO::MYSQL_ATTR_IGNORE_SPACE`** (integer) Permit spaces after function names. Makes all functions names reserved words.
- `PDO::MYSQL_ATTR_COMPRESS`** (integer) Enable network communication compression. This is not supported when compiled against `mysqlnd`.
- `PDO::MYSQL_ATTR_SSL_CA`** (integer) The file path to the SSL certificate authority.
 This exists as of PHP 5.3.7.
- `PDO::MYSQL_ATTR_SSL_CAPATH`** (integer) The file path to the directory that contains the trusted SSL CA certificates, which are stored in PEM format.
 This exists as of PHP 5.3.7.
- `PDO::MYSQL_ATTR_SSL_CERT`** (integer) The file path to the SSL certificate.
 This exists as of PHP 5.3.7.
- `PDO::MYSQL_ATTR_CIPHERS`** (integer) A list of one or more permissible ciphers to use for SSL encryption, in a format understood by OpenSSL. For example: `DHE-RSA-AES256-SHA:AES128-SHA`

This exists as of PHP 5.3.7.

`PDO::MYSQL_ATTR_KEY`
(integer)

The file path to the SSL key.

This exists as of PHP 5.3.7.

The behaviour of these functions is affected by settings in `php.ini`.

Table 22.56. PDO_MYSQL Configuration Options

Name	Default	Changeable
<code>pdo_mysql.default_socket</code>	<code>"/tmp/mysql.sock"</code>	PHP_INI_SYSTEM
<code>pdo_mysql.debug</code>	NULL	PHP_INI_SYSTEM

For further details and definitions of the `PHP_INI_*` modes, see the <http://www.php.net/manual/en/configuration.changes.modes>.

Here's a short explanation of the configuration directives.

`pdo_mysql.default_socket` string Sets a Unix domain socket. This value can either be set at compile time if a domain socket is found at configure. This ini setting is Unix only.

`pdo_mysql.debug` boolean Enables debugging for PDO_MYSQL. This setting is only available when PDO_MYSQL is compiled against `mysqlnd` and in PDO debug mode.

22.9.4.1. PDO_MYSQL DSN

Copyright 1997-2012 the PHP Documentation Group. [3186]

- PDO_MYSQL DSN

Connecting to MySQL databases

Description

The PDO_MYSQL Data Source Name (DSN) is composed of the following elements:

DSN prefix The DSN prefix is `mysql:`.

`host` The hostname on which the database server resides.

`port` The port number where the database server is listening.

`dbname` The name of the database.

`unix_socket` The MySQL Unix socket (shouldn't be used with `host` or `port`).

`charset` The character set. See the [character set](#) concepts documentation for more information.

Prior to PHP 5.3.6, this element was silently ignored. The same behaviour can be partly replicated with the `PDO::MYSQL_ATTR_INIT_COMMAND` driver option, as the following example shows.

**Warning**

The method in the below example can only be used with character sets that share the same lower 7 bit representation as ASCII, such as ISO-8859-1 and UTF-8. Users using character sets that have different representations (such as UTF-16 or Big5) *must* use the [charset](#) option provided in PHP 5.3.6 and later versions.

Example 22.213. Setting the connection character set to UTF-8 prior to PHP 5.3.6

```
<?php
$dsn = 'mysql:host=localhost;dbname=testdb';
$username = 'username';
$password = 'password';
$options = array(
    PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8',
);

$dbh = new PDO($dsn, $username, $password, $options);
?>
```

Changelog

Version	Description
5.3.6	Prior to version 5.3.6, charset was ignored.

Examples**Example 22.214. PDO_MYSQL DSN examples**

The following example shows a PDO_MYSQL DSN for connecting to MySQL databases:

```
mysql:host=localhost;dbname=testdb
```

More complete examples:

```
mysql:host=localhost;port=3307;dbname=testdb
mysql:unix_socket=/tmp/mysql.sock;dbname=testdb
```

Notes**Unix only:**

When the host name is set to "localhost", then the connection to the server is made thru a domain socket. If PDO_MYSQL is compiled against libmysql then the location of the socket file is at libmysql's compiled in location. If PDO_MYSQL is compiled against mysqlnd a default socket can be set thru the [pdo_mysql.default_socket](#) setting.

22.9.5. MySQL Native Driver ([Mysqlnd](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

MySQL Native Driver is a replacement for the MySQL Client Library (libmysql). MySQL Native Driver is part of the official PHP sources as of PHP 5.3.0.

The MySQL database extensions MySQL extension, [mysqli](#) and PDO MYSQL all communicate with the MySQL server. In the past, this was done by the extension using the services provided by the MySQL Client Library. The extensions were compiled against the MySQL Client Library in order to use its client-server protocol.

With MySQL Native Driver there is now an alternative, as the MySQL database extensions can be compiled to use MySQL Native Driver instead of the MySQL Client Library.

MySQL Native Driver is written in C as a PHP extension.

22.9.5.1. Overview

Copyright 1997-2012 the PHP Documentation Group. [3186]

What it is not

Although MySQL Native Driver is written as a PHP extension, it is important to note that it does not provide a new API to the PHP programmer. The programmer APIs for MySQL database connectivity are provided by the MySQL extension, [mysqli](#) and PDO MYSQL. These extensions can now use the services of MySQL Native Driver to communicate with the MySQL Server. Therefore, you should not think of MySQL Native Driver as an API.

Why use it?

Using the MySQL Native Driver offers a number of advantages over using the MySQL Client Library.

The older MySQL Client Library was written by MySQL AB (now Oracle Corporation) and so was released under the MySQL license. This ultimately led to MySQL support being disabled by default in PHP. However, the MySQL Native Driver has been developed as part of the PHP project, and is therefore released under the PHP license. This removes licensing issues that have been problematic in the past.

Also, in the past, you needed to build the MySQL database extensions against a copy of the MySQL Client Library. This typically meant you needed to have MySQL installed on a machine where you were building the PHP source code. Also, when your PHP application was running, the MySQL database extensions would call down to the MySQL Client library file at run time, so the file needed to be installed on your system. With MySQL Native Driver that is no longer the case as it is included as part of the standard distribution. So you do not need MySQL installed in order to build PHP or run PHP database applications.

Because MySQL Native Driver is written as a PHP extension, it is tightly coupled to the workings of PHP. This leads to gains in efficiency, especially when it comes to memory usage, as the driver uses the PHP memory management system. It also supports the PHP memory limit. Using MySQL Native Driver leads to comparable or better performance than using MySQL Client Library, it always ensures the most efficient use of memory. One example of the memory efficiency is the fact that when using the MySQL Client Library, each row is stored in memory twice, whereas with the MySQL Native Driver each row is only stored once in memory.



Reporting memory usage

Because MySQL Native Driver uses the PHP memory management system, its memory usage can be tracked with [memory_get_usage](#). This is not possible with libmysql because it uses the C function `malloc()` instead.

Special features

MySQL Native Driver also provides some special features not available when the MySQL database extensions use MySQL Client Library. These special features are listed below:

- Improved persistent connections
- The special function `mysqli_fetch_all`
- Performance statistics calls: `mysqli_get_cache_stats`, `mysqli_get_client_stats`, `mysqli_get_connection_stats`

The performance statistics facility can prove to be very useful in identifying performance bottlenecks.

MySQL Native Driver also allows for persistent connections when used with the `mysqli` extension.

SSL Support

MySQL Native Driver has supported SSL since PHP version 5.3.3

Compressed Protocol Support

As of PHP 5.3.2 MySQL Native Driver supports the compressed client server protocol. MySQL Native Driver did not support this in 5.3.0 and 5.3.1. Extensions such as `ext/mysql`, `ext/mysqli`, that are configured to use MySQL Native Driver, can also take advantage of this feature. Note that `PDO_MYSQL` does *NOT* support compression when used together with `mysqlnd`.

Named Pipes Support

Named pipes support for Windows was added in PHP version 5.4.0.

22.9.5.2. Installation

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

Changelog

Table 22.57. Changelog

Version	Description
5.3.0	The MySQL Native Driver was added, with support for all MySQL extensions (i.e., <code>mysql</code> , <code>mysqli</code> and <code>PDO_MYSQL</code>). Passing in <code>mysqlnd</code> to the appropriate configure switch enables this support.
5.4.0	The MySQL Native Driver is now the default for all MySQL extensions (i.e., <code>mysql</code> , <code>mysqli</code> and <code>PDO_MYSQL</code>). Passing in <code>mysqlnd</code> to configure is now optional.
5.5.0	SHA-256 Authentication Plugin support was added

Installation on Unix

The MySQL database extensions must be configured to use the MySQL Client Library. In order to use the MySQL Native Driver, PHP needs to be built specifying that the MySQL database extensions are compiled

with MySQL Native Driver support. This is done through configuration options prior to building the PHP source code.

For example, to build the MySQL extension, [mysqli](#) and PDO MySQL using the MySQL Native Driver, the following command would be given:

```
./configure --with-mysql=mysqlnd \
--with-mysqli=mysqlnd \
--with-pdo-mysql=mysqlnd \
[other options]
```

Installation on Windows

In the official PHP Windows distributions from 5.3 onwards, MySQL Native Driver is enabled by default, so no additional configuration is required to use it. All MySQL database extensions will use MySQL Native Driver in this case.

SHA-256 Authentication Plugin support

The MySQL Native Driver requires the OpenSSL functionality of PHP to be loaded and enabled to connect to MySQL through accounts that use the MySQL SHA-256 Authentication Plugin. For example, PHP could be configured using:

```
./configure --with-mysql=mysqlnd \
--with-mysqli=mysqlnd \
--with-pdo-mysql=mysqlnd \
--with-openssl \
[other options]
```

22.9.5.3. Runtime Configuration

Copyright 1997-2012 the PHP Documentation Group. [3186]

The behaviour of these functions is affected by settings in [php.ini](#).

Table 22.58. MySQL Native Driver Configuration Options

Name	Default	Changeable	Changelog
mysqlnd.collect_statistics	"1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.collect_memory_statistics	"0"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.debug	"0"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.log_mask	0	PHP_INI_ALL	Available since PHP 5.3.0
mysqlnd.mempool_default_size	16000	PHP_INI_ALL	Available since PHP 5.3.3
mysqlnd.net_read_timeout	"31536000"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.net_cmd_buffer_size	528.0 - "2048", 5.3.1 - "4096"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.net_read_buffer_size	"32768"	PHP_INI_SYSTEM	Available since PHP 5.3.0.

Name	Default	Changeable	Changelog
<code>mysqlnd.sha256_server_public_key</code>	<code>public_key</code>	PHP_INI_PERDIR	Available since PHP 5.5.0.

For further details and definitions of the `PHP_INI_*` modes, see the <http://www.php.net/manual/en/configuration.changes.modes>.

Here's a short explanation of the configuration directives.

`mysqlnd.collect_statistics` Enables the collection of various client statistics which can be accessed through `mysqli_get_client_stats`, `mysqli_get_connection_stats`, `mysqli_get_cache_stats` and are shown in `mysqlnd` section of the output of the `phpinfo` function as well.

boolean

This configuration setting enables all [MySQL Native Driver statistics](#) except those relating to memory management.

`mysqlnd.collect_memory_statistics` Enable the collection of various memory statistics which can be accessed through `mysqli_get_client_stats`, `mysqli_get_connection_stats`, `mysqli_get_cache_stats` and are shown in `mysqlnd` section of the output of the `phpinfo` function as well.

boolean

This configuration setting enables the memory management statistics within the overall set of [MySQL Native Driver statistics](#).

`mysqlnd.debug` string

Records communication from all extensions using `mysqlnd` to the specified log file.

The format of the directive is `mysqlnd.debug = "option1[,parameter_option1][:option2[,parameter_option2]]"`.

The options for the format string are as follows:

- `A[,file]` - Appends trace output to specified file. Also ensures that data is written after each write. This is done by closing and reopening the trace file (this is slow). It helps ensure a complete log file should the application crash.
- `a[,file]` - Appends trace output to the specified file.
- `d` - Enables output from `DEBUG_<N>` macros for the current state. May be followed by a list of keywords which selects output only for the `DEBUG` macros with that keyword. An empty list of keywords implies output for all macros.
- `f[,functions]` - Limits debugger actions to the specified list of functions. An empty list of functions implies that all functions are selected.
- `F` - Marks each debugger output line with the name of the source file containing the macro causing the output.
- `i` - Marks each debugger output line with the PID of the current process.

- L - Marks each debugger output line with the name of the source file line number of the macro causing the output.
- n - Marks each debugger output line with the current function nesting depth
- o[,file] - Similar to a[,file] but overwrites old file, and does not append.
- O[,file] - Similar to A[,file] but overwrites old file, and does not append.
- t[,N] - Enables function control flow tracing. The maximum nesting depth is specified by N, and defaults to 200.
- x - This option activates profiling.

Example:

```
d:\t:x:0,/tmp/mysqlnd.trace
```



Note

This feature is only available with a debug build of PHP. Works on Microsoft Windows if using a debug build of PHP and PHP was built using Microsoft Visual C version 9 and above.

[mysqlnd.log_mask](#) integer

Defines which queries will be logged. The default 0, which disables logging. Define using an integer, and not with PHP constants. For example, a value of 48 (16 + 32) will log slow queries which either use 'no good index' (SERVER_QUERY_NO_GOOD_INDEX_USED = 16) or no index at all (SERVER_QUERY_NO_INDEX_USED = 32). A value of 2043 (1 + 2 + 8 + ... + 1024) will log all slow query types.

The types are as follows: SERVER_STATUS_IN_TRANS=1, SERVER_STATUS_AUTOCOMMIT=2, SERVER_MORE_RESULTS_EXISTS=8, SERVER_QUERY_NO_GOOD_INDEX_USED=16, SERVER_QUERY_NO_INDEX_USED=32, SERVER_STATUS_CURSOR_EXISTS=64, SERVER_STATUS_LAST_ROW_SENT=128, SERVER_STATUS_DB_DROPPED=256, SERVER_STATUS_NO_BACKSLASH_ESCAPES=512, and SERVER_QUERY_WAS_SLOW=1024.

[mysqlnd.mempool_default_size](#) integer

Default size of the mysqlnd memory pool, which is used by result sets.

[mysqlnd.net_read_timeout](#) integer

[mysqlnd](#) and the MySQL Client Library, [libmysql](#) use different networking APIs. [mysqlnd](#) uses PHP streams, whereas [libmysql](#) uses its own wrapper around the operating level network calls. PHP, by default, sets a read timeout of 60s for streams. This is set via [php.ini](#), [default_socket_timeout](#). This default applies to all streams that set no other timeout value. [mysqlnd](#) does not set any other value and therefore connections of long running queries can be disconnected

after `default_socket_timeout` seconds resulting in an error message “2006 - MySQL Server has gone away”. The MySQL Client Library sets a default timeout of $365 * 24 * 3600$ seconds (1 year) and waits for other timeouts to occur, such as TCP/IP timeouts. `mysqlnd` now uses the same very long timeout. The value is configurable through a new `php.ini` setting: `mysqlnd.net_read_timeout`. `mysqlnd.net_read_timeout` gets used by any extension (`ext/mysql`, `ext/mysqli`, `PDO_MySQL`) that uses `mysqlnd`. `mysqlnd` tells PHP Streams to use `mysqlnd.net_read_timeout`. Please note that there may be subtle differences between `MYSQL_OPT_READ_TIMEOUT` from the MySQL Client Library and PHP Streams, for example `MYSQL_OPT_READ_TIMEOUT` is documented to work only for TCP/IP connections and, prior to MySQL 5.1.2, only for Windows. PHP streams may not have this limitation. Please check the streams documentation, if in doubt.

`mysqlnd.net_cmd_buffer_size`
long
`mysqlnd` allocates an internal command/network buffer of `mysqlnd.net_cmd_buffer_size` (in `php.ini`) bytes for every connection. If a MySQL Client Server protocol command, for example, `COM_QUERY` (“normal” query), does not fit into the buffer, `mysqlnd` will grow the buffer to the size required for sending the command. Whenever the buffer gets extended for one connection, `command_buffer_too_small` will be incremented by one.

If `mysqlnd` has to grow the buffer beyond its initial size of `mysqlnd.net_cmd_buffer_size` bytes for almost every connection, you should consider increasing the default size to avoid re-allocations.

The default buffer size is 2048 bytes in PHP 5.3.0. In later versions the default is 4096 bytes. The default can be changed either through the `php.ini` setting `mysqlnd.net_cmd_buffer_size` or using `mysqli_options(MYSQLI_OPT_NET_CMD_BUFFER_SIZE, int size)`.

It is recommended that the buffer size be set to no less than 4096 bytes because `mysqlnd` also uses it when reading certain communication packets from MySQL. In PHP 5.3.0, `mysqlnd` will not grow the buffer if MySQL sends a packet that is larger than the current size of the buffer. As a consequence, `mysqlnd` is unable to decode the packet and the client application will get an error. There are only two situations when the packet can be larger than the 2048 bytes default of `mysqlnd.net_cmd_buffer_size` in PHP 5.3.0: the packet transports a very long error message, or the packet holds column meta data from `COM_LIST_FIELD` (`mysql_list_fields()`) and the meta data come from a string column with a very long default value (>1900 bytes).

As of PHP 5.3.2 `mysqlnd` does not allow setting buffers smaller than 4096 bytes.

The value can also be set using `mysqli_options(link, MYSQLI_OPT_NET_CMD_BUFFER_SIZE, size)`.

`mysqlnd.net_read_buffer_size`
long
Maximum read chunk size in bytes when reading the body of a MySQL command packet. The MySQL client server protocol encapsulates all its commands in packets. The packets consist of a small header and

a body with the actual payload. The size of the body is encoded in the header. `mysqlnd` reads the body in chunks of `MIN(header.size, mysqlnd.net_read_buffer_size)` bytes. If a packet body is larger than `mysqlnd.net_read_buffer_size` bytes, `mysqlnd` has to call `read()` multiple times.

The value can also be set using `mysqli_options(link, MYSQLI_OPT_NET_READ_BUFFER_SIZE, size)`.

`mysqlnd.sha256_server_public_key` string **SHA-256 Authentication Plugin** related. File with the MySQL server public RSA key.

Clients can either omit setting a public RSA key, specify the key through this PHP configuration setting or set the key at runtime using `mysqli_options`. If not public RSA key file is given by the client, then the key will be exchanged as part of the standard SHA-256 Authentication Plugin authentication procedure.

22.9.5.4. Persistent Connections

Copyright 1997-2012 the PHP Documentation Group. [3186]

Using Persistent Connections

If `mysqli` is used with `mysqlnd`, when a persistent connection is created it generates a `COM_CHANGE_USER` (`mysql_change_user()`) call on the server. This ensures that re-authentication of the connection takes place.

As there is some overhead associated with the `COM_CHANGE_USER` call, it is possible to switch this off at compile time. Reusing a persistent connection will then generate a `COM_PING` (`mysql_ping`) call to simply test the connection is reusable.

Generation of `COM_CHANGE_USER` can be switched off with the compile flag `MYSQLI_NO_CHANGE_USER_ON_PCONNECT`. For example:

```
shell# CFLAGS="-DMYSQLI_NO_CHANGE_USER_ON_PCONNECT" ./configure --with-mysql=/usr/local/mysql/ --with-mysqli=
```

Or alternatively:

```
shell# export CFLAGS="-DMYSQLI_NO_CHANGE_USER_ON_PCONNECT"
shell# configure --whatever-option
shell# make clean
shell# make
```

Note that only `mysqli` on `mysqlnd` uses `COM_CHANGE_USER`. Other extension-driver combinations use `COM_PING` on initial use of a persistent connection.

22.9.5.5. Statistics

Copyright 1997-2012 the PHP Documentation Group. [3186]

Using Statistical Data

MySQL Native Driver contains support for gathering statistics on the communication between the client and the server. The statistics gathered are of three main types:

- Client statistics

- Connection statistics
- Zval cache statistics

If you are using the `mysqli` extension, these statistics can be obtained through three API calls:

- `mysqli_get_client_stats`
- `mysqli_get_connection_stats`
- `mysqli_get_cache_stats`



Note

Statistics are aggregated among all extensions that use MySQL Native Driver. For example, when compiling both `ext/mysql` and `ext/mysqli` against MySQL Native Driver, both function calls of `ext/mysql` and `ext/mysqli` will change the statistics. There is no way to find out how much a certain API call of any extension that has been compiled against MySQL Native Driver has impacted a certain statistic. You can configure the PDO MySQL Driver, `ext/mysql` and `ext/mysqli` to optionally use the MySQL Native Driver. When doing so, all three extensions will change the statistics.

Accessing Client Statistics

To access client statistics, you need to call `mysqli_get_client_stats`. The function call does not require any parameters.

The function returns an associative array that contains the name of the statistic as the key and the statistical data as the value.

Client statistics can also be accessed by calling the `phpinfo` function.

Accessing Connection Statistics

To access connection statistics call `mysqli_get_connection_stats`. This takes the database connection handle as the parameter.

The function returns an associative array that contains the name of the statistic as the key and the statistical data as the value.

Accessing Zval Cache Statistics

The MySQL Native Driver also collects statistics from its internal Zval cache. These statistics can be accessed by calling `mysqli_get_cache_stats`.

The Zval cache statistics obtained may lead to a tweaking of `php.ini` settings related to the Zval cache, resulting in better performance.

Buffered and Unbuffered Result Sets

Result sets can be buffered or unbuffered. Using default settings, `ext/mysql` and `ext/mysqli` work with buffered result sets for normal (non prepared statement) queries. Buffered result sets are cached on the client. After the query execution all results are fetched from the MySQL Server and stored in a cache on the client. The big advantage of buffered result sets is that they allow the server to free all resources allocated to a result set, once the results have been fetched by the client.

Unbuffered result sets on the other hand are kept much longer on the server. If you want to reduce memory consumption on the client, but increase load on the server, use unbuffered results. If you experience a high server load and the figures for unbuffered result sets are high, you should consider

moving the load to the clients. Clients typically scale better than servers. “Load” does not only refer to memory buffers - the server also needs to keep other resources open, for example file handles and threads, before a result set can be freed.

Prepared Statements use unbuffered result sets by default. However, you can use `mysqli_stmt_store_result` to enable buffered result sets.

Statistics returned by MySQL Native Driver

The following tables show a list of statistics returned by the `mysqli_get_client_stats`, `mysqli_get_connection_stats` and `mysqli_get_cache_stats` functions.

Table 22.59. Returned *mysqli* statistics: Network

Statistic	Scope	Description	Notes
<code>bytes_sent</code>	Connection	Number of bytes sent from PHP to the MySQL server	Can be used to check the efficiency of the compression protocol
<code>bytes_received</code>	Connection	Number of bytes received from MySQL server	Can be used to check the efficiency of the compression protocol
<code>packets_sent</code>	Connection	Number of MySQL Client Server protocol packets sent	Used for debugging Client Server protocol implementation
<code>packets_received</code>	Connection	Number of MySQL Client Server protocol packets received	Used for debugging Client Server protocol implementation
<code>protocol_overhead_in</code>	Connection	MySQL Client Server protocol overhead in bytes for incoming traffic. Currently only the Packet Header (4 bytes) is considered as overhead. <code>protocol_overhead_in = packets_received * 4</code>	Used for debugging Client Server protocol implementation
<code>protocol_overhead_out</code>	Connection	MySQL Client Server protocol overhead in bytes for outgoing traffic. Currently only the Packet Header (4 bytes) is considered as overhead. <code>protocol_overhead_out = packets_sent * 4</code>	Used for debugging Client Server protocol implementation
<code>bytes_received_ok</code>	Connection	Total size in bytes of MySQL Client Server protocol OK packets received. OK packets can contain a status message. The length of the status message can vary and thus the size of an OK packet is not fixed.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received_ok</code>	Connection	Number of MySQL Client Server protocol OK packets received.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>bytes_received_eof</code>	Connection	Total size in bytes of MySQL Client Server protocol EOF packets received. EOF can vary in size depending on the server version. Also, EOF can transport an error message.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received_eof</code>	Connection	Number of MySQL Client Server protocol EOF packets. Like with other packet	Used for debugging CS protocol implementation. Note that the total size

Statistic	Scope	Description	Notes
		statistics the number of packets will be increased even if PHP does not receive the expected packet but, for example, an error message.	in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>bytes_received</code>	Connection	Total size in bytes of MySQL Client Server protocol result set header packets. The size of the packets varies depending on the payload (<code>LOAD LOCAL INFILE</code> , <code>INSERT</code> , <code>UPDATE</code> , <code>SELECT</code> , error message).	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received</code>	Connection	Number of MySQL Client Server protocol result set header packets.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>bytes_received</code>	Connection	Total size in bytes of MySQL Client Server protocol result set meta data (field information) packets. Of course the size varies with the fields in the result set. The packet may also transport an error or an EOF packet in case of <code>COM_LIST_FIELDS</code> .	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received</code>	Connection	Number of MySQL Client Server protocol result set meta data (field information) packets.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>bytes_received</code>	Connection	Total size in bytes of MySQL Client Server protocol result set row data packets. The packet may also transport an error or an EOF packet. You can reverse engineer the number of error and EOF packets by subtracting <code>rows_fetched_from_server_normal</code> and <code>rows_fetched_from_server_ps</code> from <code>bytes_received_rset_row_packet</code> .	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received</code>	Connection	Number of MySQL Client Server protocol result set row data packets and their total size in bytes.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>bytes_received</code>	Connection	Total size in bytes of MySQL Client Server protocol OK for Prepared Statement Initialization packets (prepared statement init packets). The packet may also transport an error. The packet size depends on the MySQL version: 9 bytes with MySQL 4.1 and 12 bytes from MySQL 5.0 on. There is no safe way to know how many errors happened. You may be able to guess that an error	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).

Statistic	Scope	Description	Notes
		has occurred if, for example, you always connect to MySQL 5.0 or newer and, <code>bytes_received_prepare_response_packet</code> != <code>packets_received_prepare_response</code> * 12. See also <code>ps_prepared_never_executed</code> , <code>ps_prepared_once_executed</code> .	
<code>packets_received_prepare_response</code>	Connection	Number of MySQL Client Server protocol OK for Prepared Statement Initialization packets (prepared statement init packets).	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>bytes_received_prepare_response</code>	Connection	Total size in bytes of MySQL Client Server protocol COM_CHANGE_USER packets. The packet may also transport an error or EOF.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received_prepare_response</code>	Connection	Number of MySQL Client Server protocol COM_CHANGE_USER packets	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received_prepare_response</code>	Connection	Number of MySQL Client Server protocol commands sent from PHP to MySQL. There is no way to know which specific commands and how many of them have been sent. At its best you can use it to check if PHP has sent any commands to MySQL to know if you can consider to disable MySQL support in your PHP binary. There is also no way to reverse engineer the number of errors that may have occurred while sending data to MySQL. The only error that is recorded is <code>command_buffer_too_small</code> (see below).	Only useful for debugging CS protocol implementation.
<code>bytes_received_prepare_response</code>	Connection	Number of bytes of payload fetched by the PHP client from <i>mysqlnd</i> using the text protocol.	<p>This is the size of the actual data contained in result sets that do not originate from prepared statements and which have been fetched by the PHP client. Note that although a full result set may have been pulled from MySQL by <i>mysqlnd</i>, this statistic only counts actual data pulled from <i>mysqlnd</i> by the PHP client. An example of a code sequence that will increase the value is as follows:</p> <pre> mysqlnd \$mysqli = new mysqli(); \$res = \$mysqli->query("SELECT 'abc'"); \$res->fetch_assoc(); \$res->close(); </pre> <p>Every fetch operation will increase the value.</p>

Statistic	Scope	Description	Notes
			<p>The statistic will not be increased if the result set is only buffered on the client, but not fetched, such as in the following example:</p> <pre>\$mysqli = new mysqli(); \$res = \$mysqli->query("SELECT 'abc'"); \$res->close();</pre> <p>This statistic is available as of PHP version 5.3.4.</p>
bytes_received	Connection	Number of bytes of the payload fetched by the PHP client from <i>mysqlnd</i> using the prepared statement protocol.	<p>This is the size of the actual data contained in result sets that originate from prepared statements and which has been fetched by the PHP client. The value will not be increased if the result set is not subsequently read by the PHP client. Note that although a full result set may have been pulled from MySQL by <i>mysqlnd</i>, this statistic only counts actual data pulled from <i>mysqlnd</i> by the PHP client. See also bytes_received_real_data_normal. This statistic is available as of PHP version 5.3.4.</p>

Result Set

Table 22.60. Returned *mysqlnd* statistics: Result Set

Statistic	Scope	Description	Notes
result_set_queries	Connection	Number of queries that have generated a result set. Examples of queries that generate a result set: SELECT , SHOW . The statistic will not be incremented if there is an error reading the result set header packet from the line.	You may use it as an indirect measure for the number of queries PHP has sent to MySQL, for example, to identify a client that causes a high database load.
non_result_set_queries	Connection	Number of queries that did not generate a result set. Examples of queries that do not generate a result set: INSERT , UPDATE , LOAD DATA , SHOW . The statistic will not be incremented if there is an error reading the result set header packet from the line.	You may use it as an indirect measure for the number of queries PHP has sent to MySQL, for example, to identify a client that causes a high database load.
no_index_queries	Connection	Number of queries that have generated a result set but did not use an index (see also <i>mysqld</i> start option <code>-log-queries-not-using-indexes</code>). If you want these queries to be reported you can use <code>mysqli_report(MYSQLI_REPORT_INDEX)</code> to make <i>ext/mysqli</i> throw an exception. If you prefer a warning	

Statistic	Scope	Description	Notes
		instead of an exception use <code>mysqli_report(MYSQLI_REPORT_INDEX ^ MYSQLI_REPORT_STRICT)</code> .	
bad_index	Connection	Number of queries that have generated a result set and did not use a good index (see also <code>mysqld</code> start option <code>-log-slow-queries</code>).	If you want these queries to be reported you can use <code>mysqli_report(MYSQLI_REPORT_INDEX)</code> to make <code>ext/mysqli</code> throw an exception. If you prefer a warning instead of an exception use <code>mysqli_report(MYSQLI_REPORT_INDEX ^ MYSQLI_REPORT_STRICT)</code>
slow_query	Connection	SQL statements that took more than <code>long_query_time</code> seconds to execute and required at least <code>min_examined_row_limit</code> rows to be examined.	Not reported through <code>mysqli_report</code>
buffered	Connection	Number of buffered result sets returned by “normal” queries. “Normal” means “not prepared statement” in the following notes.	Examples of API calls that will buffer result sets on the client: <code>mysql_query</code> , <code>mysqli_query</code> , <code>mysqli_store_result</code> , <code>mysqli_stmt_get_result</code> . Buffering result sets on the client ensures that server resources are freed as soon as possible and it makes result set scrolling easier. The downside is the additional memory consumption on the client for buffering data. Note that <code>mysqlnd</code> (unlike the MySQL Client Library) respects the PHP memory limit because it uses PHP internal memory management functions to allocate memory. This is also the reason why <code>memory_get_usage</code> reports a higher memory consumption when using <code>mysqlnd</code> instead of the MySQL Client Library. <code>memory_get_usage</code> does not measure the memory consumption of the MySQL Client Library at all because the MySQL Client Library does not use PHP internal memory management functions monitored by the function!
unbuffered	Connection	Number of unbuffered result sets returned by normal (non prepared statement) queries.	Examples of API calls that will not buffer result sets on the client: <code>mysqli_use_result</code>
ps_buffered	Connection	Number of buffered result sets returned by prepared statements. By default prepared statements are unbuffered.	Examples of API calls that will not buffer result sets on the client: <code>mysqli_stmt_store_result</code>
ps_unbuffered	Connection	Number of unbuffered result sets returned by prepared statements.	By default prepared statements are unbuffered.

Statistic	Scope	Description	Notes
flushed_normal	Connection	Number of result sets from normal (non prepared statement) queries with unread data which have been flushed silently for you. Flushing happens only with unbuffered result sets.	<p>Unbuffered result sets must be fetched completely before a new query can be run on the connection otherwise MySQL will throw an error. If the application does not fetch all rows from an unbuffered result set, mysqlnd does implicitly fetch the result set to clear the line. See also rows_skipped_normal, rows_skipped_ps. Some possible causes for an implicit flush:</p> <ul style="list-style-type: none"> • Faulty client application • Client stopped reading after it found what it was looking for but has made MySQL calculate more records than needed • Client application has stopped unexpectedly
flushed_ps	Connection	Number of result sets from prepared statements with unread data which have been flushed silently for you. Flushing happens only with unbuffered result sets.	<p>Unbuffered result sets must be fetched completely before a new query can be run on the connection otherwise MySQL will throw an error. If the application does not fetch all rows from an unbuffered result set, mysqlnd does implicitly fetch the result set to clear the line. See also rows_skipped_normal, rows_skipped_ps. Some possible causes for an implicit flush:</p> <ul style="list-style-type: none"> • Faulty client application • Client stopped reading after it found what it was looking for but has made MySQL calculate more records than needed • Client application has stopped unexpectedly
ps_prepared_not_executed	Connection	Number of statements prepared but never executed.	Prepared statements occupy server resources. You should not prepare a statement if you do not plan to execute it.
ps_prepared_executed	Connection	Number of prepared statements executed only once.	One of the ideas behind prepared statements is that the same query gets executed over and over again (with different parameters) and some parsing and other preparation work can be saved, if statement execution is split up in separate prepare and execute stages. The idea is to prepare once and “cache” results, for example, the

Statistic	Scope	Description	Notes
			parse tree to be reused during multiple statement executions. If you execute a prepared statement only once the two stage processing can be inefficient compared to "normal" queries because all the caching means extra work and it takes (limited) server resources to hold the cached information. Consequently, prepared statements that are executed only once may cause performance hurts.
rows_fetched_from_buffered_result	Connection	Total number of result set rows successfully fetched from MySQL regardless if the client application has consumed them or not. Some of the rows may not have been fetched by the client application but have been flushed implicitly.	See also packets_received_rset_row
rows_buffered_from_buffered_result	Connection	Total number of successfully buffered rows originating from a "normal" query or a prepared statement. This is the number of rows that have been fetched from MySQL and buffered on client. Note that there are two distinct statistics on rows that have been buffered (MySQL to mysqlnd internal buffer) and buffered rows that have been fetched by the client application (mysqlnd internal buffer to client application). If the number of buffered rows is higher than the number of fetched buffered rows it can mean that the client application runs queries that cause larger result sets than needed resulting in rows not read by the client.	Examples of queries that will buffer results: mysqli_query , mysqli_store_result
rows_fetched_from_unbuffered_result	Connection	Total number of rows fetched by the client from a buffered result set created by a normal query or a prepared statement.	
rows_fetched_from_unbuffered_result	Connection	Total number of rows fetched by the client from an unbuffered result set created by a "normal" query or a prepared statement.	
rows_fetched_from_cursor	Connection	Total number of rows fetched by the client from a cursor created by a prepared statement.	
rows_skipped_ps	Connection	Reserved for future use (currently not supported)	
copy_on_write_per_thread	Process	With mysqlnd , variables returned by the extension point into mysqlnd internal network result buffers. If you do not	

Statistic	Scope	Description	Notes
		change the variables, fetched data will be kept only once in memory. If you change the variables, <i>mysqld</i> has to perform a copy-on-write to protect the internal network result buffers from being changed. With the MySQL Client Library you always hold fetched data twice in memory. Once in the internal MySQL Client Library buffers and once in the variables returned by the extensions. In theory <i>mysqld</i> can save up to 40% memory. However, note that the memory saving cannot be measured using memory_get_usage .	
explicit implicit	Connection, Process (only during prepared statement cleanup)	Total number of freed result sets.	The free is always considered explicit but for result sets created by an init command, for example, mysqli_options(MYSQLI_INIT_COMMAND ,
proto_text proto_text_fetched proto_text_fetched_int proto_text_fetched_short proto_text_fetched_int24 proto_text_fetched_int proto_text_fetched_bigint proto_text_fetched_decimal proto_text_fetched_float proto_text_fetched_double proto_text_fetched_date proto_text_fetched_year proto_text_fetched_time proto_text_fetched_datetime proto_text_fetched_timestamp proto_text_fetched_string proto_text_fetched_blob proto_text_fetched_enum proto_text_fetched_set proto_text_fetched_geometry proto_text_fetched_other	Connection, Process (only during prepared statement cleanup)	Total number of columns of a certain type fetched from a normal query (MySQL text protocol).	Mapping from C API / MySQL meta data type to statistics name: <ul style="list-style-type: none"> MYSQL_TYPE_NULL - proto_text_fetched_null MYSQL_TYPE_BIT - proto_text_fetched_bit MYSQL_TYPE_TINY - proto_text_fetched_tinyint MYSQL_TYPE_SHORT - proto_text_fetched_short MYSQL_TYPE_INT24 - proto_text_fetched_int24 MYSQL_TYPE_LONG - proto_text_fetched_int MYSQL_TYPE_LONGLONG - proto_text_fetched_bigint MYSQL_TYPE_DECIMAL, MYSQL_TYPE_NEWDECIMAL - proto_text_fetched_decimal MYSQL_TYPE_FLOAT - proto_text_fetched_float

Statistic	Scope	Description	Notes
proto_binary_fetched_decimal, proto_binary_fetched_float, proto_binary_fetched_double, proto_binary_fetched_date, proto_binary_fetched_year, proto_binary_fetched_time, proto_binary_fetched_datetime, proto_binary_fetched_timestamp, proto_binary_fetched_string, proto_binary_fetched_blob, proto_binary_fetched_enum, proto_binary_fetched_set, proto_binary_fetched_geometry, proto_binary_fetched_other			

Table 22.61. Returned mysqlnd statistics: Connection

Statistic	Scope	Description	Notes
connect_success, connect_failure	Connection	Total number of successful / failed connection attempt.	Reused connections and all other kinds of connections are included.
reconnect	Process	Total number of (real_)connect attempts made on an already opened connection handle.	The code sequence <code>\$link = new mysqli(...); \$link->real_connect(...)</code> will cause a reconnect. But <code>\$link = new mysqli(...); \$link->connect(...)</code> will not because <code>\$link->connect(...)</code> will explicitly close the existing connection before a new connection is established.
pconnect_success	Connection	Total number of successful persistent connection attempts.	Note that <code>connect_success</code> holds the sum of successful persistent and non-persistent connection attempts. The number of successful non-persistent connection attempts is <code>connect_success - pconnect_success</code> .
active_connections	Connection	Total number of active persistent and non-persistent connections.	
active_persistent_connections	Connection	Total number of active persistent connections.	The total number of active non-persistent connections is <code>active_connections - active_persistent_connections</code> .
explicit_closes	Connection	Total number of explicitly closed connections (ext/mysqli only).	Examples of code snippets that cause an explicit close : <pre>\$link = new mysqli(...); \$link->close(...) \$link = new mysqli(...); \$link->connect(...)</pre>
implicit_closes	Connection	Total number of implicitly closed connections (ext/mysqli only).	Examples of code snippets that cause an implicit close :

Statistic	Scope	Description	Notes
			<ul style="list-style-type: none"> <code>\$link = new mysqli(...);</code> <code>\$link->real_connect(...)</code> <code>unset(\$link)</code> Persistent connection: pooled connection has been created with <code>real_connect</code> and there may be unknown options set - close implicitly to avoid returning a connection with unknown options Persistent connection: ping/change_user fails and ext/mysqli closes the connection end of script execution: close connections that have not been closed by the user
<code>disconnect</code>	Connection	Connection failures indicated by the C API call <code>mysql_real_connect</code> during an attempt to establish a connection.	It is called <code>disconnect_close</code> because the connection handle passed to the C API call will be closed.
<code>in_middle</code>	Process	A connection has been closed in the middle of a command execution (outstanding result sets not fetched, after sending a query and before retrieving an answer, while fetching data, while transferring data with LOAD DATA).	Unless you use asynchronous queries this should only happen if your script stops unexpectedly and PHP shuts down the connections for you.
<code>init_command</code>	Connection	Total number of init command executions, for example, <code>mysqli_options(MYSQLI_INIT_COMMAND, ...)</code>	The number of successful executions is <code>init_command_executed_count</code> and failed count.
<code>init_command</code>	Connection	Total number of failed init commands.	

Table 22.62. Returned mysqlnd statistics: COM_* Command

Statistic	Scope	Description	Notes
<code>com_quit</code> , <code>com_init_db</code> , <code>com_query</code> , <code>com_field_list</code> , <code>com_create_db</code> , <code>com_drop_db</code> , <code>com_refresh</code> , <code>com_shutdown</code> , <code>com_statistics</code> , <code>com_process_info</code> , <code>com_connect</code> , <code>com_process_kill</code> , <code>com_debug</code> , <code>com_ping</code> , <code>com_time</code> , <code>com_delayed_insert</code> ,	Connection	Total number of attempts to send a certain COM_* command from PHP to MySQL.	<p>The statistics are incremented after checking the line and immediately before sending the corresponding MySQL client server protocol packet. If mysqlnd fails to send the packet over the wire the statistics will not be decremented. In case of a failure mysqlnd emits a PHP warning "Error while sending %s packet. PID= %d."</p> <p>Usage examples:</p> <ul style="list-style-type: none"> Check if PHP sends certain commands to MySQL, for example, check if a client sends <code>COM_PROCESS_KILL</code>

Statistic	Scope	Description	Notes
com_change_user, com_binlog_dump, com_table_dump, com_connect_out, com_register_slave, com_stmt_prepare, com_stmt_execute, com_stmt_send_long_data, com_stmt_close, com_stmt_reset, com_stmt_set_option, com_stmt_fetch, com_daemon			<ul style="list-style-type: none"> Calculate the average number of prepared statement executions by comparing <code>COM_EXECUTE</code> with <code>COM_PREPARE</code> Check if PHP has run any non-prepared SQL statements by checking if <code>COM_QUERY</code> is zero Identify PHP scripts that run an excessive number of SQL statements by checking <code>COM_QUERY</code> and <code>COM_EXECUTE</code>

Miscellaneous

Table 22.63. Returned *mysqld* statistics: Miscellaneous

Statistic	Scope	Description	Notes
explicit_stmt_close, implicit_stmt_close	Process	Total number of close prepared statements.	A close is always considered explicit but for a failed prepare.
mem_emalloc_count, mem_emalloc_ammount, mem_ecalloc_count, mem_ecalloc_ammount, mem_erealloc_count, mem_erealloc_ammount, mem_efree_count, mem_malloc_count, mem_malloc_ammount, mem_calloc_count, mem_calloc_ammount, mem_realloc_count, mem_realloc_ammount, mem_free_count	Process	Memory management calls.	Development only.
command_buffer_too_small	Connection	Number of network command buffer extensions while sending commands from PHP to MySQL.	<p><i>mysqld</i> allocates an internal command/network buffer of <code>mysqld.net_cmd_buffer_size</code> (<code>php.ini</code>) bytes for every connection. If a MySQL Client Server protocol command, for example, <code>COM_QUERY</code> (normal query), does not fit into the buffer, <i>mysqld</i> will grow the buffer to what is needed for sending the command. Whenever the buffer gets extended for one connection <code>command_buffer_too_small</code> will be incremented by one.</p> <p>If <i>mysqld</i> has to grow the buffer beyond its initial size of <code>mysqld.net_cmd_buffer_size</code></p>

Statistic	Scope	Description	Notes
			<p>(<code>php.ini</code>) bytes for almost every connection, you should consider to increase the default size to avoid re-allocations.</p> <p>The default buffer size is 2048 bytes in PHP 5.3.0. In future versions the default will be 4kB or larger. The default can be changed either through the <code>php.ini</code> setting <code>mysqlnd.net_cmd_buffer_size</code> or using <code>mysqli_options(MYSQLI_OPT_NET_CMD_BUFFER_SIZE, int size)</code>.</p> <p>It is recommended to set the buffer size to no less than 4096 bytes because <code>mysqlnd</code> also uses it when reading certain communication packets from MySQL. In PHP 5.3.0, <code>mysqlnd</code> will not grow the buffer if MySQL sends a packet that is larger than the current size of the buffer. As a consequence <code>mysqlnd</code> is unable to decode the packet and the client application will get an error. There are only two situations when the packet can be larger than the 2048 bytes default of <code>mysqlnd.net_cmd_buffer_size</code> in PHP 5.3.0: the packet transports a very long error message or the packet holds column meta data from <code>COM_LIST_FIELDS</code> (<code>mysql_list_fields</code>) and the meta data comes from a string column with a very long default value (>1900 bytes). No bug report on this exists - it should happen rarely.</p> <p>As of PHP 5.3.2 <code>mysqlnd</code> does not allow setting buffers smaller than 4096 bytes.</p>
<code>connection_reused</code>			

22.9.5.6. Notes

Copyright 1997-2012 the PHP Documentation Group. [3186]

This section provides a collection of miscellaneous notes on MySQL Native Driver usage.

- Using `mysqlnd` means using PHP streams for underlying connectivity. For `mysqlnd`, the PHP streams documentation (<http://www.php.net/manual/en/book.stream>) should be consulted on such details as timeout settings, not the documentation for the MySQL Client Library.

22.9.5.7. MySQL Native Driver Plugin API

Copyright 1997-2012 the PHP Documentation Group. [3186]

The MySQL Native Driver Plugin API is a feature of MySQL Native Driver, or `mysqlnd`. `mysqlnd` plugins operate in the layer between PHP applications and the MySQL server. This is comparable to MySQL Proxy. MySQL Proxy operates on a layer between any MySQL client application, for example, a PHP application and, the MySQL server. `mysqlnd` plugins can undertake typical MySQL Proxy tasks such as load balancing, monitoring and performance optimizations. Due to the different architecture and location, `mysqlnd` plugins do not have some of MySQL Proxy's disadvantages. For example, with plugins, there is no single point of failure, no dedicated proxy server to deploy, and no new programming language to learn (Lua).

A `mysqlnd` plugin can be thought of as an extension to `mysqlnd`. Plugins can intercept the majority of `mysqlnd` functions. The `mysqlnd` functions are called by the PHP MySQL extensions such as `ext/mysql`, `ext/mysqli`, and `PDO_MYSQL`. As a result, it is possible for a `mysqlnd` plugin to intercept all calls made to these extensions from the client application.

Internal `mysqlnd` function calls can also be intercepted, or replaced. There are no restrictions on manipulating `mysqlnd` internal function tables. It is possible to set things up so that when certain `mysqlnd` functions are called by the extensions that use `mysqlnd`, the call is directed to the appropriate function in the `mysqlnd` plugin. The ability to manipulate `mysqlnd` internal function tables in this way allows maximum flexibility for plugins.

`mysqlnd` plugins are in fact PHP Extensions, written in C, that use the `mysqlnd` plugin API (which is built into MySQL Native Driver, `mysqlnd`). Plugins can be made 100% transparent to PHP applications. No application changes are needed because plugins operate on a different layer. The `mysqlnd` plugin can be thought of as operating in a layer below `mysqlnd`.

The following list represents some possible applications of `mysqlnd` plugins.

- Load Balancing
 - Read/Write Splitting. An example of this is the PECL/`mysqlnd_ms` (Master Slave) extension. This extension splits read/write queries for a replication setup.
 - Failover
 - Round-Robin, least loaded
- Monitoring
 - Query Logging
 - Query Analysis
 - Query Auditing. An example of this is the PECL/`mysqlnd_sip` (SQL Injection Protection) extension. This extension inspects queries and executes only those that are allowed according to a ruleset.
- Performance
 - Caching. An example of this is the PECL/`mysqlnd_qc` (Query Cache) extension.
 - Throttling
 - Sharding. An example of this is the PECL/`mysqlnd_mc` (Multi Connect) extension. This extension will attempt to split a SELECT statement into n-parts, using `SELECT ... LIMIT part_1, SELECT LIMIT part_n`. It sends the queries to distinct MySQL servers and merges the result at the client.

MySQL Native Driver Plugins Available

There are a number of [mysqlnd](#) plugins already available. These include:

- [PECL/mysqlnd_mc](#) - Multi Connect plugin.
- [PECL/mysqlnd_ms](#) - Master Slave plugin.
- [PECL/mysqlnd_qc](#) - Query Cache plugin.
- [PECL/mysqlnd_pscache](#) - Prepared Statement Handle Cache plugin.
- [PECL/mysqlnd_sip](#) - SQL Injection Protection plugin.
- [PECL/mysqlnd_uh](#) - User Handler plugin.

22.9.5.7.1. A comparison of [mysqlnd](#) plugins with MySQL Proxy

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

[Mysqlnd](#) plugins and MySQL Proxy are different technologies using different approaches. Both are valid tools for solving a variety of common tasks such as load balancing, monitoring, and performance enhancements. An important difference is that MySQL Proxy works with all MySQL clients, whereas [mysqlnd](#) plugins are specific to PHP applications.

As a PHP Extension, a [mysqlnd](#) plugin gets installed on the PHP application server, along with the rest of PHP. MySQL Proxy can either be run on the PHP application server or can be installed on a dedicated machine to handle multiple PHP application servers.

Deploying MySQL Proxy on the application server has two advantages:

1. No single point of failure
2. Easy to scale out (horizontal scale out, scale by client)

MySQL Proxy (and [mysqlnd](#) plugins) can solve problems easily which otherwise would have required changes to existing applications.

However, MySQL Proxy does have some disadvantages:

- MySQL Proxy is a new component and technology to master and deploy.
- MySQL Proxy requires knowledge of the Lua scripting language.

MySQL Proxy can be customized with C and Lua programming. Lua is the preferred scripting language of MySQL Proxy. For most PHP experts Lua is a new language to learn. A [mysqlnd](#) plugin can be written in C. It is also possible to write plugins in PHP using [PECL/mysqlnd_uh](#).

MySQL Proxy runs as a daemon - a background process. MySQL Proxy can recall earlier decisions, as all state can be retained. However, a [mysqlnd](#) plugin is bound to the request-based lifecycle of PHP. MySQL Proxy can also share one-time computed results among multiple application servers. A [mysqlnd](#) plugin would need to store data in a persistent medium to be able to do this. Another daemon would need to be used for this purpose, such as Memcache. This gives MySQL Proxy an advantage in this case.

MySQL Proxy works on top of the wire protocol. With MySQL Proxy you have to parse and reverse engineer the MySQL Client Server Protocol. Actions are limited to those that can be achieved by

manipulating the communication protocol. If the wire protocol changes (which happens very rarely) MySQL Proxy scripts would need to be changed as well.

[Mysqlnd](#) plugins work on top of the C API, which mirrors the [libmysql](#) client and Connector/C APIs. This C API is basically a wrapper around the MySQL Client Server protocol, or wire protocol, as it is sometimes called. You can intercept all C API calls. PHP makes use of the C API, therefore you can hook all PHP calls, without the need to program at the level of the wire protocol.

[Mysqlnd](#) implements the wire protocol. Plugins can therefore parse, reverse engineer, manipulate and even replace the communication protocol. However, this is usually not required.

As plugins allow you to create implementations that use two levels (C API and wire protocol), they have greater flexibility than MySQL Proxy. If a [mysqlnd](#) plugin is implemented using the C API, any subsequent changes to the wire protocol do not require changes to the plugin itself.

22.9.5.7.2. Obtaining the mysqlnd plugin API

Copyright 1997-2012 the PHP Documentation Group. [3186]

The [mysqlnd](#) plugin API is simply part of the MySQL Native Driver PHP extension, [ext/mysqlnd](#). Development started on the [mysqlnd](#) plugin API in December 2009. It is developed as part of the PHP source repository, and as such is available to the public either via Git, or through source snapshot downloads.

The following table shows PHP versions and the corresponding [mysqlnd](#) version contained within.

Table 22.64. The bundled mysqlnd version per PHP release

PHP Version	MySQL Native Driver version
5.3.0	5.0.5
5.3.1	5.0.5
5.3.2	5.0.7
5.3.3	5.0.7
5.3.4	5.0.7

Plugin developers can determine the [mysqlnd](#) version through accessing [MYSQLND_VERSION](#), which is a string of the format “mysqlnd 5.0.7-dev - 091210 - \$Revision: 300535”, or through [MYSQLND_VERSION_ID](#), which is an integer such as 50007. Developers can calculate the version number as follows:

Table 22.65. MYSQLND_VERSION_ID calculation table

Version (part)	Example
Major*10000	5*10000 = 50000
Minor*100	0*100 = 0
Patch	7 = 7
MYSQLND_VERSION_ID	50007

During development, developers should refer to the [mysqlnd](#) version number for compatibility and version tests, as several iterations of [mysqlnd](#) could occur during the lifetime of a PHP development branch with a single PHP version number.

22.9.5.7.3. MySQL Native Driver Plugin Architecture

Copyright 1997-2012 the PHP Documentation Group. [3186]

This section provides an overview of the `mysqlnd` plugin architecture.

MySQL Native Driver Overview

Before developing `mysqlnd` plugins, it is useful to know a little of how `mysqlnd` itself is organized. `mysqlnd` consists of the following modules:

Table 22.66. The `mysqlnd` organization chart, per module

Modules Statistics	<code>mysqlnd_statistics.c</code>
Connection	<code>mysqlnd.c</code>
Resultset	<code>mysqlnd_result.c</code>
Resultset Metadata	<code>mysqlnd_result_meta.c</code>
Statement	<code>mysqlnd_ps.c</code>
Network	<code>mysqlnd_net.c</code>
Wire protocol	<code>mysqlnd_wireprotocol.c</code>

C Object Oriented Paradigm

At the code level, `mysqlnd` uses a C pattern for implementing object orientation.

In C you use a `struct` to represent an object. Members of the struct represent object properties. Struct members pointing to functions represent methods.

Unlike with other languages such as C++ or Java, there are no fixed rules on inheritance in the C object oriented paradigm. However, there are some conventions that need to be followed that will be discussed later.

The PHP Life Cycle

When considering the PHP life cycle there are two basic cycles:

- PHP engine startup and shutdown cycle
- Request cycle

When the PHP engine starts up it will call the module initialization (MINIT) function of each registered extension. This allows each module to setup variables and allocate resources that will exist for the lifetime of the PHP engine process. When the PHP engine shuts down it will call the module shutdown (MSHUTDOWN) function of each extension.

During the lifetime of the PHP engine it will receive a number of requests. Each request constitutes another life cycle. On each request the PHP engine will call the request initialization function of each extension. The extension can perform any variable setup and resource allocation required for request processing. As the request cycle ends the engine calls the request shutdown (RSHUTDOWN) function of each extension so the extension can perform any cleanup required.

How a plugin works

A [mysqlnd](#) plugin works by intercepting calls made to [mysqlnd](#) by extensions that use [mysqlnd](#). This is achieved by obtaining the [mysqlnd](#) function table, backing it up, and replacing it by a custom function table, which calls the functions of the plugin as required.

The following code shows how the [mysqlnd](#) function table is replaced:

```
/* a place to store original function table */
struct st_mysqlnd_conn_methods org_methods;

void minit_register_hooks(TSRMLS_D) {
    /* active function table */
    struct st_mysqlnd_conn_methods * current_methods
        = mysqlnd_conn_get_methods();

    /* backup original function table */
    memcpy(&org_methods, current_methods,
        sizeof(struct st_mysqlnd_conn_methods));

    /* install new methods */
    current_methods->query = MYSQLND_METHOD(my_conn_class, query);
}
```

Connection function table manipulations must be done during Module Initialization (MINIT). The function table is a global shared resource. In an multi-threaded environment, with a TSRM build, the manipulation of a global shared resource during the request processing will almost certainly result in conflicts.



Note

Do not use any fixed-size logic when manipulating the [mysqlnd](#) function table: new methods may be added at the end of the function table. The function table may change at any time in the future.

Calling parent methods

If the original function table entries are backed up, it is still possible to call the original function table entries - the parent methods.

In some cases, such as for [Connection::stmt_init\(\)](#), it is vital to call the parent method prior to any other activity in the derived method.

```
MYSQLND_METHOD(my_conn_class, query)(MYSQLND *conn,
    const char *query, unsigned int query_len TSRMLS_DC) {

    php_printf("my_conn_class::query(query = %s)\n", query);

    query = "SELECT 'query rewritten' FROM DUAL";
    query_len = strlen(query);

    return org_methods.query(conn, query, query_len); /* return with call to parent */
}
```

Extending properties

A [mysqlnd](#) object is represented by a C struct. It is not possible to add a member to a C struct at run time. Users of [mysqlnd](#) objects cannot simply add properties to the objects.

Arbitrary data (properties) can be added to a [mysqlnd](#) objects using an appropriate function of the [mysqlnd_plugin_get_plugin_<object>_data\(\)](#) family. When allocating an object [mysqlnd](#)

reserves space at the end of the object to hold a `void *` pointer to arbitrary data. *mysqlnd* reserves space for one `void *` pointer per plugin.

The following table shows how to calculate the position of the pointer for a specific plugin:

Table 22.67. Pointer calculations for *mysqlnd*

Memory address	Contents
0	Beginning of the <i>mysqlnd</i> object C struct
n	End of the <i>mysqlnd</i> object C struct
n + (m x sizeof(void*))	void* to object data of the m-th plugin

If you plan to subclass any of the *mysqlnd* object constructors, which is allowed, you must keep this in mind!

The following code shows extending properties:

```

/* any data we want to associate */
typedef struct my_conn_properties {
    unsigned long query_counter;
} MY_CONN_PROPERTIES;

/* plugin id */
unsigned int my_plugin_id;

void minit_register_hooks(TSRMLS_D) {
    /* obtain unique plugin ID */
    my_plugin_id = mysqlnd_plugin_register();
    /* snip - see Extending Connection: methods */
}

static MY_CONN_PROPERTIES** get_conn_properties(const MYSQLND *conn TSRMLS_DC) {
    MY_CONN_PROPERTIES** props;
    props = (MY_CONN_PROPERTIES**)mysqlnd_plugin_get_plugin_connection_data(
        conn, my_plugin_id);
    if (!props || !(*props)) {
        *props = mnd_pccalloc(1, sizeof(MY_CONN_PROPERTIES), conn->persistent);
        (*props)->query_counter = 0;
    }
    return props;
}

```

The plugin developer is responsible for the management of plugin data memory.

Use of the *mysqlnd* memory allocator is recommended for plugin data. These functions are named using the convention: `mnd_*loc()`. The *mysqlnd* allocator has some useful features, such as the ability to use a debug allocator in a non-debug build.

Table 22.68. When and how to subclass

	When to subclass?	Each instance has its own private function table?	How to subclass?
Connection (MYSQLND)	MINIT	No	<code>mysqlnd_conn_get_methods()</code>
Resultset (MYSQLND_RES)	MINIT or later	Yes	<code>mysqlnd_result_get_methods()</code> or object method function table manipulation

Resultset Meta (MYSQLND_RES_METADATA)	MINIT	No	mysqlnd_result_metadata_get_me
Statement (MYSQLND_STMT)	MINIT	No	mysqlnd_stmt_get_methods()
Network (MYSQLND_NET)	MINIT or later	Yes	mysqlnd_net_get_methods() or object method function table manipulation
Wire protocol (MYSQLND_PROTOCOL)	MINIT or later	Yes	mysqlnd_protocol_get_methods() or object method function table manipulation

You must not manipulate function tables at any time later than MINIT if it is not allowed according to the above table.

Some classes contain a pointer to the method function table. All instances of such a class will share the same function table. To avoid chaos, in particular in threaded environments, such function tables must only be manipulated during MINIT.

Other classes use copies of a globally shared function table. The class function table copy is created together with the object. Each object uses its own function table. This gives you two options: you can manipulate the default function table of an object at MINIT, and you can additionally refine methods of an object without impacting other instances of the same class.

The advantage of the shared function table approach is performance. There is no need to copy a function table for each and every object.

Table 22.69. Constructor status

	Allocation, construction, reset	Can be modified?	Caller
Connection (MYSQLND)	mysqlnd_init()	No	mysqlnd_connect()
Resultset (MYSQLND_RESULT)	Allocation: <ul style="list-style-type: none"> • Connection::result_init() Reset and re-initialized during: <ul style="list-style-type: none"> • Result::use_result() • Result::store_result 	Yes, but call parent!	<ul style="list-style-type: none"> • Connection::list_fields() • Statement::get_result() • Statement::prepare() (Metadata only) • Statement::resultMetaData()
Resultset Meta (MYSQLND_RES_METADATA)	Connection::result_meta_init()	Yes, but call parent!	Result::read_result_metadata()
Statement (MYSQLND_STMT)	Connection::stmt_init()	Yes, but call parent!	Connection::stmt_init()
Network (MYSQLND_NET)	mysqlnd_net_init()	No	Connection::init()
Wire protocol (MYSQLND_PROTOCOL)	mysqlnd_protocol_init()	No	Connection::init()

It is strongly recommended that you do not entirely replace a constructor. The constructors perform memory allocations. The memory allocations are vital for the *mysqlnd* plugin API and the object logic of

`mysqlnd`. If you do not care about warnings and insist on hooking the constructors, you should at least call the parent constructor before doing anything in your constructor.

Regardless of all warnings, it can be useful to subclass constructors. Constructors are the perfect place for modifying the function tables of objects with non-shared object tables, such as Resultset, Network, Wire Protocol.

Table 22.70. Destruction status

	Derived method must call parent?	Destructor
Connection	yes, after method execution	<code>free_contents()</code> , <code>end_psession()</code>
Resultset	yes, after method execution	<code>free_result()</code>
Resultset Meta	yes, after method execution	<code>free()</code>
Statement	yes, after method execution	<code>dtor()</code> , <code>free_stmt_content()</code>
Network	yes, after method execution	<code>free()</code>
Wire protocol	yes, after method execution	<code>free()</code>

The destructors are the appropriate place to free properties, `mysqlnd_plugin_get_plugin_<object>_data()`.

The listed destructors may not be equivalent to the actual `mysqlnd` method freeing the object itself. However, they are the best possible place for you to hook in and free your plugin data. As with constructors you may replace the methods entirely but this is not recommended. If multiple methods are listed in the above table you will need to hook all of the listed methods and free your plugin data in whichever method is called first by `mysqlnd`.

The recommended method for plugins is to simply hook the methods, free your memory and call the parent implementation immediately following this.



Caution

Due to a bug in PHP versions 5.3.0 to 5.3.3, plugins do not associate plugin data with a persistent connection. This is because `ext/mysql` and `ext/mysql_i` do not trigger all the necessary `mysqlnd_end_psession()` method calls and the plugin may therefore leak memory. This has been fixed in PHP 5.3.4.

22.9.5.7.4. The `mysqlnd` plugin API

Copyright 1997-2012 the PHP Documentation Group. [3186]

The following is a list of functions provided in the `mysqlnd` plugin API:

- `mysqlnd_plugin_register()`
- `mysqlnd_plugin_count()`
- `mysqlnd_plugin_get_plugin_connection_data()`
- `mysqlnd_plugin_get_plugin_result_data()`
- `mysqlnd_plugin_get_plugin_stmt_data()`
- `mysqlnd_plugin_get_plugin_net_data()`

- `mysqlnd_plugin_get_plugin_protocol_data()`
- `mysqlnd_conn_get_methods()`
- `mysqlnd_result_get_methods()`
- `mysqlnd_result_meta_get_methods()`
- `mysqlnd_stmt_get_methods()`
- `mysqlnd_net_get_methods()`
- `mysqlnd_protocol_get_methods()`

There is no formal definition of what a plugin is and how a plugin mechanism works.

Components often found in plugins mechanisms are:

- A plugin manager
- A plugin API
- Application services (or modules)
- Application service APIs (or module APIs)

The `mysqlnd` plugin concept employs these features, and additionally enjoys an open architecture.

No Restrictions

A plugin has full access to the inner workings of `mysqlnd`. There are no security limits or restrictions. Everything can be overwritten to implement friendly or hostile algorithms. It is recommended you only deploy plugins from a trusted source.

As discussed previously, plugins can use pointers freely. These pointers are not restricted in any way, and can point into another plugin's data. Simple offset arithmetic can be used to read another plugin's data.

It is recommended that you write cooperative plugins, and that you always call the parent method. The plugins should always cooperate with `mysqlnd` itself.

Table 22.71. Issues: an example of chaining and cooperation

Extension	<code>mysqlnd.query()</code> pointer	call stack if calling parent
<code>ext/mysqlnd</code>	<code>mysqlnd.query()</code>	<code>mysqlnd.query</code>
<code>ext/mysqlnd_cache</code>	<code>mysqlnd_cache.query()</code>	<ol style="list-style-type: none"> 1. <code>mysqlnd_cache.query()</code> 2. <code>mysqlnd.query</code>
<code>ext/mysqlnd_monitor</code>	<code>mysqlnd_monitor.query()</code>	<ol style="list-style-type: none"> 1. <code>mysqlnd_monitor.query()</code> 2. <code>mysqlnd_cache.query()</code> 3. <code>mysqlnd.query</code>

In this scenario, a cache (`ext/mysqlnd_cache`) and a monitor (`ext/mysqlnd_monitor`) plugin are loaded. Both subclass `Connection::query()`. Plugin registration happens at `MINIT` using the logic

shown previously. PHP calls extensions in alphabetical order by default. Plugins are not aware of each other and do not set extension dependencies.

By default the plugins call the parent implementation of the query method in their derived version of the method.

PHP Extension Recap

This is a recap of what happens when using an example plugin, `ext/mysqlnd_plugin`, which exposes the `mysqlnd` C plugin API to PHP:

- Any PHP MySQL application tries to establish a connection to 192.168.2.29
- The PHP application will either use `ext/mysql`, `ext/mysqli` or `PDO_MYSQL`. All three PHP MySQL extensions use `mysqlnd` to establish the connection to 192.168.2.29.
- `Mysqlnd` calls its connect method, which has been subclassed by `ext/mysqlnd_plugin`.
- `ext/mysqlnd_plugin` calls the userspace hook `proxy::connect()` registered by the user.
- The userspace hook changes the connection host IP from 192.168.2.29 to 127.0.0.1 and returns the connection established by `parent::connect()`.
- `ext/mysqlnd_plugin` performs the equivalent of `parent::connect(127.0.0.1)` by calling the original `mysqlnd` method for establishing a connection.
- `ext/mysqlnd` establishes a connection and returns to `ext/mysqlnd_plugin`. `ext/mysqlnd_plugin` returns as well.
- Whatever PHP MySQL extension had been used by the application, it receives a connection to 127.0.0.1. The PHP MySQL extension itself returns to the PHP application. The circle is closed.

22.9.5.7.5. Getting started building a `mysqlnd` plugin

Copyright 1997-2012 the PHP Documentation Group. [3186]

It is important to remember that a `mysqlnd` plugin is itself a PHP extension.

The following code shows the basic structure of the `MINIT` function that will be used in the typical `mysqlnd` plugin:

```
/* my_php_mysqlnd_plugin.c */

static PHP_MINIT_FUNCTION(mysqlnd_plugin) {
    /* globals, ini entries, resources, classes */

    /* register mysqlnd plugin */
    mysqlnd_plugin_id = mysqlnd_plugin_register();

    conn_m = mysqlnd_get_conn_methods();
    memcpy(org_conn_m, conn_m,
           sizeof(struct st_mysqlnd_conn_methods));

    conn_m->query = MYSQLND_METHOD(mysqlnd_plugin_conn, query);
    conn_m->connect = MYSQLND_METHOD(mysqlnd_plugin_conn, connect);
}
```

```
/* my_mysqlnd_plugin.c */
```

```
enum_func_status MYSQLND_METHOD(mysqlnd_plugin_conn, query)(/* ... */) {
    /* ... */
}
enum_func_status MYSQLND_METHOD(mysqlnd_plugin_conn, connect)(/* ... */) {
    /* ... */
}
```

Task analysis: from C to userspace

```
class proxy extends mysqlnd_plugin_connection {
    public function connect($host, ...) { .. }
}
mysqlnd_plugin_set_conn_proxy(new proxy());
```

Process:

1. PHP: user registers plugin callback
2. PHP: user calls any PHP MySQL API to connect to MySQL
3. C: ext/*mysql* calls mysqlnd method
4. C: mysqlnd ends up in ext/mysqlnd_plugin
5. C: ext/mysqlnd_plugin
 - a. Calls userspace callback
 - b. Or original [mysqlnd](#) method, if userspace callback not set

You need to carry out the following:

1. Write a class "mysqlnd_plugin_connection" in C
2. Accept and register proxy object through "mysqlnd_plugin_set_conn_proxy()"
3. Call userspace proxy methods from C (optimization - zend_interfaces.h)

Userspace object methods can either be called using [call_user_function\(\)](#) or you can operate at a level closer to the Zend Engine and use [zend_call_method\(\)](#).

Optimization: calling methods from C using zend_call_method

The following code snippet shows the prototype for the [zend_call_method](#) function, taken from [zend_interfaces.h](#).

```
ZEND_API zval* zend_call_method(
    zval **object_pp, zend_class_entry *obj_ce,
    zend_function **fn_proxy, char *function_name,
    int function_name_len, zval **retval_ptr_ptr,
    int param_count, zval* arg1, zval* arg2 TSRMLS_DC
);
```

Zend API supports only two arguments. You may need more, for example:

```
enum_func_status (*func_mysqlnd_conn_connect)(
    MYSQLND *conn, const char *host,
    const char * user, const char * passwd,
```

```

unsigned int passwd_len, const char * db,
unsigned int db_len, unsigned int port,
const char * socket, unsigned int mysql_flags TSRMLS_DC
);

```

To get around this problem you will need to make a copy of `zend_call_method()` and add a facility for additional parameters. You can do this by creating a set of `MY_ZEND_CALL_METHOD_WRAPPER` macros.

Calling PHP userspace

This code snippet shows the optimized method for calling a userspace function from C:

```

/* my_mysqlnd_plugin.c */

MYSQLND_METHOD(my_conn_class,connect)(
    MYSQLND *conn, const char *host /* ... */ TSRMLS_DC) {
    enum_func_status ret = FAIL;
    zval * global_user_conn_proxy = fetch_userspace_proxy();
    if (global_user_conn_proxy) {
        /* call userspace proxy */
        ret = MY_ZEND_CALL_METHOD_WRAPPER(global_user_conn_proxy, host, /*...*/);
    } else {
        /* or original mysqlnd method = do nothing, be transparent */
        ret = org_methods.connect(conn, host, user, passwd,
            passwd_len, db, db_len, port,
            socket, mysql_flags TSRMLS_CC);
    }
    return ret;
}

```

Calling userspace: simple arguments

```

/* my_mysqlnd_plugin.c */

MYSQLND_METHOD(my_conn_class,connect)(
    /* ... */, const char *host, /* ... */) {
    /* ... */
    if (global_user_conn_proxy) {
        /* ... */
        zval* zv_host;
        MAKE_STD_ZVAL(zv_host);
        ZVAL_STRING(zv_host, host, 1);
        MY_ZEND_CALL_METHOD_WRAPPER(global_user_conn_proxy, zv_retval, zv_host /*, ...*/);
        zval_ptr_dtor(&zv_host);
        /* ... */
    }
    /* ... */
}

```

Calling userspace: structs as arguments

```

/* my_mysqlnd_plugin.c */

MYSQLND_METHOD(my_conn_class, connect)(
    MYSQLND *conn, /* ... */) {
    /* ... */
    if (global_user_conn_proxy) {
        /* ... */
        zval* zv_conn;
        ZEND_REGISTER_RESOURCE(zv_conn, (void *)conn, le_mysqlnd_plugin_conn);
        MY_ZEND_CALL_METHOD_WRAPPER(global_user_conn_proxy, zv_retval, zv_conn, zv_host /*, ...*/);
        zval_ptr_dtor(&zv_conn);
    }
}

```

```

        /* ... */
    }
    /* ... */
}

```

The first argument of many [mysqlnd](#) methods is a C "object". For example, the first argument of the `connect()` method is a pointer to `MYSQLND`. The struct `MYSQLND` represents a [mysqlnd](#) connection object.

The [mysqlnd](#) connection object pointer can be compared to a standard I/O file handle. Like a standard I/O file handle a [mysqlnd](#) connection object shall be linked to the userspace using the PHP resource variable type.

From C to userspace and back

```

class proxy extends mysqlnd_plugin_connection {
    public function connect($conn, $host, ...) {
        /* "pre" hook */
        printf("Connecting to host = '%s'\n", $host);
        debug_print_backtrace();
        return parent::connect($conn);
    }

    public function query($conn, $query) {
        /* "post" hook */
        $ret = parent::query($conn, $query);
        printf("Query = '%s'\n", $query);
        return $ret;
    }
}
mysqlnd_plugin_set_conn_proxy(new proxy());

```

PHP users must be able to call the parent implementation of an overwritten method.

As a result of subclassing it is possible to refine only selected methods and you can choose to have "pre" or "post" hooks.

Buildin class: `mysqlnd_plugin_connection::connect()`

```

/* my_mysqlnd_plugin_classes.c */

PHP_METHOD("mysqlnd_plugin_connection", connect) {
    /* ... simplified! ... */
    zval* mysqlnd_rsrc;
    MYSQLND* conn;
    char* host; int host_len;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "rs",
        &mysqlnd_rsrc, &host, &host_len) == FAILURE) {
        RETURN_NULL();
    }
    ZEND_FETCH_RESOURCE(conn, MYSQLND* conn, &mysqlnd_rsrc, -1,
        "Mysqlnd Connection", le_mysqlnd_plugin_conn);
    if (PASS == org_methods.connect(conn, host, /* simplified! */ TSRMLS_CC))
        RETVAL_TRUE;
    else
        RETVAL_FALSE;
}

```

22.9.6. Mysqlnd replication and load balancing plugin ([mysqlnd_ms](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqlnd` replication and load balancing plugin ([mysqlnd_ms](#)) adds easy to use MySQL replication support to all PHP MySQL extensions that use `mysqlnd`.

As of version PHP 5.3.3 the MySQL native driver for PHP (`mysqlnd`) features an internal plugin C API. C plugins, such as the replication and load balancing plugin, can extend the functionality of `mysqlnd`.

The MySQL native driver for PHP is a C library that ships together with PHP as of PHP 5.3.0. It serves as a drop-in replacement for the MySQL Client Library (`libmysql/libmysqlclient`). Using `mysqlnd` has several advantages: no extra downloads are required because it's bundled with PHP, it's under the PHP license, there is lower memory consumption in certain cases, and it contains new functionality such as asynchronous queries.

`mysqlnd` plugins like `mysqlnd_ms` operate, for the most part, transparently from a user perspective. The replication and load balancing plugin supports all PHP applications, and all MySQL PHP extensions. It does not change existing APIs. Therefore, it can easily be used with existing PHP applications.

22.9.6.1. Key Features

Copyright 1997-2012 the PHP Documentation Group. [3186]

The key features of `PECL/mysqlnd_ms` are as follows.

- Transparent and therefore easy to use.
 - Supports all of the PHP MySQL extensions.
 - SSL support.
 - A consistent API.
 - Little to no application changes required, dependent on the required usage scenario.
 - Lazy connections: connections to master and slave servers are not opened before a SQL statement is executed.
 - Optional: automatic use of master after the first write in a web request, to lower the possible impact of replication lag.
- Can be used with any MySQL clustering solution.
 - MySQL Replication: Read-write splitting is done by the plugin. Primary focus of the plugin.
 - MySQL Cluster: Read-write splitting can be disabled. Configuration of multiple masters possible
 - Third-party solutions: the plugin is optimized for MySQL Replication but can be used with any other kind of MySQL clustering solution.
- Featured read-write split strategies
 - Automatic detection of SELECT.
 - Supports SQL hints to overrule automatism.
 - User-defined.
 - Can be disabled for, for example, when using synchronous clusters such as MySQL Cluster.
- Featured load balancing strategies

- Round Robin: choose a different slave in round-robin fashion for every slave request.
- Random: choose a random slave for every slave request.
- Random once (sticky): choose a random slave once to run all slave requests for the duration of a web request.
- User-defined. The application can register callbacks with `mysqLnd_ms`.
- PHP 5.4.0 or newer: transaction aware when using API calls only to control transactions.
- Weighted load balancing: servers can be assigned different priorities, for example, to direct more requests to a powerful machine than to another less powerful machine. Or, to prefer nearby machines to reduce latency.
- Global transaction ID
 - Client-side emulation. Makes manual master server failover and slave promotion easier with asynchronous clusters, such as MySQL Replication.
 - Support for built-in global transaction identifier feature of MySQL 5.6.5-m8 or newer.
 - Supports using transaction ids to identify up-to-date asynchronous slaves for reading when session consistency is required.
 - Throttling: optionally, the plugin can wait for a slave to become "synchronous" before continuing.
- Service and consistency levels
 - Applications can request eventual, session and strong consistency service levels for connections. Appropriate cluster nodes will be searched automatically.
 - Eventual consistent MySQL Replication slave accesses can be replaced with fast local cache accesses transparently to reduce server load.

22.9.6.2. Limitations

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The built-in read-write-split mechanism is very basic. Every query which starts with `SELECT` is considered a read request to be sent to a MySQL slave server. All other queries (such as `SHOW` statements) are considered as write requests that are sent to the MySQL master server. The build-in behavior can be overruled using [SQL hints](#), or a user-defined [callback function](#).

The read-write splitter is not aware of multi-statements. Multi-statements are considered as one statement. The decision of where to run the statement will be based on the beginning of the statement string. For example, if using `mysqli_multi_query` to execute the multi-statement `SELECT id FROM test ; INSERT INTO test(id) VALUES (1)`, the statement will be redirected to a slave server because it begins with `SELECT`. The `INSERT` statement, which is also part of the multi-statement, will not be redirected to a master server.



Note

Applications must be aware of the consequences of connection switches that are performed for load balancing purposes. Please check the documentation on [connection pooling and switching](#), [transaction handling](#), [failover load balancing](#) and [read-write splitting](#).

22.9.6.3. On the name

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The shortcut `mysqlnd_ms` stands for `mysqlnd master slave plugin`. The name was chosen for a quick-and-dirty proof-of-concept. In the beginning the developers did not expect to continue using the code base.

22.9.6.4. Quickstart and Examples

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The mysqlnd replication load balancing plugin is easy to use. This quickstart will demo typical use-cases, and provide practical advice on getting started.

It is strongly recommended to read the reference sections in addition to the quickstart. The quickstart tries to avoid discussing theoretical concepts and limitations. Instead, it will link to the reference sections. It is safe to begin with the quickstart. However, before using the plugin in mission critical environments we urge you to read additionally the background information from the reference sections.

The focus is on using PECL `mysqlnd_ms` for work with an asynchronous MySQL cluster, namely MySQL replication. Generally speaking an asynchronous cluster is more difficult to use than a synchronous one. Thus, users of, for example, MySQL Cluster will find more information than needed.

22.9.6.4.1. Setup

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The plugin is implemented as a PHP extension. See also the [installation instructions](#) to install the [PECL/mysqlnd_ms](#) extension.

Compile or configure the PHP MySQL extension (API) ([mysqli](#), [PDO_MYSQL](#), [mysql](#)) that you plan to use with support for the `mysqlnd` library. PECL/`mysqlnd_ms` is a plugin for the `mysqlnd` library. To use the plugin with any of the PHP MySQL extensions, the extension has to use the `mysqlnd` library.

Then, load the extension into PHP and activate the plugin in the PHP configuration file using the PHP configuration directive named `mysqlnd_ms.enable`.

Example 22.215. Enabling the plugin (php.ini)

```
mysqlnd_ms.enable=1
mysqlnd_ms.ini_file=/path/to/mysqlnd_ms_plugin.ini
```

The plugin uses its own configuration file. Use the PHP configuration directive `mysqlnd_ms.ini_file` to set the full file path to the plugin-specific configuration file. This file must be readable by PHP (e.g., the web server user).

Create a plugin-specific configuration file. Save the file to the path set by the PHP configuration directive `mysqlnd_ms.ini_file`.

The plugins [configuration file](#) is JSON based. It is divided into one or more sections. Each section has a name, for example, `myapp`. Every section makes its own set of configuration settings.

A section must, at a minimum, list the MySQL replication master server, and set a list of slaves. The plugin supports using only one master server per section. Multi-master MySQL replication setups are not yet fully

supported. Use the configuration setting [master](#) to set the hostname, and the port or socket of the MySQL master server. MySQL slave servers are configured using the [slave](#) keyword.

Example 22.216. Minimal plugin-specific configuration file ([mysqlnd_ms_plugin.ini](#))

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": [
    ]
  }
}
```

Configuring a MySQL slave server list is required, although it may contain an empty list. It is recommended to always configure at least one slave server.

Server lists can use [anonymous or non-anonymous syntax \[3582\]](#). Non-anonymous lists include alias names for the servers, such as `master_0` for the master in the above example. The quickstart uses the more verbose non-anonymous syntax.

Example 22.217. Recommended minimal plugin-specific config ([mysqlnd_ms_plugin.ini](#))

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      }
    }
  }
}
```

If there are at least two servers in total, the plugin can start to load balance and switch connections. Switching connections is not always transparent and can cause issues in certain cases. The reference sections about [connection pooling and switching](#), [transaction handling](#), [fail over load balancing](#) and [read-write splitting](#) all provide more details. And potential pitfalls are described later in this guide.

It is the responsibility of the application to handle potential issues caused by connection switches, by configuring a master with at least one slave server, which allows switching to work therefore related problems can be found.

The MySQL master and MySQL slave servers, which you configure, do not need to be part of MySQL replication setup. For testing purpose you can use single MySQL server and make it known to the plugin as a master and slave server as shown below. This could help you to detect many potential issues with connection switches. However, such a setup will not be prone to the issues caused by replication lag.

Example 22.218. Using one server as a master and as a slave (testing only!)

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    }
  }
}
```

22.9.6.4.2. Running statements

Copyright 1997-2012 the PHP Documentation Group. [3186]

The plugin can be used with any PHP MySQL extension ([mysqli](#), [mysql](#), and [PDO_MYSQL](#)) that is compiled to use the [mysqlnd](#) library. PECL/mysqlnd_ms plugs into the [mysqlnd](#) library. It does not change the API or behavior of those extensions.

Whenever a connection to MySQL is being opened, the plugin compares the host parameter value of the connect call, with the section names from the plugin specific configuration file. If, for example, the plugin specific configuration file has a section [myapp](#) then the section should be referenced by opening a MySQL connection to the host [myapp](#)

Example 22.219. Plugin specific configuration file (mysqlnd_ms_plugin.ini)

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      }
    }
  }
}
```

Example 22.220. Opening a load balanced connection

```
<?php
/* Load balanced following "myapp" section rules from the plugins config file */
$mysqli = new mysqli("myapp", "username", "password", "database");
$pdo = new PDO('mysql:host=myapp;dbname=database', 'username', 'password');
```

```
$mysql = mysql_connect("myapp", "username", "password");
?>
```

The connection examples above will be load balanced. The plugin will send read-only statements to the MySQL slave server with the IP `192.168.2.27` and will listen on port `3306` for the MySQL client connection. All other statements will be directed to the MySQL master server running on the host `localhost`. If on Unix like operating systems, the master on `localhost` will be accepting MySQL client connections on the Unix domain socket `/tmp/mysql.sock`, while TCP/IP is the default port on Windows. The plugin will use the user name `username` and the password `password` to connect to any of the MySQL servers listed in the section `myapp` of the plugins configuration file. Upon connect, the plugin will select `database` as the current schemata.

The username, password and schema name are taken from the connect API calls and used for all servers. In other words: you must use the same username and password for every MySQL server listed in a plugin configuration file section. This is not a general limitation. As of PECL/mysqlnd_ms 1.1.0, it is possible to set the `username` [3584] and `password` [3584] for any server in the plugins configuration file, to be used instead of the credentials passed to the API call.

The plugin does not change the API for running statements. [Read-write splitting](#) works out of the box. The following example assumes that there is no significant replication lag between the master and the slave.

Example 22.221. Executing statements

```
<?php
/* Load balanced following "myapp" section rules from the plugins config file */
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Statements will be run on the master */
if (!$mysqli->query("DROP TABLE IF EXISTS test")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
if (!$mysqli->query("CREATE TABLE test(id INT)")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
if (!$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}

/* read-only: statement will be run on a slave */
if (!$res = $mysqli->query("SELECT id FROM test")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
} else {
    $row = $res->fetch_assoc();
    $res->close();
    printf("Slave returns id = '%s'\n", $row['id']);
}
$mysqli->close();
?>
```

The above example will output something similar to:

```
Slave returns id = '1'
```

22.9.6.4.3. Connection state

Copyright 1997-2012 the PHP Documentation Group. [3186]

The plugin changes the semantics of a PHP MySQL connection handle. A new connection handle represents a connection pool, instead of a single MySQL client-server network connection. The connection pool consists of a master connection, and optionally any number of slave connections.

Every connection from the connection pool has its own state. For example, SQL user variables, temporary tables and transactions are part of the state. For a complete list of items that belong to the state of a connection, see the [connection pooling and switching](#) concepts documentation. If the plugin decides to switch connections for load balancing, the application could be given a connection which has a different state. Applications must be made aware of this.

Example 22.222. Plugin config with one slave and one master

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp/mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      }
    }
  }
}
```

Example 22.223. Pitfall: connection state and SQL user variables

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
  /* Of course, your error handling is nicer... */
  die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Connection 1, connection bound SQL user variable, no SELECT thus run on master */
if (!$mysqli->query("SET @myrole='master'")) {
  printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}

/* Connection 2, run on slave because SELECT */
if (!$res = $mysqli->query("SELECT @myrole AS _role")) {
  printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
} else {
  $row = $res->fetch_assoc();
  $res->close();
  printf("@myrole = '%s'\n", $row['_role']);
}
$mysqli->close();
?>
```

The above example will output:

```
@myrole = ''
```

The example opens a load balanced connection and executes two statements. The first statement `SET @myrole='master'` does not begin with the string `SELECT`. Therefore the plugin does not recognize it as a read-only query which shall be run on a slave. The plugin runs the statement on the connection to the master. The statement sets a SQL user variable which is bound to the master connection. The state of the master connection has been changed.

The next statement is `SELECT @myrole AS _role`. The plugin does recognize it as a read-only query and sends it to the slave. The statement is run on a connection to the slave. This second connection does not have any SQL user variables bound to it. It has a different state than the first connection to the master. The requested SQL user variable is not set. The example script prints `@myrole = ''`.

It is the responsibility of the application developer to take care of the connection state. The plugin does not monitor all connection state changing activities. Monitoring all possible cases would be a very CPU intensive task, if it could be done at all.

The pitfalls can easily be worked around using SQL hints.

22.9.6.4.4. SQL Hints

Copyright 1997-2012 the PHP Documentation Group. [3186]

SQL hints can force a query to choose a specific server from the connection pool. It gives the plugin a hint to use a designated server, which can solve issues caused by connection switches and connection state.

SQL hints are standard compliant SQL comments. Because SQL comments are supposed to be ignored by SQL processing systems, they do not interfere with other programs such as the MySQL Server, the MySQL Proxy, or a firewall.

Three SQL hints are supported by the plugin: The `MYSQLND_MS_MASTER_SWITCH` hint makes the plugin run a statement on the master, `MYSQLND_MS_SLAVE_SWITCH` enforces the use of the slave, and `MYSQLND_MS_LAST_USED_SWITCH` will run a statement on the same server that was used for the previous statement.

The plugin scans the beginning of a statement for the existence of an SQL hint. SQL hints are only recognized if they appear at the beginning of the statement.

Example 22.224. Plugin config with one slave and one master

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      }
    }
  }
}
```

```
}

```

Example 22.225. SQL hints to prevent connection switches

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Connection 1, connection bound SQL user variable, no SELECT thus run on master */
if (!$mysqli->query("SET @myrole='master'")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}

/* Connection 1, run on master because of SQL hint */
if (!($res = $mysqli->query(sprintf("/*%s*/SELECT @myrole AS _role", MYSQLND_MS_LAST_USED_SWITCH)))) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
} else {
    $row = $res->fetch_assoc();
    $res->close();
    printf("@myrole = '%s'\n", $row['_role']);
}
$mysqli->close();
?>
```

The above example will output:

```
@myrole = 'master'
```

In the above example, using [MYSQLND_MS_LAST_USED_SWITCH](#) prevents session switching from the master to a slave when running the [SELECT](#) statement.

SQL hints can also be used to run [SELECT](#) statements on the MySQL master server. This may be desired if the MySQL slave servers are typically behind the master, but you need current data from the cluster.

In version 1.2.0 the concept of a service level has been introduced to address cases when current data is required. Using a service level requires less attention and removes the need of using SQL hints for this use case. Please, find more information below in the service level and consistency section.

Example 22.226. Fighting replication lag

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Force use of master, master has always fresh and current data */
if (!$mysqli->query(sprintf("/*%s*/SELECT critical_data FROM important_table", MYSQLND_MS_MASTER_SWITCH))) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
?>
```

A use case may include the creation of tables on a slave. If an SQL hint is not given, then the plugin will send `CREATE` and `INSERT` statements to the master. Use the SQL hint `MYSQLND_MS_SLAVE_SWITCH` if you want to run any such statement on a slave, for example, to build temporary reporting tables.

Example 22.227. Table creation on a slave

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Force use of slave */
if (!$mysqli->query(sprintf("/*%s*/CREATE TABLE slave_reporting(id INT)", MYSQLND_MS_SLAVE_SWITCH))) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* Continue using this particular slave connection */
if (!$mysqli->query(sprintf("/*%s*/INSERT INTO slave_reporting(id) VALUES (1), (2), (3)", MYSQLND_MS_LAST_USED))) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* Don't use MYSQLND_MS_SLAVE_SWITCH which would allow switching to another slave! */
if ($res = $mysqli->query(sprintf("/*%s*/SELECT COUNT(*) AS _num FROM slave_reporting", MYSQLND_MS_LAST_USED))) {
    $row = $res->fetch_assoc();
    $res->close();
    printf("There are %d rows in the table 'slave_reporting'", $row['_num']);
} else {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
$mysqli->close();
?>
```

The SQL hint `MYSQLND_MS_LAST_USED` forbids switching a connection, and forces use of the previously used connection.

22.9.6.4.5. Transactions

Copyright 1997-2012 the PHP Documentation Group. [3186]

The current version of the plugin is not transaction safe by default, because it is not aware of running transactions in all cases. SQL transactions are units of work to be run on a single server. The plugin does not always know when the unit of work starts and when it ends. Therefore, the plugin may decide to switch connections in the middle of a transaction.

No kind of MySQL load balancer can detect transaction boundaries without any kind of hint from the application.

You can either use SQL hints to work around this limitation. Alternatively, you can activate transaction API call monitoring. In the latter case you must use API calls only to control transactions, see below.

Example 22.228. Plugin config with one slave and one master

```
[myapp]
{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost",
                "socket": "\tmp\mysql.sock"
```

```

    }
    },
    "slave": {
        "slave_0": {
            "host": "192.168.2.27",
            "port": "3306"
        }
    }
}
}
}

```

Example 22.229. Using SQL hints for transactions

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Not a SELECT, will use master */
if (!$mysqli->query("START TRANSACTION")) {
    /* Please use better error handling in your code */
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}

/* Prevent connection switch! */
if (!$mysqli->query(sprintf("/*%s*/INSERT INTO test(id) VALUES (1)", MYSQLND_MS_LAST_USED_SWITCH))) {
    /* Please do proper ROLLBACK in your code, don't just die */
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if ($res = $mysqli->query(sprintf("/*%s*/SELECT COUNT(*) AS _num FROM test", MYSQLND_MS_LAST_USED_SWITCH))) {
    $row = $res->fetch_assoc();
    $res->close();
    if ($row['_num'] > 1000) {
        if (!$mysqli->query(sprintf("/*%s*/INSERT INTO events(task) VALUES ('cleanup')", MYSQLND_MS_LAST_USED_SWITCH))) {
            die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
        }
    }
} else {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if (!$mysqli->query(sprintf("/*%s*/UPDATE log SET last_update = NOW()", MYSQLND_MS_LAST_USED_SWITCH))) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if (!$mysqli->query(sprintf("/*%s*/COMMIT", MYSQLND_MS_LAST_USED_SWITCH))) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}

$mysqli->close();
?>

```

Starting with PHP 5.4.0, the `mysqlnd` library allows the plugin to monitor the status of the `autocommit` mode, if the mode is set by API calls instead of using SQL statements such as `SET AUTOCOMMIT=0`. This makes it possible for the plugin to become transaction aware. In this case, you do not need to use SQL hints.

If using PHP 5.4.0 or newer, API calls that enable `autocommit` mode, and when setting the plugin configuration option `trx_stickiness=master`, the plugin can automatically disable load balancing and connection switches for SQL transactions. In this configuration, the plugin stops load balancing if `autocommit` is disabled and directs all statements to the master. This prevents connection switches in

the middle of a transaction. Once `autocommit` is re-enabled, the plugin starts to load balance statements again.

Example 22.230. Transaction aware load balancing: `trx_stickiness` setting

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    },
    "trx_stickiness": "master"
  }
}
```

Example 22.231. Transaction aware

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
  /* Of course, your error handling is nicer... */
  die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Disable autocommit, plugin will run all statements on the master */
$mysqli->autocommit(FALSE);

if (!$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
  /* Please do proper ROLLBACK in your code, don't just die */
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if ($res = $mysqli->query("SELECT COUNT(*) AS _num FROM test")) {
  $row = $res->fetch_assoc();
  $res->close();
  if ($row['_num'] > 1000) {
    if (!$mysqli->query("INSERT INTO events(task) VALUES ('cleanup')")) {
      die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
  }
} else {
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if (!$mysqli->query("UPDATE log SET last_update = NOW()")) {
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if (!$mysqli->commit()) {
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}

/* Plugin assumes that the transaction has ended and starts load balancing again */
$mysqli->autocommit(TRUE);
$mysqli->close();
?>
```



Version requirement

The plugin configuration option [trx_stickiness=master](#) requires PHP 5.4.0 or newer.

22.9.6.4.6. Service level and consistency

Copyright 1997-2012 the PHP Documentation Group. [3186]



Version requirement

Service levels have been introduced in PECL [mysqlnd_ms](#) version 1.2.0-alpha. [mysqlnd_ms_set_qos](#) is available with PHP 5.4.0 or newer.

Different types of MySQL cluster solutions offer different service and data consistency levels to their users. An asynchronous MySQL replication cluster offers eventual consistency by default. A read executed on an asynchronous slave may return current, stale or no data at all, depending on whether the slave has replayed all changesets from the master or not.

Applications using an MySQL replication cluster need to be designed to work correctly with eventual consistent data. In some cases, however, stale data is not acceptable. In those cases only certain slaves or even only master accesses are allowed to achieve the required quality of service from the cluster.

As of PECL [mysqlnd_ms](#) 1.2.0 the plugin is capable of selecting MySQL replication nodes automatically that deliver session consistency or strong consistency. Session consistency means that one client can read its writes. Other clients may or may not see the clients' write. Strong consistency means that all clients will see all writes from the client.

Example 22.232. Session consistency: read your writes

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    }
  }
}
```

Example 22.233. Requesting session consistency

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
  /* Of course, your error handling is nicer... */
  die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* read-write splitting: master used */
if (!$mysqli->query("INSERT INTO orders(order_id, item) VALUES (1, 'christmas tree, 1.8m')")) {
  /* Please use better error handling in your code */
}
```

```

    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}

/* Request session consistency: read your writes */
if (!$mysqli->set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Plugin picks a node which has the changes, here: master */
if (!$res = $mysqli->query("SELECT item FROM orders WHERE order_id = 1"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

var_dump($res->fetch_assoc());

/* Back to eventual consistency: stale data allowed */
if (!$mysqli->set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Plugin picks any slave, stale data is allowed */
if (!$res = $mysqli->query("SELECT item, price FROM specials"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
?>

```

Service levels can be set in the plugins configuration file and at runtime using `mysqlnd_ms_set_qos`. In the example the function is used to enforce session consistency (read your writes) for all future statements until further notice. The `SELECT` statement on the `orders` table is run on the master to ensure the previous write can be seen by the client. Read-write splitting logic has been adapted to fulfill the service level.

After the application has read its changes from the `orders` table it returns to the default service level, which is eventual consistency. Eventual consistency puts no restrictions on choosing a node for statement execution. Thus, the `SELECT` statement on the `specials` table is executed on a slave.

The new functionality supersedes the use of SQL hints and the `master_on_write` configuration option. In many cases `mysqlnd_ms_set_qos` is easier to use, more powerful improves portability.

Example 22.234. Maximum age/slave lag

```

{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost",
                "socket": "\tmp\mysql.sock"
            }
        },
        "slave": {
            "slave_0": {
                "host": "127.0.0.1",
                "port": "3306"
            }
        },
        "failover" : "master"
    }
}

```

Example 22.235. Limiting slave lag

```
<?php
```

```

$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Read from slaves lagging no more than four seconds */
$ret = mysqlnd_ms_set_qos($mysqli,
    MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL,
    MYSQLND_MS_QOS_OPTION_AGE, 4);

if (!$ret)
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Plugin picks any slave, which may or may not have the changes */
if (!$res = $mysqli->query("SELECT item, price FROM daytrade"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Back to default: use of all slaves and masters permitted */
if (!mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
?>

```

The eventual consistency service level can be used with an optional parameter to set a maximum slave lag for choosing slaves. If set, the plugin checks `SHOW SLAVE STATUS` for all configured slaves. In case of the example, only slaves for which `Slave_IO_Running=Yes`, `Slave_SQL_Running=Yes` and `Seconds_Behind_Master <= 4` is true are considered for executing the statement `SELECT item, price FROM daytrade`.

Checking `SHOW SLAVE STATUS` is done transparently from an applications perspective. Errors, if any, are reported as warnings. No error will be set on the connection handle. Even if all `SHOW SLAVE STATUS` SQL statements executed by the plugin fail, the execution of the users statement is not stopped, given that master fail over is enabled. Thus, no application changes are required.



Expensive and slow operation

Checking `SHOW SLAVE STATUS` for all slaves adds overhead to the application. It is an expensive and slow background operation. Try to minimize the use of it. Unfortunately, a MySQL replication cluster does not give clients the possibility to request a list of candidates from a central instance. Thus, a more efficient way of checking the slaves lag is not available.

Please, note the limitations and properties of `SHOW SLAVE STATUS` as explained in the MySQL reference manual.

To prevent `mysqlnd_ms` from emitting a warning if no slaves can be found that lag no more than the defined number of seconds behind the master, it is necessary to enable master fail over in the plugins configuration file. If no slaves can be found and fail over is turned on, the plugin picks a master for executing the statement.

If no slave can be found and fail over is turned off, the plugin emits a warning, it does not execute the statement and it sets an error on the connection.

Example 22.236. Fail over not set

```

{
  "myapp": {
    "master": {

```

```

        "master_0": {
            "host": "localhost",
            "socket": "\tmp/mysql.sock"
        },
        "slave": {
            "slave_0": {
                "host": "127.0.0.1",
                "port": "3306"
            }
        }
    }
}

```

Example 22.237. No slave within time limit

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Read from slaves lagging no more than four seconds */
$ret = mysqlnd_ms_set_qos($mysqli,
    MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL,
    MYSQLND_MS_QOS_OPTION_AGE, 4);

if (!$ret)
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Plugin picks any slave, which may or may not have the changes */
if (!$res = $mysqli->query("SELECT item, price FROM daytrade"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Back to default: use of all slaves and masters permitted */
if (!$mysqli->set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
?>

```

The above example will output:

```

PHP Warning: mysqli::query(): (mysqlnd_ms) Couldn't find the appropriate slave connection. 0 slaves to ch
PHP Warning: mysqli::query(): (mysqlnd_ms) No connection selected by the last filter in %s on line %d
[2000] (mysqlnd_ms) No connection selected by the last filter

```

22.9.6.4.7. Global transaction IDs

Copyright 1997-2012 the PHP Documentation Group. [3186]



Version requirement

A client-side global transaction ID injection has been introduced in `mysqlnd_ms` version 1.2.0-alpha. The feature is not required for synchronous clusters, such as MySQL Cluster. Use it with asynchronous clusters such as classical MySQL replication.

As of MySQL 5.6.5-m8 the MySQL server features built-in global transaction identifiers. The MySQL built-in global transaction ID feature is supported by PECL/`mysqlnd_ms` 1.3.0-alpha or later.

PECL/`mysqlnd_ms` can either use its own global transaction ID emulation or the global transaction ID feature built-in to MySQL 5.6.5-m8 or later. From a developer perspective the client-side and server-side approach offer the same features with regards to service levels provided by PECL/`mysqlnd_ms`. Their differences are discussed in the [concepts section](#).

The quickstart first demonstrates the use of the client-side global transaction ID emulation built-in to PECL/`mysqlnd_ms` before its show how to use the server-side counterpart. The order ensures that the underlying idea is discussed first.

Idea and client-side emulation

In its most basic form a global transaction ID (GTID) is a counter in a table on the master. The counter is incremented whenever a transaction is committed on the master. Slaves replicate the table. The counter serves two purposes. In case of a master failure, it helps the database administrator to identify the most recent slave for promoting it to the new master. The most recent slave is the one with the highest counter value. Applications can use the global transaction ID to search for slaves which have replicated a certain write (identified by a global transaction ID) already.

PECL/`mysqlnd_ms` can inject SQL for every committed transaction to increment a GTID counter. The so created GTID is accessible by the application to identify an applications write operation. This enables the plugin to deliver session consistency (read your writes) service level by not only querying masters but also slaves which have replicated the change already. Read load is taken away from the master.

Client-side global transaction ID emulation has some limitations. Please, read the [concepts section](#) carefully to fully understand the principles and ideas behind it, before using in production environments. The background knowledge is not required to continue with the quickstart.

First, create a counter table on your master server and insert a record into it. The plugin does not assist creating the table. Database administrators must make sure it exists. Depending on the error reporting mode, the plugin will silently ignore the lack of the table or bail out.

Example 22.238. Create counter table on master

```
CREATE TABLE `trx` (
  `trx_id` int(11) DEFAULT NULL,
  `last_update` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1
INSERT INTO `trx`(`trx_id`) VALUES (1);
```

In the plugins configuration file set the SQL to update the global transaction ID table using `on_commit` from the [global_transaction_id_injection](#) section. Make sure the table name used for the `UPDATE` statement is fully qualified. In the example, `test.trx` is used to refer to table `trx` in the schema (database) `test`. Use the table that was created in the previous step. It is important to set the fully qualified table name because the connection on which the injection is done may use a different default database. Make sure the user that opens the connection is allowed to execute the `UPDATE`.

Enable reporting of errors that may occur when `mysqlnd_ms` does global transaction ID injection.

Example 22.239. Plugin config: SQL for client-side GTID injection

```
{
```

```

"myapp": {
  "master": {
    "master_0": {
      "host": "localhost",
      "socket": "\tmp\mysql.sock"
    }
  },
  "slave": {
    "slave_0": {
      "host": "127.0.0.1",
      "port": "3306"
    }
  },
  "global_transaction_id_injection":{
    "on_commit":"UPDATE test.trx SET trx_id = trx_id + 1",
    "report_error":true
  }
}
}

```

Example 22.240. Transparent global transaction ID injection

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
  /* Of course, your error handling is nicer... */
  die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("DROP TABLE IF EXISTS test"))
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("CREATE TABLE test(id INT)"))
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("INSERT INTO test(id) VALUES (1)"))
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* auto commit mode, read on slave, no increment */
if (!$res = $mysqli->query("SELECT id FROM test"))
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

var_dump($res->fetch_assoc());
?>

```

The above example will output:

```

array(1) {
  ["id"]=>
  string(1) "1"
}

```

The example runs three statements in auto commit mode on the master, causing three transactions on the master. For every such statement, the plugin will inject the configured `UPDATE` transparently before executing the users SQL statement. When the script ends the global transaction ID counter on the master has been incremented by three.

The fourth SQL statement executed in the example, a `SELECT`, does not trigger an increment. Only transactions (writes) executed on a master shall increment the GTID counter.



SQL for global transaction ID: efficient solution wanted!

The SQL used for the client-side global transaction ID emulation is inefficient. It is optimized for clarity not for performance. Do not use it for production environments. Please, help finding an efficient solution for inclusion in the manual. We appreciate your input.

Example 22.241. Plugin config: SQL for fetching GTID

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    },
    "global_transaction_id_injection":{
      "on_commit":"UPDATE test.trx SET trx_id = trx_id + 1",
      "fetch_last_gtid" : "SELECT MAX(trx_id) FROM test.trx",
      "report_error":true
    }
  }
}
```

Example 22.242. Obtaining GTID after injection

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
  /* Of course, your error handling is nicer... */
  die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("DROP TABLE IF EXISTS test"))
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

printf("GTID after transaction %s\n", mysqlnd_ms_get_last_gtid($mysqli));

/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("CREATE TABLE test(id INT)"))
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

printf("GTID after transaction %s\n", mysqlnd_ms_get_last_gtid($mysqli));
?>
```

The above example will output:

```
GTID after transaction 7
```

GTID after transaction 8

Applications can ask PECL `mysqld_ms` for a global transaction ID which belongs to the last write operation performed by the application. The function `mysqld_ms_get_last_gtid` returns the GTID obtained when executing the SQL statement from the `fetch_last_gtid` entry of the `global_transaction_id_injection` section from the plugins configuration file. The function may be called after the GTID has been incremented.

Applications are advised not to run the SQL statement themselves as this bares the risk of accidentally causing an implicit GTID increment. Also, if the function is used, it is easy to migrate an application from one SQL statement for fetching a transaction ID to another, for example, if any MySQL server ever features built-in global transaction ID support.

The quickstart shows a SQL statement which will return a GTID equal or greater to that created for the previous statement. It is exactly the GTID created for the previous statement if no other clients have incremented the GTID in the time span between the statement execution and the `SELECT` to fetch the GTID. Otherwise, it is greater.

Example 22.243. Plugin config: Checking for a certain GTID

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    },
    "global_transaction_id_injection": {
      "on_commit": "UPDATE test.trx SET trx_id = trx_id + 1",
      "fetch_last_gtid" : "SELECT MAX(trx_id) FROM test.trx",
      "check_for_gtid" : "SELECT trx_id FROM test.trx WHERE trx_id >= #GTID",
      "report_error": true
    }
  }
}
```

Example 22.244. Session consistency service level and GTID combined

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
  /* Of course, your error handling is nicer... */
  die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)"))
  die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
```

```

/* GTID as an identifier for the last write */
$gtid = mysqlnd_ms_get_last_gtid($mysqli);

/* Session consistency (read your writes): try to read from slaves not only master */
if (false == mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION, MYSQLND_MS_QOS_OPTION_GTID, $gtid)
    die(sprintf("[006] [%d] %s\n", $mysqli->errno, $mysqli->error));
}

/* Either run on master or a slave which has replicated the INSERT */
if (!$res = $mysqli->query("SELECT id FROM test")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}

var_dump($res->fetch_assoc());
?>

```

A GTID returned from `mysqlnd_ms_get_last_gtid` can be used as an option for the session consistency service level. Session consistency delivers read your writes. Session consistency can be requested by calling `mysqlnd_ms_set_qos`. In the example, the plugin will execute the `SELECT` statement either on the master or on a slave which has replicated the previous `INSERT` already.

PECL `mysqlnd_ms` will transparently check every configured slave if it has replicated the `INSERT` by checking the slaves GTID table. The check is done running the SQL set with the `check_for_gtid` option from the `global_transaction_id_injection` section of the plugins configuration file. Please note, that this is a slow and expensive procedure. Applications should try to use it sparsely and only if read load on the master becomes to high otherwise.

Use of the server-side global transaction ID feature

Starting with MySQL 5.6.5-m8 the MySQL Replication system features server-side global transaction IDs. Transaction identifiers are automatically generated and maintained by the server. Users do not need to take care of maintaining them. There is no need to setup any tables in advance, or for setting `on_commit`. A client-side emulation is no longer needed.

Clients can continue to use global transaction identifier to achieve session consistency when reading from MySQL Replication slaves. The algorithm works as described above. Different SQL statements must be configured for `fetch_last_gtid` and `check_for_gtid`. The statements are given below. Please note, MySQL 5.6.5-m8 is a development version. Details of the server implementation may change in the future and require adoption of the SQL statements shown.

Using the following configuration any of the above described functionality can be used together with the server-side global transaction ID feature. `mysqlnd_ms_get_last_gtid` and `mysqlnd_ms_set_qos` continue to work as described above. The only difference is that the server does not use a simple sequence number but a string containing of a server identifier and a sequence number. Thus, users cannot easily derive an order from GTIDs returned by `mysqlnd_ms_get_last_gtid`.

Example 22.245. Plugin config: using MySQL 5.6.5-m8 built-in GTID feature

```

{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {

```

```

        "host": "127.0.0.1",
        "port": "3306"
    },
    },
    "global_transaction_id_injection":{
        "fetch_last_gtid" : "SELECT @@GLOBAL.GTID_DONE AS trx_id FROM DUAL",
        "check_for_gtid" : "SELECT GTID_SUBSET('#GTID', @@GLOBAL.GTID_DONE) AS trx_id FROM DUAL",
        "report_error":true
    }
}
}
}

```

22.9.6.4.8. Cache integration

Copyright 1997-2012 the PHP Documentation Group. [3186]



Version requirement and dependencies

Cache integration is available as of PECL/`mysqlnd_ms` 1.3.0-beta (under development). [PECL/`mysqlnd_qc` 1.1.0](#) (under development) or newer is used as a cache. Both plugins must be installed. PECL/`mysqlnd_ms` must be compiled to support the cache feature. Use of PHP 5.4.0 or newer is mandatory.

Databases clusters can deliver different levels of consistency. As of PECL/`mysqlnd_ms` 1.2.0 it is possible to advice the plugin to consider only cluster nodes that can deliver the consistency level requested. For example, if using asynchronous MySQL Replication with its cluster-wide eventual consistency, it is possible to request session consistency (read your writes) at any time using `mysqlnd_ms_set_quos`. Please, see also the [service level and consistency](#) introduction.

Example 22.246. Recap: quality of service to request read your writes

```

/* Request session consistency: read your writes */
if (!mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

```

Assuming PECL/`mysqlnd` has been explicitly told to deliver no consistency level higher than eventual consistency, it is possible to replace a database node read access with a client-side cache using time-to-live (TTL) as its invalidation strategy. Both the database node and the cache may or may not serve current data as this is what eventual consistency defines.

Replacing a database node read access with a local cache access can improve overall performance and lower the database load. If the cache entry is every reused by other clients than the one creating the cache entry, a database access is saved and thus database load is lowered. Furthermore, system performance can become better if computation and delivery of a database query is slower than a local cache access.

Example 22.247. Plugin config: no special entries for caching

```

{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost",
                "socket": "\tmp\mysql.sock"
            }
        },
        "slave": {
            "slave_0": {
                "host": "127.0.0.1",

```

```

        "port": "3306"
    }
},
}
}

```

Example 22.248. Caching a slave request

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Explicitly allow eventual consistency and caching (TTL <= 60 seconds) */
if (false == mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL, MYSQLND_MS_QOS_OPTION_CACHE, 60))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}

/* To make this example work, we must wait for a slave to catch up. Brute force style. */
$attempts = 0;
do {
    /* check if slave has the table */
    if ($res = $mysqli->query("SELECT id FROM test")) {
        break;
    } else if ($mysqli->errno) {
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    /* wait for slave to catch up */
    usleep(200000);
} while ($attempts++ < 10);

/* Query has been run on a slave, result is in the cache */
assert($res);
var_dump($res->fetch_assoc());

/* Served from cache */
$res = $mysqli->query("SELECT id FROM test");
?>

```

The example shows how to use the cache feature. First, you have to set the quality of service to eventual consistency and explicitly allow for caching. This is done by calling `mysqlnd_ms_set_qos`. Then, the result set of every read-only statement is cached for upto that many seconds as allowed with `mysqlnd_ms_set_qos`.

The actual TTL is lower or equal to the value set with `mysqlnd_ms_set_qos`. The value passed to the function sets the maximum age (seconds) of the data delivered. To calculate the actual TTL value the replication lag on a slave is checked and subtracted from the given value. If, for example, the maximum age is set to 60 seconds and the slave reports a lag of 10 seconds the resulting TTL is 50 seconds. The TTL is calculated individually for every cached query.

Example 22.249. Read your writes and caching combined

```

<?php

```

```

$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Explicitly allow eventual consistency and caching (TTL <= 60 seconds) */
if (false == mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL, MYSQLND_MS_QOS_OPTION_CACHE,
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}

/* To make this example work, we must wait for a slave to catch up. Brute force style. */
$attempts = 0;
do {
    /* check if slave has the table */
    if ($res = $mysqli->query("SELECT id FROM test")) {
        break;
    } else if ($mysqli->errno) {
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    /* wait for slave to catch up */
    usleep(200000);
} while ($attempts++ < 10);

assert($res);

/* Query has been run on a slave, result is in the cache */
var_dump($res->fetch_assoc());

/* Served from cache */
if (!($res = $mysqli->query("SELECT id FROM test")))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
var_dump($res->fetch_assoc());

/* Update on master */
if (!$mysqli->query("UPDATE test SET id = 2"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Read your writes */
if (false == mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION)) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}

/* Fetch latest data */
if (!($res = $mysqli->query("SELECT id FROM test")))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
var_dump($res->fetch_assoc());
?>
    
```

The quality of service can be changed at any time to avoid further cache usage. If needed, you can switch to read your writes (session consistency). In that case, the cache will not be used and fresh data is read.

22.9.6.4.9. Failover

Copyright 1997-2012 the PHP Documentation Group. [3186]

By default, the plugin does not attempt to fail over if connecting to a host fails. This prevents pitfalls related to [connection state](#). It is recommended to manually handle connection errors in a way similar to a failed transaction. You should catch the error, rebuild the connection state and rerun your query as shown below.

If connection state is no issue to you, you can alternatively enable automatic and silent failover. Depending on the configuration, the automatic and silent failover will either attempt to fail over to the master before issuing an error or, try to connect to other slaves, given the query allows for it, before attempting to connect to a master.

Example 22.250. Manual failover, automatic optional

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp/mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "simulate_slave_failure",
        "port": "0"
      },
      "slave_1": {
        "host": "127.0.0.1",
        "port": 3311
      }
    },
    "filters": { "roundrobin": [] }
  }
}
```

Example 22.251. Manual failover

```
<?php
$link = new mysqli("myapp", "username", "password", "database");
if (!$link)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

$sql = "SELECT 1 FROM DUAL";

/* error handling as it should be done regardless of the plugin */
if (!$res = $link->query($sql)) {
    /* plugin specific: check for connection error */
    switch ($link->errno) {
        case 2002:
        case 2003:
        case 2005:
            printf("Connection error - trying next slave!\n");
            /* load balancer will pick next slave */
            $res = $link->query($sql);
            break;
        default:
            /* no connection error, failover is unlikely to help */
            die(sprintf("SQL error: [%d] %s", $link->errno, $link->error));
            break;
    }
}
if ($res) {
    var_dump($res->fetch_assoc());
}
?>
```

22.9.6.5. Concepts

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

This explains the architecture and related concepts for this plugin, and describes the impact that MySQL replication and this plugin have on developmental tasks while using a database cluster. Reading and understanding these concepts is required, in order to use this plugin with success.

22.9.6.5.1. Architecture

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The mysqlnd replication and load balancing plugin is implemented as a PHP extension. It is written in C and operates under the hood of PHP. During the startup of the PHP interpreter, in the module init phase of the PHP engine, it gets registered as a [mysqlnd](#) plugin to replace selected mysqlnd C methods.

At PHP runtime, it inspects queries sent from mysqlnd (PHP) to the MySQL server. If a query is recognized as read-only, it will be sent to one of the configured slave servers. Statements are considered read-only if they either start with [SELECT](#), the SQL hint `/*ms=slave*/` or a slave had been chosen for running the previous query, and the query started with the SQL hint `/*ms=last_used*/`. In all other cases, the query will be sent to the MySQL replication master server.

For better portability, applications should use the [MYSQLND_MS_MASTER_SWITCH](#), [MYSQLND_MS_SLAVE_SWITCH](#), and [MYSQLND_MS_LAST_USED_SWITCH](#) [predefined mysqlnd_ms constants](#), instead of their literal values, such as `/*ms=slave*/`.

The plugin handles the opening and closing of database connections to both master and slave servers. From an application point of view, there continues to be only one connection handle. However, internally, this one public connection handle represents a pool of network connections that are managed by the plugin. The plugin proxies queries to the master server, and to the slaves using multiple connections.

Database connections have a state consisting of, for example, transaction status, transaction settings, character set settings, and temporary tables. The plugin will try to maintain the same state among all internal connections, whenever this can be done in an automatic and transparent way. In cases where it is not easily possible to maintain state among all connections, such as when using [BEGIN TRANSACTION](#), the plugin leaves it to the user to handle.

22.9.6.5.2. Connection pooling and switching

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The replication and load balancing plugin changes the semantics of a PHP MySQL connection handle. The existing API of the PHP MySQL extensions ([mysqli](#), [mysql](#), and [PDO_MYSQL](#)) are not changed in a way that functions are added or removed. But their behaviour changes when using the plugin. Existing applications do not need to be adapted to a new API, but they may need to be modified because of the behaviour changes.

The plugin breaks the one-by-one relationship between a [mysqli](#), [mysql](#), and [PDO_MYSQL](#) connection handle and a MySQL network connection. And a [mysqli](#), [mysql](#), and [PDO_MYSQL](#) connection handle represents a local pool of connections to the configured MySQL replication master and MySQL replication slave servers. The plugin redirects queries to the master and slave servers. At some point in time one and the same PHP connection handle may point to the MySQL master server. Later on, it may point to one of the slave servers or still the master. Manipulating and replacing the network connection referenced by a PHP MySQL connection handle is not a transparent operation.

Every MySQL connection has a state. The state of the connections in the connection pool of the plugin can differ. Whenever the plugin switches from one wire connection to another, the current state of the user connection may change. The applications must be aware of this.

The following list shows what the connection state consists of. The list may not be complete.

- Transaction status
- Temporary tables
- Table locks
- Session system variables and session user variables
- The current database set using `USE` and other state chaining SQL commands
- Prepared statements
- `HANDLER` variables
- Locks acquired with `GET_LOCK()`

Connection switches happen right before queries are executed. The plugin does not switch the current connection until the next statement is executed.



Replication issues

See also the MySQL reference manual chapter about [replication features](#) and related issues. Some restrictions may not be related to the PHP plugin, but are properties of the MySQL replication system.

Broadcasted messages

The plugins philosophy is to align the state of connections in the pool only if the state is under full control of the plugin, or if it is necessary for security reasons. Just a few actions that change the state of the connection fall into this category.

The following is a list of connection client library calls that change state, and are broadcasted to all open connections in the connection pool.

If any of the listed calls below are to be executed, the plugin loops over all open master and slave connections. The loop continues until all servers have been contacted, and the loop does not break if a server indicates a failure. If possible, the failure will propagate to the called user API function, which may be detected depending on which underlying library function was triggered.

Library call	Notes	Version
<code>change_user</code>	Called by the <code>mysqli_change_user</code> user API call. Also triggered upon reuse of a persistent <code>mysqli</code> connection.	Since 1.0.0.
<code>select_db</code>	Called by the following user API calls: <code>mysql_select_db</code> , <code>mysqli_list_tables</code> , <code>mysql_db_query</code> , <code>mysqli_list_fields</code> , <code>mysqli_select_db</code> . Note, that SQL <code>USE</code> is not monitored.	Since 1.0.0.
<code>set_charset</code>	Called by the following user API calls: <code>mysql_set_charset</code> , <code>mysqli_set_charset</code> . Note, that SQL <code>SET NAMES</code> is not monitored.	Since 1.0.0.

Library call	Notes	Version
<code>set_server</code>	Called by the following user API calls: <code>mysqli_multi_query</code> , <code>mysqli_real_query</code> , <code>mysqli_query</code> , <code>mysql_query</code> .	Since 1.0.0.
<code>set_client</code>	Called by the following user API calls: <code>mysqli_options</code> , <code>mysqli_ssl_set</code> , <code>mysqli_connect</code> , <code>mysql_connect</code> , <code>mysql_pconnect</code> .	Since 1.0.0.
<code>set_auto</code>	Called by the following user API calls: <code>mysqli_autocommit</code> , <code>PDO::setAttribute(PDO::ATTR_AUTOCOMMIT)</code> .	Since 1.0.0. PHP >= 5.4.0.
<code>ssl_set</code>	Called by the following user API calls: <code>mysqli_ssl_set</code> .	Since 1.1.0.

Broadcasting and lazy connections

The plugin does not proxy or “remember” all settings to apply them on connections opened in the future. This is important to remember, if using [lazy connections](#). Lazy connections are connections which are not opened before the client sends the first connection. Use of lazy connections is the default plugin action.

The following connection library calls each changed state, and their execution is recorded for later use when lazy connections are opened. This helps ensure that the connection state of all connections in the connection pool are comparable.

Library call	Notes	Version
<code>change_user</code>	User, password and database recorded for future use.	Since 1.1.0.
<code>select_db</code>	Database recorded for future use.	Since 1.1.0.
<code>set_charset</code>	Calls <code>set_client_option(MYSQL_SET_CHARSET_NAME, charset)</code> on lazy connection to ensure <code>charset</code> will be used upon opening the lazy connection.	Since 1.1.0.
<code>set_auto</code>	Adds <code>SET AUTOCOMMIT=0 1</code> to the list of init commands of a lazy connection using <code>set_client_option(MYSQL_INIT_COMMAND, "SET AUTOCOMMIT=...%quot;)</code> .	Since 1.1.0. PHP >= 5.4.0.



Connection state

The connection state is not only changed by API calls. Thus, even if PECL `mysqlnd_ms` monitors all API calls, the application must still be aware. Ultimately, it is the applications responsibility to maintain the connection state, if needed.

Charsets and string escaping

Due to the use of lazy connections, which are a default, it can happen that an application tries to escape a string for use within SQL statements before a connection has been established. In this case string escaping is not possible. The string escape function does not know what charset to use before a connection has been established.

To overcome the problem a new configuration setting `server_charset` has been introduced in version 1.4.0.

Attention has to be paid on escaping strings with a certain charset but using the result on a connection that uses a different charset. Please note, that PECL/`mysqlnd_ms` manipulates connections and one application level connection represents a pool of multiple connections that all may have different default

charsets. It is recommended to configure the servers involved to use the same default charsets. The configuration setting `server_charset` does help with this situation as well. If using `server_charset`, the plugin will set the given charset on all newly opened connections.

22.9.6.5.3. Transaction handling

Copyright 1997-2012 the PHP Documentation Group. [3186]

Transaction handling is fundamentally changed. An SQL transaction is a unit of work that is run on one database server. The unit of work consists of one or more SQL statements.

By default the plugin is not aware of SQL transactions. The plugin may switch connections for load balancing at any point in time. Connection switches may happen in the middle of a transaction. This is against the nature of an SQL transaction. By default, the plugin is not transaction safe.

Any kind of MySQL load balancer must be hinted about the begin and end of a transaction. Hinting can either be done implicitly by monitoring API calls or using SQL hints. Both options are supported by the plugin, depending on your PHP version. API monitoring requires PHP 5.4.0 or newer. The plugin, like any other MySQL load balancer, cannot detect transaction boundaries based on the MySQL Client Server Protocol. Thus, entirely transparent transaction aware load balancing is not possible. The least intrusive option is API monitoring, which requires little to no application changes, depending on your application.

Please, find examples of using SQL hints or the API monitoring in the [examples section](#). The details behind the API monitoring, which makes the plugin transaction aware, are described below.

Beginning with PHP 5.4.0, the `mysqlnd` library allows this plugin to subclass the library C API call `set_autocommit()`, to detect the status of `autocommit` mode.

The PHP MySQL extensions either issue a query (such as `SET AUTOCOMMIT=0|1`), or use the `mysqlnd` library call `set_autocommit()` to control the `autocommit` setting. If an extension makes use of `set_autocommit()`, the plugin can be made transaction aware. Transaction awareness cannot be achieved if using SQL to set the `autocommit` mode. The library function `set_autocommit()` is called by the `mysqli_autocommit` and `PDO::setAttribute(PDO::ATTR_AUTOCOMMIT)` user API calls.

The plugin configuration option `trx_stickiness=master` can be used to make the plugin transactional aware. In this mode, the plugin stops load balancing if `autocommit` becomes disabled, and directs all statements to the master until `autocommit` gets enabled.

An application that does not want to set SQL hints for transactions but wants to use the transparent API monitoring to avoid application changes must make sure that the `autocommit` settings is changed exclusively through the listed API calls.

22.9.6.5.4. Error handling

Copyright 1997-2012 the PHP Documentation Group. [3186]

Applications using PECL/`mysqlnd_ms` should implement proper error handling for all user API calls. And because the plugin changes the semantics of a connection handle, API calls may return unexpected errors. If using the plugin on a connection handle that no longer represents an individual network connection, but a connection pool, an error code and error message will be set on the connection handle whenever an error occurs on any of the network connections behind.

If using lazy connections, which is the default, connections are not opened until they are needed for query execution. Therefore, an API call for a statement execution may return a connection error. In the example below, an error is provoked when trying to run a statement on a slave. Opening a slave connection fails because the plugin configuration file lists an invalid host name for the slave.

Example 22.252. Provoking a connection error

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp/mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "invalid_host_name",
      }
    },
    "lazy_connections": 1
  }
}
```

The explicit activation of lazy connections is for demonstration purpose only.

Example 22.253. Connection error on query execution

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Connection 1, connection bound SQL user variable, no SELECT thus run on master */
if (!$mysqli->query("SET @myrole='master'")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}

/* Connection 2, run on slave because SELECT, provoke connection error */
if (!$res = $mysqli->query("SELECT @myrole AS _role")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
} else {
    $row = $res->fetch_assoc();
    $res->close();
    printf("@myrole = '%s'\n", $row['_role']);
}
$mysqli->close();
?>
```

The above example will output something similar to:

```
PHP Warning: mysqli::query(): php_network_getaddresses: getaddrinfo failed: Name or service not known in :
PHP Warning: mysqli::query(): [2002] php_network_getaddresses: getaddrinfo failed: Name or service not known
[2002] php_network_getaddresses: getaddrinfo failed: Name or service not known
```

Applications are expected to handle possible connection errors by implementing proper error handling.

Depending on the use case, applications may want to handle connection errors differently from other errors. Typical connection errors are [2002 \(CR_CONNECTION_ERROR\) - Can't connect to local MySQL server through socket '%s' \(%d\)](#), [2003 \(CR_CONN_HOST_ERROR\) - Can't connect](#)

to MySQL server on '%s' (%d) and 2005 (CR_UNKNOWN_HOST) - Unknown MySQL server host '%s' (%d). For example, the application may test for the error codes and manually perform a fail over. The plugins philosophy is not to offer automatic fail over, beyond master fail over, because fail over is not a transparent operation.

Example 22.254. Provoking a connection error

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "invalid_host_name"
      },
      "slave_1": {
        "host": "192.168.78.136"
      }
    },
    "lazy_connections": 1,
    "filters": {
      "roundrobin": [
      ]
    }
  }
}
```

Explicitly activating lazy connections is done for demonstration purposes, as is round robin load balancing as opposed to the default `random once` type.

Example 22.255. Most basic failover

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Connection 1, connection bound SQL user variable, no SELECT thus run on master */
if (!$mysqli->query("SET @myrole='master'")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}

/* Connection 2, first slave */
$res = $mysqli->query("SELECT VERSION() AS _version");
/* Hackish manual fail over */
if (2002 == $mysqli->errno || 2003 == $mysqli->errno || 2004 == $mysqli->errno) {
    /* Connection 3, first slave connection failed, trying next slave */
    $res = $mysqli->query("SELECT VERSION() AS _version");
}

if (!$res) {
    printf("ERROR, [%d] '%s'\n", $mysqli->errno, $mysqli->error);
} else {
    /* Error messages are taken from connection 3, thus no error */
    printf("SUCCESS, [%d] '%s'\n", $mysqli->errno, $mysqli->error);
    $row = $res->fetch_assoc();
}
```

```
$res->close();
printf("version = %s\n", $row['_version']);
}
$mysqli->close();
?>
```

The above example will output something similar to:

```
[1045] Access denied for user 'username'@'localhost' (using password: YES)
PHP Warning: mysqli::query(): php_network_getaddresses: getaddrinfo failed: Name or service not known in :
PHP Warning: mysqli::query(): [2002] php_network_getaddresses: getaddrinfo failed: Name or service not known in :
SUCCESS, [0] ''
version = 5.6.2-m5-log
```

In some cases, it may not be easily possible to retrieve all errors that occur on all network connections through a connection handle. For example, let's assume a connection handle represents a pool of three open connections. One connection to a master and two connections to the slaves. The application changes the current database using the user API call `mysqli_select_db`, which then calls the `mysqLnd` library function to change the schemata. `mysqLnd_ms` monitors the function, and tries to change the current database on all connections to harmonize their state. Now, assume the master succeeds in changing the database, and both slaves fail. Upon the initial error from the first slave, the plugin will set an appropriate error on the connection handle. The same is done when the second slave fails to change the database. The error message from the first slave is lost.

Such cases can be debugged by either checking for errors of the type `E_WARNING` (see above) or, if no other option, investigation of the [mysqLnd_ms debug and trace log](#).

22.9.6.5.5. Failover

Copyright 1997-2012 the PHP Documentation Group. [3186]

By default, connection failover handling is left to the user. The application is responsible for checking return values of the database functions it calls and reacting to possible errors. If, for example, the plugin recognizes a query as a read-only query to be sent to the slave servers, and the slave server selected by the plugin is not available, the plugin will raise an error after not executing the statement.

Default: manual failover

It is up to the application to handle the error and, if required, re-issue the query to trigger the selection of another slave server for statement execution. The plugin will make no attempts to failover automatically, because the plugin cannot ensure that an automatic failover will not change the state of the connection. For example, the application may have issued a query which depends on SQL user variables which are bound to a specific connection. Such a query might return incorrect results if the plugin would switch the connection implicitly as part of automatic failover. To ensure correct results, the application must take care of the failover, and rebuild the required connection state. Therefore, by default, no automatic failover is performed by the plugin.

A user that does not change the connection state after opening a connection may activate automatic failover. Please note, that automatic failover logic is limited to connection attempts. Automatic failover is not used for already established connections. There is no way to instruct the plugin to attempt failover on a connection that has been connected to MySQL already in the past.

Automatic failover

The failover policy is configured in the plugins configuration file, by using the [failover](#) configuration directive.

Automatic and silent failover can be enabled through the [failover](#) configuration directive. Automatic failover can either be configured to try exactly one master after a slave failure or, alternatively, loop over slaves and masters before returning an error to the user. The number of connection attempts can be limited and failed hosts can be excluded from future load balancing attempts. Limiting the number of retries and remembering failed hosts are considered experimental features, albeit being reasonable stable. Syntax and semantics may change in future versions.

A basic manual failover example is provided within the [error handling](#) section.

Standby servers

Using [weighted load balancing](#), introduced in PECL/mysqlnd 1.4.0, it is possible to configure standby servers that are sparsely used during normal operations. A standby server that is primarily used as a worst-case standby failover target can be assigned a very low weight/priority in relation to all other servers. As long as all servers are up and running the majority of the workload is assigned to the servers which have high weight values. Few requests will be directed to the standby system which has a very low weight value.

Upon failure of the servers with a high priority, you can still failover to the standby, which has been given a low load balancing priority by assigning a low weight to it. Failover can be done manually or automatically. If done automatically, you may want to combine it with the [remember_failed](#) option.

At this point, it is not possible to instruct the load balancer to direct no requests at all to a standby. This may not be much of a limitation given that the highest weight you can assign to a server is 65535. Given two slaves, of which one shall act as a standby and has been assigned a weight of 1, the standby will have to handle far less than one percent of the overall workload.

Failover and primary copy

Please note, if using a primary copy cluster, such as MySQL Replication, it is difficult to do connection failover in case of a master failure. At any time there is only one master in the cluster for a given dataset. The master is a single point of failure. If the master fails, clients have no target to fail over write requests. In case of a master outage the database administrator must take care of the situation and update the client configurations, if need be.

22.9.6.5.6. Load balancing

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

Four load balancing strategies are supported to distribute statements over the configured MySQL slave servers:

random	Chooses a random server whenever a statement is executed.
random once (default)	Chooses a random server after the first statement is executed, and uses the decision for the rest of the PHP request.
	It is the default, and the lowest impact on the connection state.
round robin	Iterates over the list of configured servers.
user-defined via callback	Is used to implement any other strategy.

The load balancing policy is configured in the plugins configuration file using the [random](#), [roundrobin](#), and [user filters](#).

Servers can be prioritized assigning a weight. A server that has been given a weight of two will get twice as many requests as a server that has been given the default weight of one. Prioritization can be handy in heterogenous environments. For example, you may want to assign more requests to a powerful machine than to a less powerful. Or, you may have configured servers that are close or far from the client, thus expose different latencies.

22.9.6.5.7. Read-write splitting

Copyright 1997-2012 the PHP Documentation Group. [3186]

The plugin executes read-only statements on the configured MySQL slaves, and all other queries on the MySQL master. Statements are considered read-only if they either start with `SELECT`, the SQL hint `/*ms=slave*/`, or if a slave had been chosen for running the previous query and the query starts with the SQL hint `/*ms=last_used*/`. In all other cases, the query will be sent to the MySQL replication master server. It is recommended to use the constants `MYSQLND_MS_SLAVE_SWITCH`, `MYSQLND_MS_MASTER_SWITCH` and `MYSQLND_MS_LAST_USED_SWITCH` instead of `/*ms=slave*/`. See also the [list of mysqlnd_ms constants](#).

SQL hints are a special kind of standard compliant SQL comments. The plugin does check every statement for certain SQL hints. The SQL hints are described within the [mysqlnd_ms constants](#) documentation, constants that are exported by the extension. Other systems involved with the statement processing, such as the MySQL server, SQL firewalls, and SQL proxies, are unaffected by the SQL hints, because those systems are designed to ignore SQL comments.

The built-in read-write splitter can be replaced by a user-defined filter, see also the [user filter](#) documentation.

A user-defined read-write splitter can request the built-in logic to send a statement to a specific location, by invoking `mysqlnd_ms_is_select`.



Note

The built-in read-write splitter is not aware of multi-statements. Multi-statements are seen as one statement. The splitter will check the beginning of the statement to decide where to run the statement. If, for example, a multi-statement begins with `SELECT 1 FROM DUAL; INSERT INTO test(id) VALUES (1); ...` the plugin will run it on a slave although the statement is not read-only.

22.9.6.5.8. Filter

Copyright 1997-2012 the PHP Documentation Group. [3186]



Version requirement

Filters exist as of `mysqlnd_ms` version 1.1.0-beta.

[filters](#). PHP applications that implement a MySQL replication cluster must first identify a group of servers in the cluster which could execute a statement before the statement is executed by one of the candidates. In other words: a defined list of servers must be filtered until only one server is available.

The process of filtering may include using one or more filters, and filters can be chained. And they are executed in the order they are defined in the plugins configuration file.



Explanation: comparing filter chaining to pipes

The concept of chained filters can be compared to using pipes to connect command line utilities on an operating system command shell. For example, an input stream is passed to a processor, filtered, and then transferred to be output. Then, the output is passed as input to the next command, which is connected to the previous using the pipe operator.

Available filters:

- Load balancing filters: `random` and `roundrobin`.
- Selection filter: `user`, `user_multi`, `quality_of_service`.

The `random` filter implements the 'random' and 'random once' load balancing policies. The 'round robin' load balancing can be configured through the `roundrobin` filter. Setting a 'user defined callback' for server selection is possible with the `user` filter. The `quality_of_service` filter finds cluster nodes capable of delivering a certain service, for example, read-your-writes or, not lagging more seconds behind the master than allowed.

Filters can accept parameters to change their behaviour. The `random` filter accepts an optional `sticky` parameter. If set to true, the filter changes load balancing from random to random once. Random picks a random server every time a statement is to be executed. Random once picks a random server when the first statement is to be executed and uses the same server for the rest of the PHP request.

One of the biggest strength of the filter concept is the possibility to chain filters. This strength does not become immediately visible because the `random`, `roundrobin` and `user` filters are supposed to output no more than one server. If a filter reduces the list of candidates for running a statement to only one server, it makes little sense to use that one server as input for another filter for further reduction of the list of candidates.

An example filter sequence that will fail:

- Statement to be executed: `SELECT 1 FROM DUAL`. Passed to all filters.
- All configured nodes are passed as input to the first filter. Master nodes: `master_0`. Slave nodes: `slave_0`, `slave_1`
- Filter: `random`, argument `sticky=1`. Picks a random slave once to be used for the rest of the PHP request. Output: `slave_0`.
- Output of `slave_0` and the statement to be executed is passed as input to the next filter. Here: `roundrobin`, server list passed to filter is: `slave_0`.
- Filter: `roundrobin`. Server list consists of one server only, round robin will always return the same server.

If trying to use such a filter sequence, the plugin may emit a warning like `(mysqlnd_ms) Error while creating filter '%s' . Non-multi filter '%s' already created. Stopping in %s on line %d`. Furthermore, an appropriate error on the connection handle may be set.

A second type of filter exists: multi filter. A multi filter emits zero, one or multiple servers after processing. The `quality_of_service` filter is an example. If the service quality requested sets an upper limit for the slave lag and more than one slave is lagging behind less than the allowed number of seconds, the filter returns more than one cluster node. A multi filter must be followed by other to further reduce the list of candidates for statement execution until a candidate is found.

A filter sequence with the `quality_of_service` multi filter followed by a load balancing filter.

- Statement to be executed: `SELECT sum(price) FROM orders WHERE order_id = 1`. Passed to all filters.
- All configured nodes are passed as input to the first filter. Master nodes: `master_0`. Slave nodes: `slave_0, slave_1, slave_2, slave_3`
- Filter: `quality_of_service`, rule set: `session_consistency` (read-your-writes) Output: `master_0`
- Output of `master_0` and the statement to be executed is passed as input to the next filter, which is `roundrobin`.
- Filter: `roundrobin`. Server list consists of one server. Round robin selects `master_0`.

A filter sequence must not end with a multi filter. If trying to use a filter sequence which ends with a multi filter the plugin may emit a warning like `(mysqlnd_ms) Error in configuration. Last filter is multi filter. Needs to be non-multi one. Stopping in %s on line %d`. Furthermore, an appropriate error on the connection handle may be set.



Speculation towards the future: MySQL replication filtering

In future versions, there may be additional multi filters. For example, there may be a `table` filter to support MySQL replication filtering. This would allow you to define rules for which database or table is to be replicated to which node of a replication cluster. Assume your replication cluster consists of four slaves (`slave_0, slave_1, slave_2, slave_3`) two of which replicate a database named `sales` (`slave_0, slave_1`). If the application queries the database `sales`, the hypothetical `table` filter reduces the list of possible servers to `slave_0` and `slave_1`. Because the output and list of candidates consists of more than one server, it is necessary and possible to add additional filters to the candidate list, for example, using a load balancing filter to identify a server for statement execution.

22.9.6.5.9. Service level and consistency

Copyright 1997-2012 the PHP Documentation Group. [3186]



Version requirement

Service levels have been introduced in `mysqlnd_ms` version 1.2.0-alpha. `mysqlnd_ms_set_qos` requires PHP 5.4.0 or newer.

The plugin can be used with different kinds of MySQL database clusters. Different clusters can deliver different levels of service to applications. The service levels can be grouped by the data consistency levels that can be achieved. The plugin knows about:

- eventual consistency
- session consistency
- strong consistency

Depending how a cluster is used it may be possible to achieve higher service levels than the default one. For example, a read from an asynchronous MySQL replication slave is eventual consistent. Thus, one may say the default consistency level of a MySQL replication cluster is eventual consistency. However, if the master only is used by a client for reading and writing during a session, session consistency (read your

writes) is given. PECL `mysqLnd` 1.2.0 abstracts the details of choosing an appropriate node for any of the above service levels from the user.

Service levels can be set through the qualify-of-service filter in the [plugins configuration file](#) and at runtime using the function `mysqLnd_ms_set_qos`.

The plugin defines the different service levels as follows.

Eventual consistency is the default service provided by an asynchronous cluster, such as classical MySQL replication. A read operation executed on an arbitrary node may or may not return stale data. The applications view of the data is eventual consistent.

Session consistency is given if a client can always read its own writes. An asynchronous MySQL replication cluster can deliver session consistency if clients always use the master after the first write or never query a slave which has not yet replicated the clients write operation.

The plugins understanding of strong consistency is that all clients always see the committed writes of all other clients. This is the default when using MySQL Cluster or any other cluster offering synchronous data distribution.

Service level parameters

Eventual consistency and session consistency service level accept parameters.

Eventual consistency is the service provided by classical MySQL replication. By default, all nodes qualify for read requests. An optional `age` parameter can be given to filter out nodes which lag more than a certain number of seconds behind the master. The plugin is using `SHOW SLAVE STATUS` to measure the lag. Please, see the MySQL reference manual to learn about accuracy and reliability of the `SHOW SLAVE STATUS` command.

Session consistency (read your writes) accepts an optional `GTID` parameter to consider reading not only from the master but also from slaves which already have replicated a certain write described by its transaction identifier. This way, when using asynchronous MySQL replication, read requests may be load balanced over slaves while still ensuring session consistency.

The latter requires the use of [client-side global transaction id injection](#).

Advantages of the new approach

The new approach supersedes the use of SQL hints and the configuration option `master_on_write` in some respects. If an application running on top of an asynchronous MySQL replication cluster cannot accept stale data for certain reads, it is easier to tell the plugin to choose appropriate nodes than prefixing all read statements in question with the SQL hint to enforce the use of the master. Furthermore, the plugin may be able to use selected slaves for reading.

The `master_on_write` configuration option makes the plugin use the master after the first write (session consistency, read your writes). In some cases, session consistency may not be needed for the rest of the session but only for some, few read operations. Thus, `master_on_write` may result in more read load on the master than necessary. In those cases it is better to request a higher than default service level only for those reads that actually need it. Once the reads are done, the application can return to default service level. Switching between service levels is only possible using `mysqLnd_ms_set_qos`.

Performance considerations

A MySQL replication cluster cannot tell clients which slaves are capable of delivering which level of service. Thus, in some cases, clients need to query the slaves to check their status. PECL `mysqLnd_ms`

transparently runs the necessary SQL in the background. However, this is an expensive and slow operation. SQL statements are run if eventual consistency is combined with an age (slave lag) limit and if session consistency is combined with a global transaction ID.

If eventual consistency is combined with an maximum age (slave lag), the plugin selects candidates for statement execution and load balancing for each statement as follows. If the statement is a write all masters are considered as candidates. Slaves are not checked and not considered as candidates. If the statement is a read, the plugin transparently executes `SHOW SLAVE STATUS` on every slaves connection. It will loop over all connections, send the statement and then start checking for results. Usually, this is slightly faster than a loop over all connections in which for every connection a query is send and the plugin waits for its results. A slave is considered a candidate if `SHOW SLAVE STATUS` reports `Slave_IO_Running=Yes`, `Slave_SQL_Running=Yes` and `Seconds_Behind_Master` is less or equal than the allowed maximum age. In case of an SQL error, the plugin emits a warning but does not set an error on the connection. The error is not set to make it possible to use the plugin as a drop-in.

If session consistency is combined with a global transaction ID, the plugin executes the SQL statement set with the `fetch_last_gtid` entry of the `global_transaction_id_injection` section from the plugins configuration file. Further details are identical to those described above.

In version 1.2.0 no additional optimizations are done for executing background queries. Future versions may contain optimizations, depending on user demand.

If no parameters and options are set, no SQL is needed. In that case, the plugin consider all nodes of the type shown below.

- eventual consistency, no further options set: all masters, all slaves
- session consistency, no further options set: all masters
- strong consistency (no options allowed): all masters

Throttling

The quality of service filter can be combed with [Global transaction IDs](#) to throttle clients. Throttling does reduce the write load on the master by slowing down clients. If session consistency is requested and global transactions identifier are used to check the status of a slave, the check can be done in two ways. By default a slave is checked and skipped immediately if it does not match the criteria for session consistency. Alternatively, the plugin can wait for a slave to catch up to the master until session consistency is possible. To enable the throttling, you have to set `wait_for_gtid_timeout` configuration option.

22.9.6.5.10. Global transaction IDs

Copyright 1997-2012 the PHP Documentation Group. [3186]



Version requirement

Client side global transaction ID injection exists as of `mysqlnd_ms` version 1.2.0-alpha. Transaction boundaries are detected by monitoring API calls. This is possible as of PHP 5.4.0. Please, see also [Transaction handling](#).

As of MySQL 5.6.5-m8 the MySQL server features built-in global transaction identifiers. The MySQL built-in global transaction ID feature is supported by PECL/`mysqlnd_ms` 1.3.0-alpha or later. Neither are client-side transaction boundary monitoring nor any setup activities required if using the server feature.

Idea and client-side emulation

PECL/mysqlnd_ms can do client-side transparent global transaction ID injection. In its most basic form, a global transaction identifier is a counter which is incremented for every transaction executed on the master. The counter is held in a table on the master. Slaves replicate the counter table.

In case of a master failure a database administrator can easily identify the most recent slave for promoting it as a new master. The most recent slave has the highest transaction identifier.

Application developers can ask the plugin for the global transaction identifier (GTID) for their last successful write operation. The plugin will return an identifier that refers to a transaction no older than that of the clients last write operation. Then, the GTID can be passed as a parameter to the quality of service (QoS) filter as an option for session consistency. Session consistency ensures read your writes. The filter ensures that all reads are either directed to a master or a slave which has replicated the write referenced by the GTID.

When injection is done

The plugin transparently maintains the GTID table on the master. In autocommit mode the plugin injects an `UPDATE` statement before executing the users statement for every master use. In manual transaction mode, the injection is done before the application calls `commit()` to close a transaction. The configuration option `report_error` of the GTID section in the plugins configuration file is used to control whether a failed injection shall abort the current operation or be ignored silently (default).

Please note, the PHP version requirements for [transaction boundary monitoring](#) and their limits.

Limitations

Client-side global transaction ID injection has shortcomings. The potential issues are not specific to PECL/mysqlnd_ms but are rather of general nature.

- Global transaction ID tables must be deployed on all masters and replicas.
- The GTID can have holes. Only PHP clients using the plugin will maintain the table. Other clients will not.
- Client-side transaction boundary detection is based on API calls only.
- Client-side transaction boundary detection does not take implicit commit into account. Some MySQL SQL statements cause an implicit commit and cannot be rolled back.

Using server-side global transaction identifier

Starting with PECL/mysqlnd_ms 1.3.0-alpha the MySQL 5.6.5-m8 or newer built-in global transaction identifier feature is supported. Use of the server feature lifts all of the above listed limitations. Please, see the MySQL Reference Manual for limitations and preconditions for using server built-in global transaction identifiers.

Whether to use the client-side emulation or the server built-in functionality is a question not directly related to the plugin, thus it is not discussed in depth. There are no plans to remove the client-side emulation and you can continue to use it, if the server-side solution is no option. This may be the case in heterogenous environments with old MySQL server or, if any of the server-side solution limitations is not acceptable.

From an applications perspective there is hardly a difference in using one or the other approach. The following properties differ.

- Client-side emulation, as shown in the manual, is using an easy to compare sequence number for global transactions. Multi-master is not handled to keep the manual examples easy.

Server-side built-in feature is using a combination of a server identifier and a sequence number as a global transaction identifier. Comparison cannot use numeric algebra. Instead a SQL function must be used. Please, see the MySQL Reference Manual for details.

- Plugin global transaction ID statistics are only available with client-side emulation because they monitor the emulation.



Global transaction identifiers in distributed systems

Global transaction identifiers can serve multiple purposes in the context of distributed systems, such as a database cluster. Global transaction identifiers can be used for, for example, system wide identification of transactions, global ordering of transactions, heartbeat mechanism and for checking the replication status of replicas. PECL/[mysqlnd_ms](#), a clientside driver based software, does focus on using GTIDs for tasks that can be handled at the client, such as checking the replication status of replicas for asynchronous replication setups.

22.9.6.5.11. Cache integration

Copyright 1997-2012 the PHP Documentation Group. [3186]



Version requirement

The feature requires used of PECL/[mysqlnd_ms](#) 1.3.0-beta or later and PECL/[mysqlnd_qc](#) 1.1.0-alpha or new. PECL/[mysqlnd_ms](#) must be compiled to support the feature. PHP 5.4.0 or newer is required.



Suitable MySQL clusters

The feature is targeted for use with MySQL Replication (primary copy). Currently, no other kinds of MySQL clusters are supported. Users of such cluster must control PECL/[mysqlnd_qc](#) manually if they are interested in client-side query caching.

Support for MySQL replication clusters (asynchronous primary copy) is the main focus of PECL/[mysqlnd_ms](#). The slaves of a MySQL replication cluster may or may not reflect the latest updates from the master. Slaves are asynchronous and can lag behind the master. A read from a slave is eventual consistent from a cluster-wide perspective.

The same level of consistency is offered by a local cache using time-to-live (TTL) invalidation strategy. Current data or stale data may be served. Eventually, data searched for in the cache is not available and the source of the cache needs to be accessed.

Given that both a MySQL Replication slave (asynchronous secondary) and a local TTL-driven cache deliver the same level of service it is possible to transparently replace a remote database access with a local cache access to gain better possibility.

As of PECL/[mysqlnd_ms](#) 1.3.0-beta the plugin is capable of transparently controlling PECL/[mysqlnd_ms](#) 1.1.0-alpha or newer to cache a read-only query if explicitly allowed by setting an appropriate quality of service through [mysqlnd_ms_set_qos](#). Please, see the [quickstart](#) for a code example. Both plugins must be installed, PECL/[mysqlnd_ms](#) must be compiled to support the cache feature and PHP 5.4.0 or newer has to be used.

Applications have full control of cache usage and can request fresh data at any time, if need be. The cache usage can be enabled and disabled time during the execution of a script. The cache will be used if [mysqlnd_ms_set_qos](#) sets the quality of service to eventual consistency and enables cache usage.

Cache usage is disabled by requesting higher consistency levels, for example, session consistency (read your writes). Once the quality of service has been relaxed to eventual consistency the cache can be used again.

If caching is enabled for a read-only statement, PECL/mysqlnd_ms may inject [SQL hints to control caching](#) by PECL/mysqlnd_qc. It may modify the SQL statement it got from the application. Subsequent SQL processors are supposed to ignore the SQL hints. A SQL hint is a SQL comment. Comments must not be ignored, for example, by the database server.

The TTL of a cache entry is computed on a per statement basis. Applications set an maximum age for the data they want to retrieve using [mysqlnd_ms_set_qos](#). The age sets an approximate upper limit of how many seconds the data returned may lag behind the master.

The following logic is used to compute the actual TTL if caching is enabled. The logic takes the estimated slave lag into account for choosing a TTL. If, for example, there are two slaves lagging 5 and 10 seconds behind and the maximum age allowed is 60 seconds, the TTL is set to 50 seconds. Please note, the age setting is no more than an estimated guess.

- Check whether the statement is read-only. If not, don't cache.
- If caching is enabled, check the slave lag of all configured slaves. Establish slave connections if none exist so far and lazy connections are used.
- Send [SHOW SLAVE STATUS](#) to all slaves. Do not wait for the first slave to reply before sending to the second slave. Clients often wait long for replies, thus we send out all requests in a burst before fetching in a second stage.
- Loop over all slaves. For every slave wait for its reply. Do not start checking another slave before the currently waited for slave has replied. Check for [Slave_IO_Running=Yes](#) and [Slave_SQL_Running=Yes](#). If both conditions hold true, fetch the value of [Seconds_Behind_Master](#). In case of any errors or if conditions fail, set an error on the slave connection. Skip any such slave connection for the rest of connection filtering.
- Search for the maximum value of [Seconds_Behind_Master](#) from all slaves that passed the previous conditions. Subtract the value from the maximum age provided by the user with [mysqlnd_ms_set_qos](#). Use the result as a TTL.
- The filtering may sort out all slaves. If so, the maximum age is used as TTL, because the maximum lag found equals zero. It is perfectly valid to sort out all slaves. In the following it is up to subsequent filter to decide what to do. The built-in load balancing filter will pick the master.
- Inject the appropriate SQL hints to enable caching by PECL/mysqlnd_qc.
- Proceed with the connection filtering, e.g. apply load balancing rules to pick a slave.
- PECL/mysqlnd_qc is loaded after PECL/mysqlnd_ms by PHP. Thus, it will see all query modifications of PECL/mysqlnd_ms and cache the query if instructed to do so.

The algorithm may seem expensive. [SHOW SLAVE STATUS](#) is a very fast operation. Given a sufficient number of requests and cache hits per second the cost of checking the slaves lag can easily outweigh the costs of the cache decision.

Suggestions on a better algorithm are always welcome.

22.9.6.5.12. Supported clusters

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

Any application using any kind of MySQL cluster is faced with the same tasks:

- Identify nodes capable of executing a given statement with the required service level
- Load balance requests within the list of candidates
- Automatic fail over within candidates, if needed

The plugin is optimized for fulfilling these tasks in the context of a classical asynchronous MySQL replication cluster consisting of a single master and many slaves (primary copy). When using classical, asynchronous MySQL replication all of the above listed tasks need to be mastered at the client side.

Other types of MySQL cluster may have lower requirements on the application side. For example, if all nodes in the cluster can answer read and write requests, no read-write splitting needs to be done (multi-master, update-all). If all nodes in the cluster are synchronous, they automatically provide the highest possible quality of service which makes choosing a node easier. In this case, the plugin may serve the application after some reconfiguration to disable certain features, such as built-in read-write splitting.



Documentation focus

The documentation focusses describing the use of the plugin with classical asynchronous MySQL replication clusters (primary copy). Support for this kind of cluster has been the original development goal. Use of other clusters is briefly described below. Please note, that this is still work in progress.

Using asynchronous clusters with single master

Primary use case of the plugin. Follow the hints given in the descriptions of each feature.



Version requirement

The following cluster may require use of settings not available before `mysqLnd_ms 1.2.0-alpha`.

Using asynchronous clusters with multiple masters

This setup is currently unsupported.

The plugin has no built-in functionality to direct certain writes to certain masters. It is considered to add table filtering to future versions. Table filter would allow restricting both read and writes to certain slaves and masters based on the database/schema and table used by a statement.

A table filtering feature is prepared in the plugins source code. However, it is instable and must not be used. Bug reports on table filtering will be rejected.

Using synchronous clusters such as MySQL Cluster

MySQL Cluster is a synchronous cluster solution. All cluster nodes accept read and write requests. In the context of the plugin, all nodes shall be considered as masters.

Use the load balancing and fail over features only.

- Disable the plugins [built-in read-write splitting](#).
- Configure masters only.
- Consider random once load balancing strategy, which is the plugins default. If random once is used, only masters are configured and no SQL hints are used to force using a certain node, no connection switches

will happen for the duration of a web request. Thus, no special handling is required for transactions. The plugin will pick one master at the beginning of the PHP script and use it until the script terminates.

- Do not set the quality of service. All nodes have all the data. This automatically gives you the highest possible service quality (strong consistency).
- Do not enable client-side global transaction injection. It is neither required to help with server-side fail over nor to assist the quality of service filter choosing an appropriate node.

Disabling built-in read-write splitting.

- Set `mysqlnd_ms.disable_rw_split=1`
- Do not use [SQL hints](#) to enforce the use of slaves

Configure masters only.

- Set `mysqlnd_ms.multi_master=1`
- Do not configure any slaves
- Set `failover=master` in the plugins configuration file to avoid warnings about the empty slave list

22.9.6.6. Installing/Configuring

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

22.9.6.6.1. Requirements

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

[PHP 5.3.6](#) or newer. Some advanced functionality requires [PHP 5.4.0](#) or newer.

The `mysqlnd_ms` replication and load balancing plugin supports all PHP applications and all available PHP MySQL extensions ([mysqli](#), [mysql](#), [PDO_MYSQL](#)). The PHP MySQL extension must be configured to use [mysqlnd](#) in order to be able to use the `mysqlnd_ms` plugin for [mysqlnd](#).

22.9.6.6.2. Installation

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

This [PECL](#) extension is not bundled with PHP.

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: http://pecl.php.net/package/mysqlnd_ms

A DLL for this PECL extension is currently unavailable. See also the [building on Windows](#) section.

22.9.6.6.3. Runtime Configuration

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The behaviour of these functions is affected by settings in `php.ini`.

Table 22.72. MySQLnd_ms Configure Options

Name	Default	Changeable	Changelog
<code>mysqlnd_ms.enable</code>	0	PHP_INI_SYSTEM	

Name	Default	Changeable	Changelog
<code>mysqlnd_ms.force_config_usage</code>	0	PHP_INI_SYSTEM	
<code>mysqlnd_ms.ini_file</code>	""	PHP_INI_SYSTEM	
<code>mysqlnd_ms.config_file</code>	""	PHP_INI_SYSTEM	
<code>mysqlnd_ms.collect_statistics</code>	0	PHP_INI_SYSTEM	
<code>mysqlnd_ms.multi_master</code>	0	PHP_INI_SYSTEM	
<code>mysqlnd_ms.disable_rw_split</code>	0	PHP_INI_SYSTEM	

Here's a short explanation of the configuration directives.

`mysqlnd_ms.enable` integer Enables or disables the plugin. If disabled, the extension will not plug into `mysqlnd` to proxy internal `mysqlnd` C API calls.

`mysqlnd_ms.force_config_usage` integer If enabled, the plugin checks if the host (server) parameters value of any MySQL connection attempt, matches a section name from the plugin configuration file. If not, the connection attempt is blocked.

`mysqlnd_ms.ini_file` string Plugin specific configuration file. This setting has been renamed to `mysqlnd_ms.config_file` in version 1.4.0.

`mysqlnd_ms.config_file` string Plugin specific configuration file. This setting supersedes `mysqlnd_ms.ini_file` since 1.4.0.

`mysqlnd_ms.collect_statistics` integer Enables or disables the collection of statistics. The collection of statistics is disabled by default for performance reasons. Statistics are returned by the function `mysqlnd_ms_get_stats`.

`mysqlnd_ms.multi_master` integer Enables or disables support of MySQL multi master replication setups. Please, see also [supported clusters](#).

`mysqlnd_ms.disable_rw_split` integer Enables or disables built-in read write splitting.

Controls whether load balancing and lazy connection functionality can be used independently of read write splitting. If read write splitting is disabled, only servers from the master list will be used for statement execution. All configured slave servers will be ignored.

The SQL hint `MYSQLND_MS_USE_SLAVE` will not be recognized. If found, the statement will be redirected to a master.

Disabling read write splitting impacts the return value of `mysqlnd_ms_query_is_select`. The function will no longer propose query execution on slave servers.



Multiple master servers

Setting `mysqlnd_ms.multi_master=1` allows the plugin to use multiple master servers, instead of only the first master server of the master list.

Please, see also [supported clusters](#).

22.9.6.6.4. Plugin configuration file (>=1.1.x)

Copyright 1997-2012 the PHP Documentation Group. [3186]



Changelog: Feature was added in PECL/mysqlnd_ms 1.1.0-beta

The below description applies to PECL/mysqlnd_ms >= 1.1.0-beta. It is not valid for prior versions.

The plugin uses its own configuration file. The configuration file holds information about the MySQL replication master server, the MySQL replication slave servers, the server pick (load balancing) policy, the failover strategy, and the use of lazy connections.

The plugin loads its configuration file at the beginning of a web request. It is then cached in memory and used for the duration of the web request. This way, there is no need to restart PHP after deploying the configuration file. Configuration file changes will become active almost instantly.

The PHP configuration directive `mysqlnd_ms.ini_file` is used to set the plugins configuration file. Please note, that the PHP configuration directive may not be evaluated for every web request. Therefore, changing the plugins configuration file name or location may require a PHP restart. However, no restart is required to read changes if an already existing plugin configuration file is updated.

Using and parsing JSON is efficient, and using JSON makes it easier to express hierarchical data structures than the standard `php.ini` format.

Example 22.256. Converting a PHP array (hash) into JSON format

Or alternatively, a developer may be more familiar with the PHP array syntax, and prefer it. This example demonstrates how a developer might convert a PHP array to JSON.

```
<?php
$config = array(
    "myapp" => array(
        "master" => array(
            "master_0" => array(
                "host" => "localhost",
                "socket" => "/tmp/mysql.sock",
            ),
        ),
        "slave" => array(),
    ),
);

file_put_contents("mysqlnd_ms.ini", json_encode($config, JSON_PRETTY_PRINT));
printf("mysqlnd_ms.ini file created...\n");
printf("Dumping file contents...\n");
printf("%s\n", str_repeat("-", 80));
echo file_get_contents("mysqlnd_ms.ini");
printf("\n%s\n", str_repeat("-", 80));
?>
```

The above example will output:

```
mysqlnd_ms.ini file created...
Dumping file contents...
-----
{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost",
                "socket": "\tmp\mysql.sock"
            }
        }
    }
}
```

```

    },
    "slave": [
    ]
  }
}
-----

```

A plugin configuration file consists of one or more sections. Sections are represented by the top-level object properties of the object encoded in the JSON file. Sections could also be called *configuration names*.

Applications reference sections by their name. Applications use section names as the host (server) parameter to the various connect methods of the [mysqli](#), [mysql](#) and [PDO_MYSQL](#) extensions. Upon connect, the `mysqlnd` plugin compares the hostname with all of the section names from the plugin configuration file. If the hostname and section name match, then the plugin will load the settings for that section.

Example 22.257. Using section names example

```

{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27"
      },
      "slave_1": {
        "host": "192.168.2.27",
        "port": 3306
      }
    }
  },
  "localhost": {
    "master": [
      {
        "host": "localhost",
        "socket": "\\path\\to\\mysql.sock"
      }
    ],
    "slave": [
      {
        "host": "192.168.3.24",
        "port": "3305"
      },
      {
        "host": "192.168.3.65",
        "port": "3309"
      }
    ]
  }
}

```

<?php

```
/* All of the following connections will be load balanced */
$mysqli = new mysqli("myapp", "username", "password", "database");
$pdo = new PDO('mysql:host=myapp;dbname=database', 'username', 'password');
$mysql = mysql_connect("myapp", "username", "password");

$mysqli = new mysqli("localhost", "username", "password", "database");
?>
```

Section names are strings. It is valid to use a section name such as `192.168.2.1`, `127.0.0.1` or `localhost`. If, for example, an application connects to `localhost` and a plugin configuration section `localhost` exists, the semantics of the connect operation are changed. The application will no longer only use the MySQL server running on the host `localhost`, but the plugin will start to load balance MySQL queries following the rules from the `localhost` configuration section. This way you can load balance queries from an application without changing the applications source code. Please keep in mind, that such a configuration may not contribute to overall readability of your applications source code. Using section names that can be mixed up with host names should be seen as a last resort.

Each configuration section contains, at a minimum, a list of master servers and a list of slave servers. The master list is configured with the keyword `master`, while the slave list is configured with the `slave` keyword. Failing to provide a slave list will result in a fatal `E_ERROR` level error, although a slave list may be empty. It is possible to allow no slaves. However, this is only recommended with synchronous clusters, please see also [supported clusters](#). The main part of the documentation focusses on the use of asynchronous MySQL replication clusters.

The master and slave server lists can be optionally indexed by symbolic names for the servers they describe. Alternatively, an array of descriptions for slave and master servers may be used.

Example 22.258. List of anonymous slaves

```
"slave": [
  {
    "host": "192.168.3.24",
    "port": "3305"
  },
  {
    "host": "192.168.3.65",
    "port": "3309"
  }
]
```

An anonymous server list is encoded by the `JSON array` type. Optionally, symbolic names may be used for indexing the slave or master servers of a server list, and done so using the `JSON object` type.

Example 22.259. Master list using symbolic names

```
"master": {
  "master_0": {
    "host": "localhost"
  }
}
```

It is recommended to index the server lists with symbolic server names. The alias names will be shown in error messages.

The order of servers is preserved and taken into account by `mysqlnd_ms`. If, for example, you configure round robin load balancing strategy, the first `SELECT` statement will be executed on the slave that appears first in the slave server list.

A configured server can be described with the `host`, `port`, `socket`, `db`, `user`, `password` and `connect_flags`. It is mandatory to set the database server host using the `host` keyword. All other settings are optional.

Example 22.260. Keywords to configure a server

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "db_server_host",
        "port": "db_server_port",
        "socket": "db_server_socket",
        "db": "database_resp_schema",
        "user": "user",
        "password": "password",
        "connect_flags": 0
      }
    },
    "slave": {
      "slave_0": {
        "host": "db_server_host",
        "port": "db_server_port",
        "socket": "db_server_socket"
      }
    }
  }
}
```

If a setting is omitted, the plugin will use the value provided by the user API call used to open a connection. Please, see the [using section names example \[3581\]](#) above.

The configuration file format has been changed in version 1.1.0-beta to allow for chained filters. Filters are responsible for filtering the configured list of servers to identify a server for execution of a given statement. Filters are configured with the `filter` keyword. Filters are executed by `mysqlnd_ms` in the order of their appearance. Defining filters is optional. A configuration section in the plugins configuration file does not need to have a `filters` entry.

Filters replace the `pick[]` setting from prior versions. The new `random` and `roundrobin` provide the same functionality.

Example 22.261. New `roundrobin` filter, old functionality

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      },
      "slave_1": {
```

```

        "host": "192.168.78.137",
        "port": "3306"
    },
    "filters": {
        "roundrobin": [
            ]
        }
    }
}

```

The function `mysqlnd_ms_set_user_pick_server` has been removed. Setting a callback is now done with the `user` filter. Some filters accept parameters. The `user` filter requires and accepts a mandatory `callback` parameter to set the callback previously set through the function `mysqlnd_ms_set_user_pick_server`.

Example 22.262. The `user` filter replaces `mysqlnd_ms_set_user_pick_server`

```

"filters": {
    "user": {
        "callback": "pick_server"
    }
}

```

Here is a short explanation of the configuration directives that can be used.

master array or object

List of MySQL replication master servers. The list of either of the `JSON type array` to declare an anonymous list of servers or of the `JSON type object`. Please, see [above \[3582\]](#) for examples.

Setting at least one master server is mandatory. The plugin will issue an error of type `E_ERROR` if the user has failed to provide a master server list for a configuration section. The fatal error may read `(mysqlnd_ms) Section [master] doesn't exist for host [name_of_a_config_section] in %s on line %d`.

A server is described with the `host`, `port`, `socket`, `db`, `user`, `password` and `connect_flags`. It is mandatory to provide at a value for `host`. If any of the other values is not given, it will be taken from the user API connect call, please, see also: [using section names example \[3581\]](#).

Table of server configuration keywords.

| Keyword | Description | Version |
|-------------------|--|--------------|
| <code>host</code> | Database server host. This is a mandatory setting. Failing to provide, will cause an error of type <code>E_RECOVERABLE_ERROR</code> when the plugin tries to connect to the server. The error message may read <code>(mysqlnd_ms) Cannot find [host] in [%s] section in config in %s on line %d</code> . | Since 1.1.0. |
| <code>port</code> | Database server TCP/IP port. | Since 1.1.0. |

| Keyword | Description | Version |
|-------------------------|-------------------------------------|--------------|
| <code>socket</code> | Database server Unix domain socket. | Since 1.1.0. |
| <code>db</code> | Database (schemata). | Since 1.1.0. |
| <code>user</code> | MySQL database user. | Since 1.1.0. |
| <code>password</code> | MySQL database user password. | Since 1.1.0. |
| <code>connection</code> | Connection flags. | Since 1.1.0. |

The plugin supports using only one master server. An experimental setting exists to enable multi-master support. The details are not documented. The setting is meant for development only.

`slave` array or object

List of one or more MySQL replication slave servers. The syntax is identical to setting master servers, please, see `master` above for details.

The plugin supports using one or more slave servers.

Setting a list of slave servers is mandatory. The plugin will report an error of the type `E_ERROR` if `slave` is not given for a configuration section. The fatal error message may read `(mysqlnd_ms) Section [slave] doesn't exist for host [%s] in %s on line %d`. Note, that it is valid to use an empty slave server list. The error has been introduced to prevent accidentally setting no slaves by forgetting about the `slave` setting. A master-only setup is still possible using an empty slave server list.

If an empty slave list is configured and an attempt is made to execute a statement on a slave the plugin may emit a warning like `mysqlnd_ms) Couldn't find the appropriate slave connection. 0 slaves to choose from.` upon statement execution. It is possible that another warning follows such as `(mysqlnd_ms) No connection selected by the last filter.`

`global_transaction_id_injection` array or object

Global transaction identifier configuration related to both the use of the server built-in global transaction ID feature and the client-side emulation.

| Keyword | Description | Version |
|--------------------|---|--------------|
| <code>fetch</code> | SQL statement for accessing the latest global transaction identifier. The SQL statement is run if the plugin needs to know the most recent global transaction identifier. This can be the case, for example, when checking MySQL Replication slave status. Also used with <code>mysqlnd_ms_get_last_gtid</code> . | Since 1.2.0. |
| <code>check</code> | SQL statement for checking if a replica has replicated all transactions up to and including ones searched for. The SQL statement is run when searching for replicas which can offer a higher level of consistency than eventual consistency. The statement must contain a placeholder <code>#GTID</code> which is to be replaced with | Since 1.2.0. |

| Keyword | Description | Version |
|-----------------------------|---|--------------|
| | the global transaction identifier searched for by the plugin. Please, check the quickstart for examples. | |
| <code>report</code> | Whether to emit an error of type warning if an issue occurs while executing any of the configured SQL statements. | Since 1.2.0. |
| <code>on_completion</code> | Client-side global transaction ID emulation only. SQL statement to run when a transaction finished to update the global transaction identifier sequence number on the master. Please, see the quickstart for examples. | Since 1.2.0. |
| <code>wait_for_slave</code> | <p>Instructs the plugin to wait up to <code>wait_for_gtid_timeout</code> seconds for a slave to catch up when searching for slaves that can deliver session consistency. The setting limits the time spend for polling the slave status. If polling the status takes very long, the total clock time spend waiting may exceed <code>wait_for_gtid_timeout</code>. The plugin calls <code>sleep(1)</code> to sleep one second between each two polls.</p> <p>The setting can be used both with the plugins client-side emulation and the server-side global transaction identifier feature of MySQL 5.6.</p> <p>Waiting for a slave to replicate a certain GTID needed for session consistency also means throttling the client. By throttling the client the write load on the master is reduced indirectly. A primary copy based replication system, such as MySQL Replication, is given more time to reach a consistent state. This can be desired, for example, to increase the number of data copies for high availability considerations or to prevent the master from being overloaded.</p> | Since 1.4.0. |

`filters` object

List of filters. A filter is responsible to filter the list of available servers for executing a given statement. Filters can be chained. The `random` and `roundrobin` filter replace the `pick[]` directive used in prior version to select a load balancing policy. The `user` filter replaces the `mysqLnd_ms_set_user_pick_server` function.

Filters may accept parameters to refine their actions.

If no load balancing policy is set, the plugin will default to `random_once`. The `random_once` policy picks a random slave server when running the first read-only statement. The slave server will be used for all read-only statements until the PHP script execution ends. No load balancing policy is set and thus, defaulting takes place, if neither the `random` nor the `roundrobin` are part of a configuration section.

If a filter chain is configured so that a filter which output no more than once server is used as input for a filter which should be given more than one server as input, the plugin may emit a warning upon opening a connection. The warning may read: `(mysqlnd_ms) Error while creating filter '%s' . Non-multi filter '%s' already created. Stopping in %s on line %d`. Furthermore an error of the error code `2000`, the sql state `HY000` and an error message similar to the warning may be set on the connection handle.

Example 22.263. Invalid filter sequence

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "filters": [
      "roundrobin",
      "random"
    ]
  }
}
```

```
<?php
$link = new mysqli("myapp", "root", "", "test");
printf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error());
$link->query("SELECT 1 FROM DUAL");
?>
```

The above example will output:

```
PHP Warning: mysqli::mysqli(): (HY000/2000): (mysqlnd_ms) Error while cre
[2000] (mysqlnd_ms) Error while creating filter 'random' . Non-multi filter
PHP Warning: mysqli::query(): Couldn't fetch mysqli in filter_warning.php
```

Filter: `random` object

The `random` filter features the random and random once load balancing policies, set through the `pick[]` directive in older versions.

The random policy will pick a random server whenever a read-only statement is to be executed. The random once strategy picks a random slave server once and continues using the slave for the rest of the PHP web request. Random once is a default, if load balancing is not configured through a filter.

If the `random` filter is not given any arguments, it stands for random load balancing policy.

Example 22.264. Random load balancing with `random` filter

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      },
      "slave_1": {
        "host": "192.168.78.137",
        "port": "3306"
      }
    },
    "filters": [
      "random"
    ]
  }
}
```

Optionally, the `sticky` argument can be passed to the filter. If the parameter `sticky` is set to the string `1`, the filter follows the random once load balancing strategy.

Example 22.265. Random once load balancing with `random` filter

```
{
  "filters": {
    "random": {
      "sticky": "1"
    }
  }
}
```

Both the `random` and `roundrobin` filters support setting a priority, a weight for a server, since PECL/mysqld_ms 1.4.0. If the `weight` argument is passed to the filter, it must assign a weight for all servers. Servers must be given an alias name in the `slave` respectively `master` server lists. The alias must be used to reference servers for assigning a priority with `weight`.

Example 22.266. Referencing error

```
[E_RECOVERABLE_ERROR] mysqli_real_connect(): (mysqld_ms) Unknown server 'slav
```

Using a wrong alias name with `weight` may result in an error similar to the shown above.

If `weight` is omitted, the default weight of all servers is one.

Example 22.267. Assigning a `weight` for load balancing

```
{
  "myapp": {
    "master": {
      "master1": {
        "host": "localhost",
        "socket": "\/var\/run\/mysql\/mysql.sock"
      }
    },
    "slave": {
      "slave1": {
        "host": "192.168.2.28",
        "port": 3306
      },
      "slave2": {
        "host": "192.168.2.29",
        "port": 3306
      },
      "slave3": {
        "host": "192.0.43.10",
        "port": 3306
      }
    },
    "filters": {
      "random": {
        "weights": {
          "slave1": 8,
          "slave2": 4,
          "slave3": 1,
          "master1": 1
        }
      }
    }
  }
}
```

At the average a server assigned a weight of two will be selected twice as often as a server assigned a weight of one. Different weights can be assigned to reflect differently sized machines, to prefer co-located slaves which have a low network latency or, to configure a standby failover server. In the latter case, you may want to assign the standby server a very low weight in relation to the other servers. For example, given the configuration above `slave3` will get only some eight percent of the requests in the average. As long as `slave1` and `slave2` are running, it will be used sparsely, similar to a standby failover server. Upon failure of `slave1` and `slave2`, the usage of `slave3` increases. Please, check the notes on failover before using `weight` this way.

Valid weight values range from 1 to 65535.

Unknown arguments are ignored by the filter. No warning or error is given.

The filter expects one or more servers as input. Outputs one server. A filter sequence such as `random`, `roundrobin` may cause a warning and an error message to be set on the connection handle when executing a statement.

List of filter arguments.

| Keyword | Description | Version |
|---------------------|--|--------------|
| <code>sticky</code> | Enables or disabled random once load balancing policy. See above. | Since 1.2.0. |
| <code>weight</code> | Assigns a load balancing weight/priority to a server. Please, see above for a description. | Since 1.4.0. |

Filter: `roundrobin` object

If using the `roundrobin` filter, the plugin iterates over the list of configured slave servers to pick a server for statement execution. If the plugin reaches the end of the list, it wraps around to the beginning of the list and picks the first configured slave server.

Example 22.268. `roundrobin` filter

```

{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "filters": [
      "roundrobin"
    ]
  }
}

```

Expects one or more servers as input. Outputs one server. A filter sequence such as `roundrobin`, `random` may cause a warning and an error message to be set on the connection handle when executing a statement.

List of filter arguments.

| Keyword | Description | Version |
|---------------------|--|--------------|
| <code>weight</code> | Assigns a load balancing weight/priority to a server. Please, find a description above . | Since 1.4.0. |

Filter: `user` object

The `user` replaces `mysqlnd_ms_set_user_pick_server` function, which was removed in 1.1.0-beta. The filter sets a callback for user-defined read/write splitting and server selection.

The plugins built-in read/write query split mechanism decisions can be overwritten in two ways. The easiest way is to prepend a query string with the SQL hints `MYSQLND_MS_MASTER_SWITCH`, `MYSQLND_MS_SLAVE_SWITCH` or `MYSQLND_MS_LAST_USED_SWITCH`. Using SQL hints one can control, for example, whether a query shall be send to the MySQL replication master server or one of the slave servers. By help of SQL hints it is not possible to pick a certain slave server for query execution.

Full control on server selection can be gained using a callback function. Use of a callback is recommended to expert users only because the callback has to cover all cases otherwise handled by the plugin.

The plugin will invoke the callback function for selecting a server from the lists of configured master and slave servers. The callback function inspects the query to run and picks a server for query execution by returning the hosts URI, as found in the master and slave list.

If the lazy connections are enabled and the callback chooses a slave server for which no connection has been established so far and establishing the connection to the slave fails, the plugin will return an error upon the next action on the failed connection, for example, when running a query. It is the responsibility of the application developer to handle the error. For example, the application can re-run the query to trigger a new server selection and callback invocation. If so, the callback must make sure to select a different slave, or check slave availability, before returning to the plugin to prevent an endless loop.

Example 22.269. Setting a callback

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "filters": {
      "user": {
        "callback": "pick_server"
      }
    }
  }
}
```

The callback is supposed to return a host to run the query on. The host URI is to be taken from the master and slave connection lists passed to the callback function. If callback returns a value neither found in the master nor in the slave connection lists the plugin will emit an

error of the type `E_RECOVERABLE_ERROR`. The error may read like `(mysqlnd_ms) User filter callback has returned an unknown server. The server 'server that is not in master or slave list' can neither be found in the master list nor in the slave list.` If the application catches the error to ignore it, follow up errors may be set on the connection handle, for example, `(mysqlnd_ms) No connection selected by the last filter with the error code 2000 and the sqlstate HY000.` Furthermore a warning may be emitted.

Referencing a non-existing function as a callback will result in any error of the type `E_RECOVERABLE_ERROR` whenever the plugin tries to callback function. The error message may reads like: `(mysqlnd_ms) Specified callback (pick_server) is not a valid callback.` If the application catches the error to ignore it, follow up errors may be set on the connection handle, for example, `(mysqlnd_ms) Specified callback (pick_server) is not a valid callback with the error code 2000 and the sqlstate HY000.` Furthermore a warning may be emitted.

The following parameters are passed from the plugin to the callback.

| Parameter | Description | Version |
|-------------------------------|---|--------------|
| <code>connected_server</code> | URI of the currently connected database server. | Since 1.1.0. |
| <code>query</code> | Query string of the statement for which a server needs to be picked. | Since 1.1.0. |
| <code>master_list</code> | List of master servers to choose from. Note, that the list of master servers may not be identical to the list of configured master servers if the filter is not the first in the filter chain. Previously run filters may have reduced the master list already. | Since 1.1.0. |
| <code>slave_list</code> | List of slave servers to choose from. Note, that the list of master servers may not be identical to the list of configured master servers if the filter is not the first in the filter chain. Previously run filters may have reduced the master list already. | Since 1.1.0. |
| <code>last_server</code> | URI of the server of the connection used to execute the previous statement on. | Since 1.1.0. |
| <code>in_transaction</code> | Boolean flag indicating whether the statement is part of an open transaction. If autocommit mode is turned off, this will be set to <code>TRUE</code> . Otherwise it is set to <code>FALSE</code> .

Transaction detection is based on monitoring the mysqlnd library call <code>set_autocommit</code> . Monitoring is not possible before PHP 5.4.0. Please, see connection pooling and switching concepts discussion for further details. | Since 1.1.0. |

Example 22.270. Using a callback

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      },
      "slave_1": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "filters": {
      "user": {
        "callback": "pick_server"
      }
    }
  }
}
```

```
<?php
function pick_server($connected, $query, $masters, $slaves, $last_used_conn)
{
  static $slave_idx = 0;
  static $num_slaves = NULL;
  if (is_null($num_slaves))
    $num_slaves = count($slaves);

  /* default: fallback to the plugins build-in logic */
  $ret = NULL;

  printf("User has connected to '%s'...\n", $connected);
  printf("... deciding where to run '%s'\n", $query);

  $where = mysqlnd_ms_query_is_select($query);
  switch ($where)
  {
    case MYSQLND_MS_QUERY_USE_MASTER:
      printf("... using master\n");
      $ret = $masters[0];
      break;
    case MYSQLND_MS_QUERY_USE_SLAVE:
      /* SELECT or SQL hint for using slave */
      if (strstr($query, "FROM table_on_slave_a_only"))
      {
        /* a table which is only on the first configured slave */
        printf("... access to table available only on slave A detected\n");
        $ret = $slaves[0];
      }
      else
      {
        /* round robin */
        printf("... some read-only query for a slave\n");
        $ret = $slaves[$slave_idx++ % $num_slaves];
      }
      break;
    case MYSQLND_MS_QUERY_LAST_USED:
      printf("... using last used server\n");
      $ret = $last_used_connection;
  }
}
```

```

        break;
    }

    printf("... ret = '%s'\n", $ret);
    return $ret;
}

$mysqli = new mysqli("myapp", "root", "", "test");

if (!$res = $mysqli->query("SELECT 1 FROM DUAL"))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();

if (!$res = $mysqli->query("SELECT 2 FROM DUAL"))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();

if (!$res = $mysqli->query("SELECT * FROM table_on_slave_a_only"))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();

$mysqli->close();
?>

```

The above example will output:

```

User has connected to 'myapp'...
... deciding where to run 'SELECT 1 FROM DUAL'
... some read-only query for a slave
... ret = 'tcp://192.168.2.27:3306'
User has connected to 'myapp'...
... deciding where to run 'SELECT 2 FROM DUAL'
... some read-only query for a slave
... ret = 'tcp://192.168.78.136:3306'
User has connected to 'myapp'...
... deciding where to run 'SELECT * FROM table_on_slave_a_only'
... access to table available only on slave A detected
... ret = 'tcp://192.168.2.27:3306'

```

Filter: `user_multi` object

The `user_multi` differs from the `user` only in one aspect. Otherwise, their syntax is identical. The `user` filter must pick and return exactly one node for statement execution. A filter chain usually ends with a filter that emits only one node. The filter chain shall reduce the list of candidates for statement execution down to one. This, only one node left, is the case after the `user` filter has been run.

The `user_multi` filter is a multi filter. It returns a list of slave and a list of master servers. This list needs further filtering to identify exactly one node for statement execution. A multi filter is typically placed at the top of the filter chain. The `quality_of_service` filter is another example of a multi filter.

The return value of the callback set for `user_multi` must be an array with two elements. The first element holds a list of selected master

servers. The second element contains a list of selected slave servers. The lists shall contain the keys of the slave and master servers as found in the slave and master lists passed to the callback. The below example returns random master and slave lists extracted from the functions input.

Example 22.271. Returning random masters and slaves

```
<?php
function pick_server($connected, $query, $masters, $slaves, $last_used_conn)
{
    $picked_masters = array()
    foreach ($masters as $key => $value) {
        if (mt_rand(0, 2) > 1)
            $picked_masters[] = $key;
    }
    $picked_slaves = array()
    foreach ($slaves as $key => $value) {
        if (mt_rand(0, 2) > 1)
            $picked_slaves[] = $key;
    }
    return array($picked_masters, $picked_slaves);
}
?>
```

The plugin will issue an error of type `E_RECOVERABLE` if the callback fails to return a server list. The error may read `(mysqlnd_ms) User multi filter callback has not returned a list of servers to use. The callback must return an array in %s on line %d`. In case the server list is not empty but has invalid servers key/ids in it, an error of type `E_RECOVERABLE` will be thrown with an error message like `(mysqlnd_ms) User multi filter callback has returned an invalid list of servers to use. Server id is negative in %s on line %d, or similar`.

Whether an error is emitted in case of an empty slave or master list depends on the configuration. If an empty master list is returned for a write operation, it is likely that the plugin will emit a warning that may read `(mysqlnd_ms) Couldn't find the appropriate master connection. 0 masters to choose from. Something is wrong in %s on line %d`. Typically a follow up error of type `E_ERROR` will happen. In case of a read operation and an empty slave list the behavior depends on the fail over configuration. If fail over to master is enabled, no error should appear. If fail over to master is deactivated the plugin will emit a warning that may read `(mysqlnd_ms) Couldn't find the appropriate slave connection. 0 slaves to choose from. Something is wrong in %s on line %d`.

Filter: [quality_of_service](#) object

The [quality_of_service](#) identifies cluster nodes capable of delivering a certain quality of service. It is a multi filter which returns zero, one or multiple of its input servers. Thus, it must be followed by other filters to reduce the number of candidates down to one for statement execution.

The `quality_of_service` filter has been introduced in 1.2.0-alpha. In the 1.2 series the filters focus is on the consistency aspect of service quality. Different types of clusters offer different default data consistencies. For example, an asynchronous MySQL replication slave offers eventual consistency. The slave may not be able to deliver requested data because it has not replicated the write, it may serve stale database because its lagging behind or it may serve current information. Often, this is acceptable. In some cases higher consistency levels are needed for the application to work correct. In those cases, the `quality_of_service` can filter out cluster nodes which cannot deliver the necessary quality of service.

The `quality_of_service` filter can be replaced or created at runtime. A successful call to `mysqLnd_ms_set_qos` removes all existing `qos` filter entries from the filter list and installs a new one at the very beginning. All settings that can be made through `mysqLnd_ms_set_qos` can also be in the plugins configuration file. However, use of the function is by far the most common use case. Instead of setting session consistency and strong consistency service levels in the plugins configuration file it is recommended to define only masters and no slaves. Both service levels will force the use of masters only. Using an empty slave list shortens the configuration file, thus improving readability. The only service level for which there is a case of defining in the plugins configuration file is the combination of eventual consistency and maximum slave lag.

| Keyword | Description | Version |
|-----------------------|---|--------------|
| <code>eventual</code> | <p>Request eventual consistency. Allows the use of all master and slave servers. Data returned may or may not be current.</p> <p>Eventual consistency accepts an optional <code>age</code> parameter. If <code>age</code> is given the plugin considers only slaves for reading for which MySQL replication reports a slave lag less or equal to <code>age</code>. The replication lag is measure using <code>SHOW SLAVE STATUS</code>. If the plugin fails to fetch the replication lag, the slave tested is skipped. Implementation details and tips are given in the quality of service concepts section.</p> <p>Please note, if a filter chain generates an empty slave list and the PHP configuration directive <code>mysqLnd_ms.multi_master=0</code> is used, the plugin may emit a warning.</p> <p>Example 22.272. Global limit on slave lag</p> <pre>{ "myapp": { "master": { "master_0": { "host": "localhost" } } } }</pre> | Since 1.2.0. |

| Keyword | Description | Version |
|----------------------|--|--------------|
| | <pre> }, "slave": { "slave_0": { "host": "192.168.2.27", "port": "3306" }, "slave_1": { "host": "192.168.78.136", "port": "3306" } }, "filters": { "quality_of_service": { "eventual_consistency": { "age": 123 } } } } </pre> | |
| <code>session</code> | <p>Request session consistency (read your writes). Allows use of all masters and all slaves which are in sync with the master. If no further parameters are given slaves are filtered out as there is no reliable way to test if a slave has caught up to the master or is lagging behind. Please note, if a filter chain generates an empty slave list and the PHP configuration directive <code>mysqlnd_ms.multi_master=0</code> is used, the plugin may emit a warning.</p> <p>Session consistency temporarily requested using <code>mysqlnd_ms_set_qos</code> is a valuable alternative to using <code>master_on_write</code>. <code>master_on_write</code> is likely to send more statements to the master than needed. The application may be able to continue operation at a lower consistency level after it has done some critical reads.</p> | Since 1.1.0. |
| <code>strong</code> | <p>Request strong consistency. Only masters will be used.</p> | Since 1.2.0. |

`failover` Up to and including 1.3.x: string. Since 1.4.0: object.

Failover policy. Supported policies: `disabled` (default), `master`, `loop_before_master` (Since 1.4.0).

If no failover policy is set, the plugin will not do any automatic failover (`failover=disabled`). Whenever the plugin fails to connect a server it will emit a warning and set the connections error code and message. Thereafter it is up to the application to handle the error and, for example, resend the last statement to trigger the selection of another server.

Please note, the automatic failover logic is applied when opening connections only. Once a connection has been opened no automatic

attempts are made to reopen it in case of an error. If, for example, the server a connection is connected to is shut down and the user attempts to run a statement on the connection, no automatic failover will be tried. Instead, an error will be reported.

If using `failover=master` the plugin will implicitly failover to a master, if available. Please check the concepts documentation to learn about potential pitfalls and risks of using `failover=master`.

Example 22.273. Optional master failover when failing to connect to slave (PECL/mysqlnd_ms < 1.4.0)

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "failover": "master"
  }
}
```

Since PECL/mysqlnd_ms 1.4.0 the failover configuration keyword refers to an object.

Example 22.274. New syntax since 1.4.0

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "failover": { "strategy": "master" }
  }
}
```

| Keyword | Description | Version |
|-----------------------|---|--------------|
| <code>strategy</code> | Failover policy. Possible values: <code>disabled</code> (default), <code>master</code> , <code>loop_before_master</code>

A value of <code>disabled</code> disables automatic failover. | Since 1.4.0. |

| Keyword | Description | Version |
|--------------------------------|---|---|
| | <p>Setting <code>master</code> instructs the plugin to try to connect to a master in case of a slave connection error. If the master connection attempt fails, the plugin exists the failover loop and returns an error to the user.</p> <p>If using <code>loop_before_master</code> and a slave request is made, the plugin tries to connect to other slaves before failing over to a master. If multiple master are given and multi master is enabled, the plugin also loops over the list of masters and attempts to connect before returning an error to the user.</p> | |
| <code>remember_failures</code> | <p>Remember failures for the duration of a web request. Default: <code>false</code>.</p> <p>If set to <code>true</code> the plugin will remember failed hosts and skip the hosts in all future load balancing made for the duration of the current web request.</p> | <p>Since 1.4.0. Experimental feature. The feature is only available together with the <code>random</code> and <code>roundrobin</code> load balancing filter. The behavior and syntax is likely to change in the future.</p> |
| <code>max_retries</code> | <p>Maximum number of connection attempts before skipping host. Default: <code>0</code> (no limit).</p> <p>The setting is used to prevent hosts from being dropped of the host list upon the first failure. If set to <code>n > 0</code>, the plugin will keep the node in the node list even after a failed connection attempt. The node will not be removed immediately from the slave respectively master lists after the first connection failure but instead be tried to connect to up to <code>n</code> times in future load balancing rounds before being removed.</p> | <p>Since 1.4.0. Experimental feature. The feature is only available together with the <code>random</code> and <code>roundrobin</code> load balancing filter. The behavior and syntax is likely to change in the future.</p> |

`lazy_connections` bool

Setting `failover` to any other value but `disabled`, `master` or `loop_before_master` will not emit any warning or error.

Controls the use of lazy connections. Lazy connections are connections which are not opened before the client sends the first connection. Lazy connections are a default.

It is strongly recommended to use lazy connections. Lazy connections help to keep the number of open connections low. If you disable lazy connections and, for example, configure one MySQL replication master server and two MySQL replication slaves, the plugin will open three connections upon the first call to a connect function although the application might use the master connection only.

Lazy connections bare a risk if you make heavy use of actions which change the state of a connection. The plugin does not dispatch all state changing actions to all connections from the connection pool. The few dispatched actions are applied to already opened connections only. Lazy connections opened in the future are not affected. Only some settings are "remembered" and applied when lazy connections are opened.

Example 22.275. Disabling lazy connection

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "lazy_connections": 0
  }
}
```

Please, see also `server_charset` to overcome potential problems with string escaping and servers using different default charsets.

`server_charset` string

The setting has been introduced in 1.4.0. It is recommended to set it if using lazy connections.

The `server_charset` setting serves two purposes. It acts as a fallback charset to be used for string escaping done before a connection has been established and it helps to avoid escaping pitfalls in heterogeneous environments which servers using different default charsets.

String escaping takes a connections charset into account. String escaping is not possible before a connection has been opened and the

connections charset is known. The use of lazy connections delays the actual opening of connections until a statement is send.

An application using lazy connections may attempt to escape a string before sending a statement. In fact, this should be a common case as the statement string may contain the string that is to be escaped. However, due to the lazy connection feature no connection has been opened yet and escaping fails. The plugin may report an error of the type `E_WARNING` and a message like `(mysqld_ms) string escaping doesn't work without established connection. Possible solution is to add server_charset to your configuration` to inform you of the pitfall.

Setting `server_charset` makes the plugin use the given charset for string escaping done on lazy connection handles before establishing a network connection to MySQL. Furthermore, the plugin will enforce the use of the charset when the connection is established.

Enforcing the use of the configured charset used for escaping is done to prevent tapping into the pitfall of using a different charset for escaping than used later for the connection. This has the additional benefit of removing the need to align the charset configuration of all servers used. No matter what the default charset on any of the servers is, the plugin will set the configured one as a default.

The plugin does not stop the user from changing the charset at any time using the `set_charset` call or corresponding SQL statements. Please, note that the use of SQL is not recommended as it cannot be monitored by the plugin. The user can, for example, change the charset on a lazy connection handle after escaping a string and before the actual connection is opened. The charset set by the user will be used for any subsequent escaping before the connection is established. The connection will be established using the configured charset, no matter what the server charset is or what the user has set before. Once a connection has been opened, `set_charset` is of no meaning anymore.

Example 22.276. String escaping on a lazy connection handle

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "lazy_connections": 1,
    "server_charset" : "utf8"
  }
}
```

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
$mysqli->real_escape("this will be escaped using the server_charset setting -
$mysqli->set_charset("latin1");
$mysqli->real_escape("this will be escaped using latin1");
/* server_charset implicitly set - utf8 connection */
$mysqli->query("SELECT 'This connection will be set to server_charset upon est
/* latin1 used from now on */
$mysqli->set_charset("latin1");
?>
```

`master_on_write` bool

If set, the plugin will use the master server only after the first statement has been executed on the master. Applications can still send statements to the slaves using SQL hints to overrule the automatic decision.

The setting may help with replication lag. If an application runs an `INSERT` the plugin will, by default, use the master to execute all following statements, including `SELECT` statements. This helps to avoid problems with reads from slaves which have not replicated the `INSERT` yet.

Example 22.277. Master on write for consistent reads

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "master_on_write": 1
  }
}
```

Please, note the `quality_of_service` filter introduced in version 1.2.0-alpha. It gives finer control, for example, for achieving read-your-writes and, it offers additional functionality introducing `service levels`.

`trx_stickiness` string

Transaction stickiness policy. Supported policies: `disabled` (default), `master`.

The setting requires 5.4.0 or newer. If used with PHP older than 5.4.0, the plugin will emit a warning like `(mysqlnd_ms) trx_stickiness strategy is not supported before PHP 5.3.99`.

If no transaction stickiness policy is set or, if setting `trx_stickiness=disabled`, the plugin is not transaction aware. Thus, the plugin may load balance connections and switch connections

in the middle of a transaction. The plugin is not transaction safe. SQL hints must be used avoid connection switches during a transaction.

As of PHP 5.4.0 the `mysqlnd` library allows the plugin to monitor the `autocommit` mode set by calls to the libraries `set_autocommit()` function. If setting `set_stickiness=master` and `autocommit` gets disabled by a PHP MySQL extension invoking the `mysqlnd` library internal function call `set_autocommit()`, the plugin is made aware of the begin of a transaction. Then, the plugin stops load balancing and directs all statements to the master server until `autocommit` is enabled. Thus, no SQL hints are required.

An example of a PHP MySQL API function calling the `mysqlnd` library internal function call `set_autocommit()` is `mysqli_autocommit`.

Although setting `ser_stickiness=master`, the plugin cannot be made aware of `autocommit` mode changes caused by SQL statements such as `SET AUTOCOMMIT=0`.

Example 22.278. Using master to execute transactions

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "trx_stickiness": "master"
  }
}
```

22.9.6.6.5. Plugin configuration file (<= 1.0.x)

Copyright 1997-2012 the PHP Documentation Group. [3186]



Note

The below description applies to PECL/mysqlnd_ms < 1.1.0-beta. It is not valid for later versions.

The plugin is using its own configuration file. The configuration file holds information on the MySQL replication master server, the MySQL replication slave servers, the server pick (load balancing) policy, the failover strategy and the use of lazy connections.

The PHP configuration directive `mysqlnd_ms.ini_file` is used to set the plugins configuration file.

The configuration file mimics standard the `php.ini` format. It consists of one or more sections. Every section defines its own unit of settings. There is no global section for setting defaults.

Applications reference sections by their name. Applications use section names as the host (server) parameter to the various connect methods of the `mysqli`, `mysql` and `PDO_MYSQL` extensions. Upon

connect the `mysqLnd` plugin compares the hostname with all section names from the plugin configuration file. If hostname and section name match, the plugin will load the sections settings.

Example 22.279. Using section names example

```
[myapp]
master[] = localhost
slave[] = 192.168.2.27
slave[] = 192.168.2.28:3306
[localhost]
master[] = localhost:/tmp/mysql/mysql.sock
slave[] = 192.168.3.24:3305
slave[] = 192.168.3.65:3309
```

```
<?php
/* All of the following connections will be load balanced */
$mysqli = new mysqli("myapp", "username", "password", "database");
$pdo = new PDO('mysql:host=myapp;dbname=database', 'username', 'password');
$mysql = mysql_connect("myapp", "username", "password");

$mysqli = new mysqli("localhost", "username", "password", "database");
?>
```

Section names are strings. It is valid to use a section name such as `192.168.2.1`, `127.0.0.1` or `localhost`. If, for example, an application connects to `localhost` and a plugin configuration section `[localhost]` exists, the semantics of the connect operation are changed. The application will no longer only use the MySQL server running on the host `localhost` but the plugin will start to load balance MySQL queries following the rules from the `[localhost]` configuration section. This way you can load balance queries from an application without changing the applications source code.

The `master[]`, `slave[]` and `pick[]` configuration directives use a list-like syntax. Configuration directives supporting list-like syntax may appear multiple times in a configuration section. The plugin maintains the order in which entries appear when interpreting them. For example, the below example shows two `slave[]` configuration directives in the configuration section `[myapp]`. If doing round-robin load balancing for read-only queries, the plugin will send the first read-only query to the MySQL server `mysql_slave_1` because it is the first in the list. The second read-only query will be send to the MySQL server `mysql_slave_2` because it is the second in the list. Configuration directives supporting list-like syntax result are ordered from top to bottom in accordance to their appearance within a configuration section.

Example 22.280. List-like syntax

```
[myapp]
master[] = mysql_master_server
slave[] = mysql_slave_1
slave[] = mysql_slave_2
```

Here is a short explanation of the configuration directives that can be used.

`master[]` string

URI of a MySQL replication master server. The URI follows the syntax `hostname[:port|unix_domain_socket]`.

The plugin supports using only one master server.

Setting a master server is mandatory. The plugin will report a warning upon connect if the user has failed to provide a master server for a configuration section. The warning may read `(mysqlnd_ms) Cannot find master section in config`. Furthermore the plugin may set an error code for the connection handle such as `HY000/2000 (CR_UNKNOWN_ERROR)`. The corresponding error message depends on your language settings.

`slave[]` string

URI of one or more MySQL replication slave servers. The URI follows the syntax `hostname[:port|unix_domain_socket]`.

The plugin supports using one or more slave servers.

Setting a slave server is mandatory. The plugin will report a warning upon connect if the user has failed to provide at least one slave server for a configuration section. The warning may read `(mysqlnd_ms) Cannot find slaves section in config`. Furthermore the plugin may set an error code for the connection handle such as `HY000/2000 (CR_UNKNOWN_ERROR)`. The corresponding error message depends on your language settings.

`pick[]` string

Load balancing (server picking) policy. Supported policies: `random`, `random_once` (default), `roundrobin`, `user`.

If no load balancing policy is set, the plugin will default to `random_once`. The `random_once` policy picks a random slave server when running the first read-only statement. The slave server will be used for all read-only statements until the PHP script execution ends.

The `random` policy will pick a random server whenever a read-only statement is to be executed.

If using `roundrobin` the plugin iterates over the list of configured slave servers to pick a server for statement execution. If the plugin reaches the end of the list, it wraps around to the beginning of the list and picks the first configured slave server.

Setting more than one load balancing policy for a configuration section makes only sense in conjunction with `user` and `mysqlnd_ms_set_user_pick_server`. If the user defined callback fails to pick a server, the plugin falls back to the second configured load balancing policy.

`failover` string

Failover policy. Supported policies: `disabled` (default), `master`.

If no failover policy is set, the plugin will not do any automatic failover (`failover=disabled`). Whenever the plugin fails to connect a server it will emit a warning and set the connections error code and message. Thereafter it is up to the application to handle the error and, for example, resent the last statement to trigger the selection of another server.

If using `failover=master` the plugin will implicitly failover to a slave, if available. Please check the concepts documentation to learn about potential pitfalls and risks of using `failover=master`.

| | |
|------------------------------------|---|
| <code>lazy_connections</code> bool | <p>Controls the use of lazy connections. Lazy connections are connections which are not opened before the client sends the first connection.</p> <p>It is strongly recommended to use lazy connections. Lazy connections help to keep the number of open connections low. If you disable lazy connections and, for example, configure one MySQL replication master server and two MySQL replication slaves, the plugin will open three connections upon the first call to a connect function although the application might use the master connection only.</p> <p>Lazy connections bare a risk if you make heavy use of actions which change the state of a connection. The plugin does not dispatch all state changing actions to all connections from the connection pool. The few dispatched actions are applied to already opened connections only. Lazy connections opened in the future are not affected. If, for example, the connection character set is changed using a PHP MySQL API call, the plugin will change the character set of all currently opened connection. It will not remember the character set change to apply it on lazy connections opened in the future. As a result the internal connection pool would hold connections using different character sets. This is not desired. Remember that character sets are taken into account for escaping.</p> |
| <code>master_on_write</code> bool | <p>If set, the plugin will use the master server only after the first statement has been executed on the master. Applications can still send statements to the slaves using SQL hints to overrule the automatic decision.</p> <p>The setting may help with replication lag. If an application runs an <code>INSERT</code> the plugin will, by default, use the master to execute all following statements, including <code>SELECT</code> statements. This helps to avoid problems with reads from slaves which have not replicated the <code>INSERT</code> yet.</p> |
| <code>trx_stickiness</code> string | <p>Transaction stickiness policy. Supported policies: <code>disabled</code> (default), <code>master</code>.</p> <p>Experimental feature.</p> <p>The setting requires 5.4.0 or newer. If used with PHP older than 5.4.0, the plugin will emit a warning like <code>(mysqlnd_ms) trx_stickiness strategy is not supported before PHP 5.3.99</code>.</p> <p>If no transaction stickiness policy is set or, if setting <code>trx_stickiness=disabled</code>, the plugin is not transaction aware. Thus, the plugin may load balance connections and switch connections in the middle of a transaction. The plugin is not transaction safe. SQL hints must be used avoid connection switches during a transaction.</p> <p>As of PHP 5.4.0 the mysqlnd library allows the plugin to monitor the <code>autocommit</code> mode set by calls to the libraries <code>trx_autocommit()</code> function. If setting <code>trx_stickiness=master</code> and <code>autocommit</code> gets disabled by a PHP MySQL extension invoking the <code>mysqlnd</code> library internal function call <code>trx_autocommit()</code>, the plugin is made aware of the begin of a transaction. Then, the plugin stops load balancing</p> |

and directs all statements to the master server until `autocommit` is enabled. Thus, no SQL hints are required.

An example of a PHP MySQL API function calling the `mysqlnd` library internal function call `trx_autocommit()` is `mysqli_autocommit`.

Although setting `trx_stickiness=master`, the plugin cannot be made aware of `autocommit` mode changes caused by SQL statements such as `SET AUTOCOMMIT=0`.

22.9.6.6.6. Testing

Copyright 1997-2012 the PHP Documentation Group. [3186]



Note

The section applies to `mysqlnd_ms` 1.1.0 or newer, not the 1.0 series.

The PECL/`mysqlnd_ms` test suite is in the `tests/` directory of the source distribution. The test suite consists of standard phpt tests, which are described on the PHP Quality Assurance Teams website.

Running the tests requires setting up one to four MySQL servers. Some tests don't connect to MySQL at all. Others require one server for testing. Some require two distinct servers. In some cases two servers are used to emulate a replication setup. In other cases a master and a slave of an existing MySQL replication setup are required for testing. The tests will try to detect how many servers and what kind of servers are given. If the required servers are not found, the test will be skipped automatically.

Before running the tests, edit `tests/config.inc` to configure the MySQL servers to be used for testing.

The most basic configuration is as follows.

```
putenv("MYSQL_TEST_HOST=localhost");
putenv("MYSQL_TEST_PORT=3306");
putenv("MYSQL_TEST_USER=root");
putenv("MYSQL_TEST_PASSWD=");
putenv("MYSQL_TEST_DB=test");
putenv("MYSQL_TEST_ENGINE=MyISAM");
putenv("MYSQL_TEST_SOCKET=");

putenv("MYSQL_TEST_SKIP_CONNECT_FAILURE=1");
putenv("MYSQL_TEST_CONNECT_FLAGS=0");
putenv("MYSQL_TEST_EXPERIMENTAL=0");

/* replication cluster emulation */
putenv("MYSQL_TEST_EMULATED_MASTER_HOST=". getenv("MYSQL_TEST_HOST"));
putenv("MYSQL_TEST_EMULATED_SLAVE_HOST=". getenv("MYSQL_TEST_HOST"));

/* real replication cluster */
putenv("MYSQL_TEST_MASTER_HOST=". getenv("MYSQL_TEST_EMULATED_MASTER_HOST"));
putenv("MYSQL_TEST_SLAVE_HOST=". getenv("MYSQL_TEST_EMULATED_SLAVE_HOST"));
```

`MYSQL_TEST_HOST`, `MYSQL_TEST_PORT` and `MYSQL_TEST_SOCKET` define the hostname, TCP/IP port and Unix domain socket of the default database server. `MYSQL_TEST_USER` and `MYSQL_TEST_PASSWD` contain the user and password needed to connect to the database/schema configured with `MYSQL_TEST_DB`. All configured servers must have the same database user configured to give access to the test database.

Using `host`, `host:port` or `host:/path/to/socket` syntax one can set an alternate host, host and port or host and socket for any of the servers.

```
putenv("MYSQL_TEST_SLAVE_HOST=192.168.78.136:3307");
putenv("MYSQL_TEST_MASTER_HOST=myserver_hostname:/path/to/socket");
```

22.9.6.6.7. Debugging and Tracing

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqlnd` debug log can be used to debug and trace the activities of `PECL/mysqlnd_ms`. As a `mysqlnd` `PECL/mysqlnd_ms` adds trace information to the `mysqlnd` library debug file. Please, see the [mysqlnd.debug](#) PHP configuration directive documentation for a detailed description on how to configure the debug log.

Configuration setting example to activate the debug log:

```
mysqlnd.debug=d:t:x:0,/tmp/mysqlnd.trace
```



Note

This feature is only available with a debug build of PHP. Works on Microsoft Windows if using a debug build of PHP and PHP was built using Microsoft Visual C version 9 and above.

The debug log shows `mysqlnd` library and `PECL/mysqlnd_ms` plugin function calls, similar to a trace log. `mysqlnd` library calls are usually prefixed with `mysqlnd_`. `PECL/mysqlnd` internal calls begin with `mysqlnd_ms`.

Example excerpt from the debug log (connect):

```
[...]
>mysqlnd_connect
| info : host=myapp user=root db=test port=3306 flags=131072
| >mysqlnd_ms::connect
| | >mysqlnd_ms_config_json_section_exists
| | | info : section=[myapp] len=[5]
| | | >mysqlnd_ms_config_json_sub_section_exists
| | | | info : section=[myapp] len=[5]
| | | | info : ret=1
| | | <mysqlnd_ms_config_json_sub_section_exists
| | | info : ret=1
| | <mysqlnd_ms_config_json_section_exists
[...]
```

The debug log is not only useful for plugin developers but also to find the cause of user errors. For example, if your application does not do proper error handling and fails to record error messages, checking the debug and trace log may help finding the cause. Use of the debug log to debug application issues should be considered only if no other option is available. Writing the debug log to disk is a slow operation and may have negative impact on the application performance.

Example excerpt from the debug log (connection failure):

```
[...]
| | | | | info : adding error [Access denied for user 'root'@'localhost' (using password: YES)] to the
| | | | | info : PACKET_FREE(0)
| | | | | info : PACKET_FREE(0x7f3ef6323f50)
| | | | | info : PACKET_FREE(0x7f3ef6324080)
| | | | | <mysqld_auth_handshake
| | | | | info : switch_to_auth_protocol=n/a
| | | | | info : conn->error_info.error_no = 1045
| | | | | <mysqld_connect_run_authentication
| | | | | info : PACKET_FREE(0x7f3ef63236d8)
| | | | | >mysqld_conn::free_contents
| | | | | | >mysqld_net::free_contents
| | | | | | <mysqld_net::free_contents
| | | | | | info : Freeing memory of members
| | | | | | info : scheme=unix:///tmp/mysql.sock
| | | | | | >mysqld_error_list_pdtor
| | | | | | <mysqld_error_list_pdtor
| | | | | | <mysqld_conn::free_contents
| | | | | <mysqld_conn::connect
[...]
```

The trace log can also be used to verify correct behaviour of PECL/[mysqld_ms](#) itself, for example, to check which server has been selected for query execution and why.

Example excerpt from the debug log (plugin decision):

```
[...]
>mysqld_ms::query
| info : query=DROP TABLE IF EXISTS test
| >_mysqld_plugin_get_plugin_connection_data
| | info : plugin_id=5
| <_mysqld_plugin_get_plugin_connection_data
| >mysqld_ms_pick_server_ex
| | info : conn_data=0x7fb6a7d3e5a0 *conn_data=0x7fb6a7d410d0
| | >mysqld_ms_select_servers_all
| | <mysqld_ms_select_servers_all
| | >mysqld_ms_choose_connection_rr
| | | >mysqld_ms_query_is_select
[...]
```

```
| | | <mysqld_ms_query_is_select
[...]
```

```
| | | info : Init the master context
| | | info : list(0x7fb6a7d3f598) has 1
| | | info : Using master connection
| | | >mysqld_ms_advanced_connect
| | | | >mysqld_conn::connect
| | | | info : host=localhost user=root db=test port=3306 flags=131072 persistent=0 state=0
```

In this case the statement `DROP TABLE IF EXISTS test` has been executed. Note that the statement string is shown in the log file. You may want to take measures to restrict access to the log for security considerations.

The statement has been load balanced using round robin policy, as you can easily guess from the functions name `>mysqld_ms_choose_connection_rr`. It has been sent to a master server running on `host=localhost user=root db=test port=3306 flags=131072 persistent=0 state=0`.

22.9.6.6.8. Monitoring

Copyright 1997-2012 the PHP Documentation Group. [3186]

Plugin activity can be monitored using the mysqlnd trace log, mysqlnd statistics, mysqlnd_ms plugin statistics and external PHP debugging tools. Use of the trace log should be limited to debugging. It is recommended to use the plugins statistics for monitoring.

Writing a trace log is a slow operation. If using an external PHP debugging tool, please refer to the vendors manual about its performance impact and the type of information collected. In many cases, external debugging tools will provide call stacks. Often, a call stack or a trace log is more difficult to interpret than the statistics provided by the plugin.

Plugin statistics tell how often which kind of cluster node has been used (slave or master), why the node was used, if lazy connections have been used and if global transaction ID injection has been performed. The monitoring information provided enables user to verify plugin decisions and to plan their cluster resources based on usage pattern. The function [mysqlnd_ms_get_stats](#) is used to access the statistics. Please, see the functions description for a list of available statistics.

Statistics are collected on a per PHP process basis. Their scope is a PHP process. Depending on the PHP deployment model a process may serve one or multiple web requests. If using CGI model, a PHP process serves one web request. If using FastCGI or pre-fork web server models, a PHP process usually serves multiple web requests. The same is the case with a threaded web server. Please, note that threads running in parallel can update the statistics in parallel. Thus, if using a threaded PHP deployment model, statistics can be changed by more than one script at a time. A script cannot rely on the fact that it sees only its own changes to statistics.

Example 22.281. Verify plugin activity in a non-threaded deployment model

```
mysqlnd_ms.enable=1
mysqlnd_ms.collect_statistics=1
```

```
<?php
/* Load balanced following "myapp" section rules from the plugins config file (not shown) */
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

$stats_before = mysqlnd_ms_get_stats();
if ($res = $mysqli->query("SELECT 'Read request' FROM DUAL")) {
    var_dump($res->fetch_all());
}
$stats_after = mysqlnd_ms_get_stats();
if ($stats_after['use_slave'] <= $stats_before['use_slave']) {
    echo "According to the statistics the read request has not been run on a slave!";
}
?>
```

Statistics are aggregated for all plugin activities and all connections handled by the plugin. It is not possible to tell how much a certain connection handle has contributed to the overall statistics.

Utilizing PHPs [register_shutdown_function](#) function or the [auto_append_file](#) PHP configuration directive it is easily possible to dump statistics into, for example, a log file when a script finishes. Instead of using a log file it is also possible to send the statistics to an external monitoring tool for recording and display.

Example 22.282. Recording statistics during shutdown

```
mysqlnd_ms.enable=1
mysqlnd_ms.collect_statistics=1
error_log=/tmp/php_errors.log
```

```
<?php
function check_stats() {
    $msg = str_repeat("-", 80) . "\n";
    $msg .= var_export(mysqlnd_ms_get_stats(), true) . "\n";
    $msg .= str_repeat("-", 80) . "\n";
    error_log($msg);
}
register_shutdown_function("check_stats");
?>
```

22.9.6.7. Predefined Constants

Copyright 1997-2012 the PHP Documentation Group. [3186]

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

SQL hint related

Example 22.283. Example demonstrating the usage of `mysqlnd_ms` constants

The `mysqlnd` replication and load balancing plugin (`mysqlnd_ms`) performs read/write splitting. This directs write queries to a MySQL master server, and read-only queries to the MySQL slave servers. The plugin has a built-in read/write split logic. All queries which start with `SELECT` are considered read-only queries, which are then sent to a MySQL slave server that is listed in the plugin configuration file. All other queries are directed to the MySQL master server that is also specified in the plugin configuration file.

User supplied SQL hints can be used to overrule automatic read/write splitting, to gain full control on the process. SQL hints are standards compliant SQL comments. The plugin will scan the beginning of a query string for an SQL comment for certain commands, which then control query redirection. Other systems involved in the query processing are unaffected by the SQL hints because other systems will ignore the SQL comments.

The plugin supports three SQL hints to direct queries to either the MySQL slave servers, the MySQL master server, or the last used MySQL server. SQL hints must be placed at the beginning of a query to be recognized by the plugin.

For better portability, it is recommended to use the string constants `MYSQLND_MS_MASTER_SWITCH`, `MYSQLND_MS_SLAVE_SWITCH` and `MYSQLND_MS_LAST_USED_SWITCH` instead of their literal values.

```
<?php
/* Use constants for maximum portability */
$master_query = "/" . MYSQLND_MS_MASTER_SWITCH . "*/SELECT id FROM test";

/* Valid but less portable: using literal instead of constant */
$slave_query = "/*ms=slave*/SHOW TABLES";

printf("master_query = '%s'\n", $master_query);
printf("slave_query = '%s'\n", $slave_query);
?>
```

The above examples will output:

```
master_query = /*ms=master*/SELECT id FROM test
slave_query = /*ms=slave*/SHOW TABLES
```

`MYSQLND_MS_MASTER_SWITCH` (string) SQL hint used to send a query to the MySQL replication master server.

`MYSQLND_MS_SLAVE_SWITCH` (string) SQL hint used to send a query to one of the MySQL replication slave servers.

`MYSQLND_MS_LAST_USED_SWITCH` (string) SQL hint used to send a query to the last used MySQL server. The last used MySQL server can either be a master or a slave server in a MySQL replication setup.

`mysqlnd_ms_query_is_select` related

`MYSQLND_MS_QUERY_USE_MASTER` (integer) If `mysqlnd_ms_is_select` returns `MYSQLND_MS_QUERY_USE_MASTER` for a given query, the built-in read/write split mechanism recommends sending the query to a MySQL replication master server.

`MYSQLND_MS_QUERY_USE_SLAVE` (integer) If `mysqlnd_ms_is_select` returns `MYSQLND_MS_QUERY_USE_SLAVE` for a given query, the built-in read/write split mechanism recommends sending the query to a MySQL replication slave server.

`MYSQLND_MS_QUERY_USE_LAST_USED` (integer) If `mysqlnd_ms_is_select` returns `MYSQLND_MS_QUERY_USE_LAST_USED` for a given query, the built-in read/write split mechanism recommends sending the query to the last used server.

`mysqlnd_ms_set_qos`, quality of service filter and service level related

`MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL` (integer) Use to request the service level eventual consistency from the `mysqlnd_ms_set_qos`. Eventual consistency is the default quality of service when reading from an asynchronous MySQL replication slave. Data returned in this service level may or may not be stale, depending on whether the selected slaves happens to have replicated the latest changes from the MySQL replication master or not.

`MYSQLND_MS_QOS_CONSISTENCY_SESSION` (integer) Use to request the service level session consistency from the `mysqlnd_ms_set_qos`. Session consistency is defined as read your writes. The client is guaranteed to see his latest changes.

`MYSQLND_MS_QOS_CONSISTENCY_STRONG` (integer) Use to request the service level strong consistency from the `mysqlnd_ms_set_qos`. Strong consistency is used to ensure all clients see each others changes.

`MYSQLND_MS_QOS_OPTION_GTID` (integer) Used as a service level option with `mysqlnd_ms_set_qos` to parameterize session consistency.

`MYSQLND_MS_QOS_OPTION_AGE` (integer) Used as a service level option with `mysqlnd_ms_set_qos` to parameterize eventual consistency.

Other

The plugins version number can be obtained using `MYSQLND_MS_VERSION` or `MYSQLND_MS_VERSION_ID`. `MYSQLND_MS_VERSION` is the string representation of the numerical version number `MYSQLND_MS_VERSION_ID`, which is an integer such as 10000. Developers can calculate the version number as follows.

| Version (part) | Example |
|-----------------------|-----------------|
| Major*10000 | 1*10000 = 10000 |
| Minor*100 | 0*100 = 0 |
| Patch | 0 = 0 |
| MYSQLND_MS_VERSION_ID | 10000 |

`MYSQLND_MS_VERSION` (string) Plugin version string, for example, "1.0.0-prototype".

`MYSQLND_MS_VERSION_ID` (integer) Plugin version number, for example, 10000.

22.9.6.8. MySQLnd_ms Functions

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.6.8.1. `mysqlnd_ms_get_last_gtid`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_ms_get_last_gtid`

Returns the latest global transaction ID

Description

```
string mysqlnd_ms_get_last_gtid(
    mixed connection);
```

Returns a global transaction identifier which belongs to a write operation no older than the last write performed by the client. It is not guaranteed that the global transaction identifier is identical to that one created for the last write transaction performed by the client.

Parameters

connection

A PECL/mysqlnd_ms connection handle to a MySQL server of the type `PDO_MYSQL`, `mysqli` or `ext/mysql`. The connection handle is obtained when opening a connection with a host name that matches a `mysqlnd_ms` configuration file entry using any of the above three MySQL driver extensions.

Return Values

Returns a global transaction ID (GTID) on success. Otherwise, returns `FALSE`.

The function `mysqlnd_ms_get_last_gtid` returns the GTID obtained when executing the SQL statement from the `fetch_last_gtid` entry of the `global_transaction_id_injection` section from the plugins configuration file.

The function may be called after the GTID has been incremented.

Notes



Note

`mysqlnd_ms_get_last_gtid` requires PHP >= 5.4.0 and PECL `mysqlnd_ms` >= 1.2.0. Internally, it is using a `mysqlnd` library C functionality not available with PHP 5.3.

Examples

Example 22.284. `mysqlnd_ms_get_last_gtid` example

```
<?php
/* Open mysqlnd_ms connection using mysqli, PDO_MySQL or mysql extension */
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("DROP TABLE IF EXISTS test"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

printf("GTID after transaction %s\n", mysqlnd_ms_get_last_gtid($mysqli));

/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("CREATE TABLE test(id INT)"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

printf("GTID after transaction %s\n", mysqlnd_ms_get_last_gtid($mysqli));
?>
```

See Also

[Global Transaction IDs](#)

22.9.6.8.2. `mysqlnd_ms_get_last_used_connection`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_ms_get_last_used_connection](#)

Returns an array which describes the last used connection

Description

```
array mysqlnd_ms_get_last_used_connection(
    mixed connection);
```

Returns an array which describes the last used connection from the plugins connection pool currently pointed to by the user connection handle. If using the plugin, a user connection handle represents a pool of database connections. It is not possible to tell from the user connection handles properties to which database server from the pool the user connection handle points.

The function can be used to debug or monitor PECL `mysqlnd_ms`.

Parameters

connection A MySQL connection handle obtained from any of the connect functions of the [mysqli](#), [mysql](#) or [PDO_MYSQL](#) extensions.

Return Values

`FALSE` on error. Otherwise, an array which describes the connection used to execute the last statement on.

Array which describes the connection.

| Property | Description | Version |
|-----------------------------|---|--------------|
| <code>scheme</code> | Connection scheme. Either <code>tcp://host:port</code> or <code>unix://host:socket</code> . If you want to distinguish connections from each other use a combination of <code>scheme</code> and <code>thread_id</code> as a unique key. Neither <code>scheme</code> nor <code>thread_id</code> alone are sufficient to distinguish two connections from each other. Two servers may assign the same <code>thread_id</code> to two different connections. Thus, connections in the pool may have the same <code>thread_id</code> . Also, do not rely on uniqueness of <code>scheme</code> in a pool. Your QA engineers may use the same MySQL server instance for two distinct logical roles and add it multiple times to the pool. This hack is used, for example, in the test suite. | Since 1.1.0. |
| <code>host</code> | Database server host used with the connection. The host is only set with TCP/IP connections. It is empty with Unix domain or Windows named pipe connections, | Since 1.1.0. |
| <code>host_info</code> | A character string representing the server hostname and the connection type. | Since 1.1.2. |
| <code>port</code> | Database server port used with the connection. | Since 1.1.0. |
| <code>socket_or_pipe</code> | Unix domain socket or Windows named pipe used with the connection. The value is empty for TCP/IP connections. | Since 1.1.2. |
| <code>thread_id</code> | Connection thread id. | Since 1.1.0. |
| <code>last_message</code> | Info message obtained from the MySQL C API function <code>mysql_info()</code> . Please, see <code>mysqli_info</code> for a description. | Since 1.1.0. |
| <code>errno</code> | Error code. | Since 1.1.0. |
| <code>error</code> | Error message. | Since 1.1.0. |
| <code>sqlstate</code> | Error SQLstate code. | Since 1.1.0. |

Notes



Note

`mysqlnd_ms_get_last_used_connection` requires PHP \geq 5.4.0 and PECL `mysqlnd_ms` \gg 1.1.0. Internally, it is using a `mysqlnd` library C call not available with PHP 5.3.

Examples

The example assumes that `myapp` refers to a plugin configuration file section and represents a connection pool.

Example 22.285. `mysqlnd_ms_get_last_used_connection` example

```
<?php
$link = new mysqli("myapp", "user", "password", "database");
$res = $link->query("SELECT 1 FROM DUAL");
var_dump(mysqlnd_ms_get_last_used_connection($link));
?>
```

The above example will output:

```
array(10) {
  ["scheme"]=>
  string(22) "unix:///tmp/mysql.sock"
  ["host_info"]=>
  string(25) "Localhost via UNIX socket"
  ["host"]=>
  string(0) ""
  ["port"]=>
  int(3306)
  ["socket_or_pipe"]=>
  string(15) "/tmp/mysql.sock"
  ["thread_id"]=>
  int(46253)
  ["last_message"]=>
  string(0) ""
  ["errno"]=>
  int(0)
  ["error"]=>
  string(0) ""
  ["sqlstate"]=>
  string(5) "00000"
}
```

22.9.6.8.3. `mysqlnd_ms_get_stats`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_ms_get_stats`

Returns query distribution and connection statistics

Description

```
array mysqlnd_ms_get_stats();
```

Returns an array of statistics collected by the replication and load balancing plugin.

The PHP configuration setting `mysqlnd_ms.collect_statistics` controls the collection of statistics. The collection of statistics is disabled by default for performance reasons.

The scope of the statistics is the PHP process. Depending on your deployment model a PHP process may handle one or multiple requests.

Statistics are aggregated for all connections and all storage handler. It is not possible to tell how much queries originating from `mysqli`, `PDO_MySQL` or `mysql` API calls have contributed to the aggregated data values.

Parameters

This function has no parameters.

Return Values

Returns `NULL` if the PHP configuration directive `mysqlnd_ms.enable` has disabled the plugin. Otherwise, returns array of statistics.

Array of statistics

| Statistic | Description | Version |
|-------------------------------|--|--------------|
| <code>use_slave</code> | <p>The semantics of this statistic has changed between 1.0.1 - 1.1.0.</p> <p>The meaning for version 1.0.1 is as follows. Number of statements considered as read-only by the built-in query analyzer. Neither statements which begin with a SQL hint to force use of slave nor statements directed to a slave by an user-defined callback are included. The total number of statements sent to the slaves is <code>use_slave</code> + <code>use_slave_sql_hint</code> + <code>use_slave_callback</code>.</p> <p>PECL/mysqld_ms 1.1.0 introduces a new concept of chained filters. The statistics is now set by the internal load balancing filter. With version 1.1.0 the load balancing filter is always the last in the filter chain, if used. In future versions a load balancing filter may be followed by other filters causing another change in the meaning of the statistic. If, in the future, a load balancing filter is followed by another filter it is no longer guaranteed that the statement, which increments <code>use_slave</code>, will be executed on the slaves.</p> <p>The meaning for version 1.1.0 is as follows. Number of statements sent to the slaves. Statements directed to a slave by the user filter (an user-defined callback) are not included. The latter are counted by <code>use_slave_callback</code>.</p> | Since 1.0.0. |
| <code>use_master</code> | <p>The semantics of this statistic has changed between 1.0.1 - 1.1.0.</p> <p>The meaning for version 1.0.1 is as follows. Number of statements not considered as read-only by the built-in query analyzer. Neither statements which begin with a SQL hint to force use of master nor statements directed to a master by an user-defined callback are included. The total number of statements sent to the master is <code>use_master</code> + <code>use_master_sql_hint</code> + <code>use_master_callback</code>.</p> <p>PECL/mysqld_ms 1.1.0 introduces a new concept of chained filters. The statistics is now set by the internal load balancing filter. With version 1.1.0 the load balancing filter is always the last in the filter chain, if used. In future versions a load balancing filter may be followed by other filters causing another change in the meaning of the statistic. If, in the future, a load balancing filter is followed by another filter it is no longer guaranteed that the statement, which increments <code>use_master</code>, will be executed on the slaves.</p> <p>The meaning for version 1.1.0 is as follows. Number of statements sent to the masters. Statements directed to a master by the user filter (an user-defined callback) are not included. The latter are counted by <code>use_master_callback</code>.</p> | Since 1.0.0. |
| <code>use_slave_number</code> | <p>Number of statements the built-in query analyzer recommends sending to a slave because they contain no SQL hint to force use of a certain server. The recommendation may be overruled in the following. It is not guaranteed whether the statement will be executed on a slave or not. This is how often the internal <code>is_select</code> function has guessed that a slave shall be used. Please, see also the user space function <code>mysqld_ms_query_is_select</code>.</p> | Since 1.1.0. |

Mysqld replication and load balancing plugin (`mysqld_ms`)

| Statistic | Description | Version |
|-------------------------|---|--------------|
| <code>use_master</code> | Number of statements the built-in query analyzer recommends sending to a master because they contain no SQL hint to force use of a certain server. The recommendation may be overruled in the following. It is not guaranteed whether the statement will be executed on a slave or not. This is how often the internal <code>is_select</code> function has guessed that a master shall be used. Please, see also the user space function <code>mysqld_ms_query_is_select</code> . | Since 1.1.0. |
| <code>use_slave</code> | Number of statements sent to a slave because statement begins with the SQL hint to force use of slave. | Since 1.0.0. |
| <code>use_master</code> | Number of statements sent to a master because statement begins with the SQL hint to force use of master. | Since 1.0.0. |
| <code>use_last</code> | Number of statements sent to server which has run the previous statement, because statement begins with the SQL hint to force use of previously used server. | Since 1.0.0. |
| <code>use_slave</code> | Number of statements sent to a slave because an user-defined callback has chosen a slave server for statement execution. | Since 1.0.0. |
| <code>use_master</code> | Number of statements sent to a master because an user-defined callback has chosen a master server for statement execution. | Since 1.0.0. |
| <code>non_lazy</code> | Number of successfully opened slave connections from configurations not using <code>lazy connections</code> . The total number of successfully opened slave connections is <code>non_lazy_connections_slave_success + lazy_connections_slave_success</code> | Since 1.0.0. |
| <code>non_lazy</code> | Number of failed slave connection attempts from configurations not using <code>lazy connections</code> . The total number of failed slave connection attempts is <code>non_lazy_connections_slave_failure + lazy_connections_slave_failure</code> | Since 1.0.0. |
| <code>non_lazy</code> | Number of successfully opened master connections from configurations not using <code>lazy connections</code> . The total number of successfully opened master connections is <code>non_lazy_connections_master_success + lazy_connections_master_success</code> | Since 1.0.0. |
| <code>non_lazy</code> | Number of failed master connection attempts from configurations not using <code>lazy connections</code> . The total number of failed master connection attempts is <code>non_lazy_connections_master_failure + lazy_connections_master_failure</code> | Since 1.0.0. |
| <code>lazy_conn</code> | Number of successfully opened slave connections from configurations using <code>lazy connections</code> . | Since 1.0.0. |
| <code>lazy_conn</code> | Number of failed slave connection attempts from configurations using <code>lazy connections</code> . | Since 1.0.0. |
| <code>lazy_conn</code> | Number of successfully opened master connections from configurations using <code>lazy connections</code> . | Since 1.0.0. |
| <code>lazy_conn</code> | Number of failed master connection attempts from configurations using <code>lazy connections</code> . | Since 1.0.0. |
| <code>trx_auto</code> | Number of <code>autocommit</code> mode activations via API calls. This figure may be used to monitor activity related to the plugin configuration setting <code>trx_stickiness</code> . If, for example, you want to know if a certain API call invokes the <code>mysqld</code> library function <code>trx_autocommit()</code> , which is a requirement for <code>trx_stickiness</code> , you may call the user API function in | Since 1.0.0. |

| Statistic | Description | Version |
|---|---|--------------|
| | question and check if the statistic has changed. The statistic is modified only by the plugins internal subclassed <code>trx_autocommit()</code> method. | |
| <code>trx_autocommit_disable</code> | Number of <code>autocommit</code> mode deactivations via API calls. | Since 1.0.0. |
| <code>trx_master_redirect</code> | Number of statements redirected to the master while <code>trx_stickiness=master</code> and <code>autocommit</code> mode is disabled. | Since 1.0.0. |
| <code>gtid_autocommit_successful</code> | Number of successful SQL injections in autocommit mode as part of the plugins client-side <code>global transaction id emulation</code> . | Since 1.2.0. |
| <code>gtid_autocommit_failed</code> | Number of failed SQL injections in autocommit mode as part of the plugins client-side <code>global transaction id emulation</code> . | Since 1.2.0. |
| <code>gtid_commit_successful</code> | Number of successful SQL injections in commit mode as part of the plugins client-side <code>global transaction id emulation</code> . | Since 1.2.0. |
| <code>gtid_commit_failed</code> | Number of failed SQL injections in commit mode as part of the plugins client-side <code>global transaction id emulation</code> . | Since 1.2.0. |
| <code>gtid_implicit_successful</code> | Number of successful SQL injections when implicit commit is detected as part of the plugins client-side <code>global transaction id emulation</code> . Implicit commit happens, for example, when autocommit has been turned off, a query is executed and autocommit is enabled again. In that case, the statement will be committed by the server and SQL to maintain is injected before the autocommit is re-enabled. | Since 1.2.0. |
| <code>gtid_implicit_failed</code> | Number of failed SQL injections when implicit commit is detected as part of the plugins client-side <code>global transaction id emulation</code> . Implicit commit happens, for example, when autocommit has been turned off, a query is executed and autocommit is enabled again. In that case, the statement will be committed by the server and SQL to maintain is injected before the autocommit is re-enabled. | Since 1.2.0. |

Examples

Example 22.286. `mysqlnd_ms_get_stats` example

```
<?php
printf("mysqlnd_ms.enable = %d\n", ini_get("mysqlnd_ms.enable"));
printf("mysqlnd_ms.collect_statistics = %d\n", ini_get("mysqlnd_ms.collect_statistics"));
var_dump(mysqlnd_ms_get_stats());
?>
```

The above example will output:

```
mysqlnd_ms.enable = 1
mysqlnd_ms.collect_statistics = 1
array(26) {
  ["use_slave"]=>
  string(1) "0"
  ["use_master"]=>
  string(1) "0"
  ["use_slave_guess"]=>
  string(1) "0"
  ["use_master_guess"]=>
  string(1) "0"
  ["use_slave_sql_hint"]=>
  string(1) "0"
```

```
[ "use_master_sql_hint" ]=>
string(1) "0"
[ "use_last_used_sql_hint" ]=>
string(1) "0"
[ "use_slave_callback" ]=>
string(1) "0"
[ "use_master_callback" ]=>
string(1) "0"
[ "non_lazy_connections_slave_success" ]=>
string(1) "0"
[ "non_lazy_connections_slave_failure" ]=>
string(1) "0"
[ "non_lazy_connections_master_success" ]=>
string(1) "0"
[ "non_lazy_connections_master_failure" ]=>
string(1) "0"
[ "lazy_connections_slave_success" ]=>
string(1) "0"
[ "lazy_connections_slave_failure" ]=>
string(1) "0"
[ "lazy_connections_master_success" ]=>
string(1) "0"
[ "lazy_connections_master_failure" ]=>
string(1) "0"
[ "trx_autocommit_on" ]=>
string(1) "0"
[ "trx_autocommit_off" ]=>
string(1) "0"
[ "trx_master_forced" ]=>
string(1) "0"
[ "gtid_autocommit_injections_success" ]=>
string(1) "0"
[ "gtid_autocommit_injections_failure" ]=>
string(1) "0"
[ "gtid_commit_injections_success" ]=>
string(1) "0"
[ "gtid_commit_injections_failure" ]=>
string(1) "0"
[ "gtid_implicit_commit_injections_success" ]=>
string(1) "0"
[ "gtid_implicit_commit_injections_failure" ]=>
string(1) "0"
}
```

See Also

[Runtime configuration](#)
[mysqlnd_ms.collect_statistics](#)
[mysqlnd_ms.enable](#)
[Monitoring](#)

22.9.6.8.4. `mysqlnd_ms_match_wild`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_ms_match_wild`

Finds whether a table name matches a wildcard pattern or not

Description

```
bool mysqlnd_ms_match_wild(
    string table_name,
```

```
string wildcard);
```

Finds whether a table name matches a wildcard pattern or not.

This function is not of much practical relevance with PECL `mysqlnd_ms` 1.1.0 because the plugin does not support MySQL replication table filtering yet.

Parameters

| | |
|-------------------------|--|
| <code>table_name</code> | The table name to check if it is matched by the wildcard. |
| <code>wildcard</code> | The wildcard pattern to check against the table name. The wildcard pattern supports the same placeholders as MySQL replication filters do.

MySQL replication filters can be configured by using the MySQL Server configuration options <code>--replicate-wild-do-table</code> and <code>--replicate-wild-do-db</code> . Please, consult the MySQL Reference Manual to learn more about this MySQL Server feature.

The supported placeholders are: <ul style="list-style-type: none">• <code>%</code> - zero or more literals• <code>_</code> - one literal Placeholders can be escaped using <code>\</code> . |

Return Values

Returns `TRUE` if `table_name` is matched by `wildcard`. Otherwise, returns `FALSE`.

Examples

Example 22.287. `mysqlnd_ms_match_wild` example

```
<?php
var_dump(mysqlnd_ms_match_wild("schema_name.table_name", "schema%"));
var_dump(mysqlnd_ms_match_wild("abc", "_"));
var_dump(mysqlnd_ms_match_wild("table1", "table_"));
var_dump(mysqlnd_ms_match_wild("asia_customers", "%customers"));
var_dump(mysqlnd_ms_match_wild("funny%table", "funny\%table"));
var_dump(mysqlnd_ms_match_wild("funnytable", "funny%table"));
?>
```

The above example will output:

```
bool(true)
bool(false)
bool(true)
bool(true)
bool(true)
bool(true)
```

22.9.6.8.5. `mysqlnd_ms_query_is_select`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_ms_query_is_select`

Find whether to send the query to the master, the slave or the last used MySQL server

Description

```
int mysqlnd_ms_query_is_select(
    string query);
```

Finds whether to send the query to the master, the slave or the last used MySQL server.

The plugins built-in read/write split mechanism will be used to analyze the query string to make a recommendation where to send the query. The built-in read/write split mechanism is very basic and simple. The plugin will recommend sending all queries to the MySQL replication master server but those which begin with `SELECT`, or begin with a SQL hint which enforces sending the query to a slave server. Due to the basic but fast algorithm the plugin may propose to run some read-only statements such as `SHOW TABLES` on the replication master.

Parameters

query Query string to test.

Return Values

A return value of `MYSQLND_MS_QUERY_USE_MASTER` indicates that the query should be send to the MySQL replication master server. The function returns a value of `MYSQLND_MS_QUERY_USE_SLAVE` if the query can be run on a slave because it is considered read-only. A value of `MYSQLND_MS_QUERY_USE_LAST_USED` is returned to recommend running the query on the last used server. This can either be a MySQL replication master server or a MySQL replication slave server.

If read write splitting has been disabled by setting `mysqlnd_ms.disable_rw_split`, the function will always return `MYSQLND_MS_QUERY_USE_MASTER` or `MYSQLND_MS_QUERY_USE_LAST_USED`.

Examples

Example 22.288. `mysqlnd_ms_query_is_select` example

```
<?php
function is_select($query)
{
    switch (mysqlnd_ms_query_is_select($query))
    {
        case MYSQLND_MS_QUERY_USE_MASTER:
            printf("%s' should be run on the master.\n", $query);
            break;
        case MYSQLND_MS_QUERY_USE_SLAVE:
            printf("%s' should be run on a slave.\n", $query);
            break;
        case MYSQLND_MS_QUERY_USE_LAST_USED:
            printf("%s' should be run on the server that has run the previous query\n", $query);
            break;
        default:
            printf("No suggestion where to run the '%s', fallback to master recommended\n", $query);
            break;
    }
}

is_select("INSERT INTO test(id) VALUES (1)");
is_select("SELECT 1 FROM DUAL");
is_select("/*" . MYSQLND_MS_LAST_USED_SWITCH . "*/SELECT 2 FROM DUAL");
?>
```

The above example will output:

```
INSERT INTO test(id) VALUES (1) should be run on the master.
SELECT 1 FROM DUAL should be run on a slave.
/*ms=last_used*/SELECT 2 FROM DUAL should be run on the server that has run the previous query
```

See Also

Predefined Constants

`user` filter

Runtime configuration

`mysqlnd_ms.disable_rw_split`

`mysqlnd_ms.enable`

22.9.6.8.6. `mysqlnd_ms_set_qos`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_ms_set_qos`

Sets the quality of service needed from the cluster

Description

```
bool mysqlnd_ms_set_qos(
    mixed connection,
    int service_level,
    int service_level_option,
    mixed option_value);
```

Sets the quality of service needed from the cluster. A database cluster delivers a certain quality of service to the user depending on its architecture. A major aspect of the quality of service is the consistency level the cluster can offer. An asynchronous MySQL replication cluster defaults to eventual consistency for slave reads: a slave may serve stale data, current data, or it may have not the requested data at all, because it is not synchronous to the master. In a MySQL replication cluster, only master accesses can give strong consistency, which promises that all clients see each others changes.

PECL/`mysqlnd_ms` hides the complexity of choosing appropriate nodes to achieve a certain level of service from the cluster. The "Quality of Service" filter implements the necessary logic. The filter can either be configured in the plugins configuration file, or at runtime using `mysqlnd_ms_set_qos`.

Similar results can be achieved with PECL `mysqlnd_ms` < 1.2.0, if using SQL hints to force the use of a certain type of node or using the `master_on_write` plugin configuration option. The first requires more code and causes more work on the application side. The latter is less refined than using the quality of service filter. Settings made through the function call can be reversed, as shown in the example below. The example temporarily switches to a higher service level (session consistency, read your writes) and returns back to the clusters default after it has performed all operations that require the better service. This way, read load on the master can be minimized compared to using `master_on_write`, which would continue using the master after the first write.

Parameters

connection

A PECL/`mysqlnd_ms` connection handle to a MySQL server of the type `PDO_MYSQL`, `mysqli` or `ext/mysqli` for which a service level is to be set. The connection handle is obtained when opening a connection with a

| | |
|-----------------------------------|---|
| <code>service_level</code> | <p>host name that matches a <code>mysqlnd_ms</code> configuration file entry using any of the above three MySQL driver extensions.</p> <p>The requested service level:
 <code>MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL</code> ,
 <code>MYSQLND_MS_QOS_CONSISTENCY_SESSION</code> or
 <code>MYSQLND_MS_QOS_CONSISTENCY_STRONG</code> .</p> |
| <code>service_level_option</code> | <p>An option to parameterize the requested service level. The option can either be <code>MYSQLND_MS_QOS_OPTION_GTID</code> or <code>MYSQLND_MS_QOS_OPTION_AGE</code> .</p> <p>The option <code>MYSQLND_MS_QOS_OPTION_GTID</code> can be used to refine the service level <code>MYSQLND_MS_QOS_CONSISTENCY_SESSION</code> . It must be combined with a fourth function parameter, the <code>option_value</code>. The <code>option_value</code> shall be a global transaction ID obtained from <code>mysqlnd_ms_get_last_gtid</code>. If set, the plugin considers both master servers and asynchronous slaves for session consistency (read your writes). Otherwise, only masters are used to achieve session consistency. A slave is considered up-to-date and checked if it has already replicated the global transaction ID from <code>option_value</code>. Please note, searching appropriate slaves is an expensive and slow operation. Use the feature sparsely, if the master cannot handle the read load alone.</p> <p>The <code>MYSQLND_MS_QOS_OPTION_AGE</code> option can be combined with the <code>MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL</code> service level, to filter out asynchronous slaves that lag more seconds behind the master than <code>option_value</code>. If set, the plugin will only consider slaves for reading if <code>SHOW SLAVE STATUS</code> reports <code>Slave_IO_Running=Yes</code>, <code>Slave_SQL_Running=Yes</code> and <code>Seconds_Behind_Master <= option_value</code>. Please note, searching appropriate slaves is an expensive and slow operation. Use the feature sparsely in version 1.2.0. Future versions may improve the algorithm used to identify candidates. Please, see the MySQL reference manual about the precision, accuracy and limitations of the MySQL administrative command <code>SHOW SLAVE STATUS</code>.</p> |
| <code>option_value</code> | <p>Parameter value for the service level option. See also the <code>service_level_option</code> parameter.</p> |

Return Values

Returns `TRUE` if the connections service level has been switched to the requested. Otherwise, returns `FALSE`

Notes



Note

`mysqlnd_ms_set_qos` requires PHP `>= 5.4.0` and PECL `mysqlnd_ms >= 1.2.0`. Internally, it is using a `mysqlnd` library C functionality not available with PHP 5.3.

Examples

Example 22.289. `mysqlnd_ms_set_qos` example

```
<?php
/* Open mysqld_ms connection using mysqli, PDO_MySQL or mysql extension */
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));

/* Session consistency: read your writes */
$ret = mysqld_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION);
if (!$ret)
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Will use master and return fresh data, client can see his last write */
if (!$res = $mysqli->query("SELECT item, price FROM orders WHERE order_id = 1"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

/* Back to default: use of all slaves and masters permitted, stale data can happen */
if (!$mysqli->set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
?>
```

See Also

[mysqld_ms_get_last_gtid](#)
[Service level and consistency concept](#)
[Filter concept](#)

22.9.6.8.7. [mysqld_ms_set_user_pick_server](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqld_ms_set_user_pick_server](#)

Sets a callback for user-defined read/write splitting

Description

```
bool mysqld_ms_set_user_pick_server(
    string function);
```

Sets a callback for user-defined read/write splitting. The plugin will call the callback only if `pick[]=user` is the default rule for server picking in the relevant section of the plugins configuration file.

The plugins built-in read/write query split mechanism decisions can be overwritten in two ways. The easiest way is to prepend the query string with the SQL hints [MYSQLND_MS_MASTER_SWITCH](#) , [MYSQLND_MS_SLAVE_SWITCH](#) or [MYSQLND_MS_LAST_USED_SWITCH](#) . Using SQL hints one can control, for example, whether a query shall be send to the MySQL replication master server or one of the slave servers. By help of SQL hints it is not possible to pick a certain slave server for query execution.

Full control on server selection can be gained using a callback function. Use of a callback is recommended to expert users only because the callback has to cover all cases otherwise handled by the plugin.

The plugin will invoke the callback function for selecting a server from the lists of configured master and slave servers. The callback function inspects the query to run and picks a server for query execution by returning the hosts URI, as found in the master and slave list.

If the lazy connections are enabled and the callback chooses a slave server for which no connection has been established so far and establishing the connection to the slave fails, the plugin will return an error upon the next action on the failed connection, for example, when running a query. It is the responsibility of the application developer to handle the error. For example, the application can re-run the query to trigger a

new server selection and callback invocation. If so, the callback must make sure to select a different slave, or check slave availability, before returning to the plugin to prevent an endless loop.

Parameters

function

The function to be called. Class methods may also be invoked statically using this function by passing `array($classname, $methodname)` to this parameter. Additionally class methods of an object instance may be called by passing `array($objectinstance, $methodname)` to this parameter.

Return Values

Host to run the query on. The host URI is to be taken from the master and slave connection lists passed to the callback function. If callback returns a value neither found in the master nor in the slave connection lists the plugin will fallback to the second pick method configured via the `pick[]` setting in the plugin configuration file. If not second pick method is given, the plugin falls back to the build-in default pick method for server selection.

Notes



Note

`mysqlnd_ms_set_user_pick_server` is available with PECL `mysqlnd_ms` < 1.1.0. It has been replaced by the `user` filter. Please, check the [Change History](#) for upgrade notes.

Examples

Example 22.290. `mysqlnd_ms_set_user_pick_server` example

```
[myapp]
master[] = localhost
slave[] = 192.168.2.27:3306
slave[] = 192.168.78.136:3306
pick[] = user
```

```
<?php

function pick_server($connected, $query, $master, $slaves, $last_used)
{
    static $slave_idx = 0;
    static $num_slaves = NULL;
    if (is_null($num_slaves))
        $num_slaves = count($slaves);

    /* default: fallback to the plugins build-in logic */
    $ret = NULL;

    printf("User has connected to '%s'...\n", $connected);
    printf("... deciding where to run '%s'\n", $query);

    $where = mysqlnd_ms_query_is_select($query);
    switch ($where)
    {
        case MYSQLND_MS_QUERY_USE_MASTER:
            printf("... using master\n");
            $ret = $master[0];
            break;
    }
}
```

```

case MYSQLND_MS_QUERY_USE_SLAVE:
/* SELECT or SQL hint for using slave */
if (strstr($query, "FROM table_on_slave_a_only"))
{
/* a table which is only on the first configured slave */
printf("... access to table available only on slave A detected\n");
$ret = $slaves[0];
}
else
{
/* round robin */
printf("... some read-only query for a slave\n");
$ret = $slaves[$slave_idx++ % $num_slaves];
}
break;
case MYSQLND_MS_QUERY_LAST_USED:
printf("... using last used server\n");
$ret = $last_used;
break;
}

printf("... ret = '%s'\n", $ret);
return $ret;
}

mysqlnd_ms_set_user_pick_server("pick_server");

$mysqli = new mysqli("myapp", "root", "root", "test");

if (!$res = $mysqli->query("SELECT 1 FROM DUAL"))
printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
$res->close();

if (!$res = $mysqli->query("SELECT 2 FROM DUAL"))
printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
$res->close();

if (!$res = $mysqli->query("SELECT * FROM table_on_slave_a_only"))
printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
$res->close();

$mysqli->close();
?>

```

The above example will output:

```

User has connected to 'myapp'...
... deciding where to run 'SELECT 1 FROM DUAL'
... some read-only query for a slave
... ret = 'tcp://192.168.2.27:3306'
User has connected to 'myapp'...
... deciding where to run 'SELECT 2 FROM DUAL'
... some read-only query for a slave
... ret = 'tcp://192.168.78.136:3306'
User has connected to 'myapp'...
... deciding where to run 'SELECT * FROM table_on_slave_a_only'
... access to table available only on slave A detected
... ret = 'tcp://192.168.2.27:3306'

```

See Also

[mysqlnd_ms_query_is_select](#)

[Filter concept](#)

[user filter](#)

22.9.6.9. Change History

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

This change history is a high level summary of selected changes that may impact applications and/or break backwards compatibility.

See also the [CHANGES](#) file in the source distribution for a complete list of changes.

22.9.6.9.1. PECL/mysqlnd_ms 1.5 series

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

1.5.0-alpha

- Release date: under development
- Motto/theme: Tweaking based on user feedback



Note

This is the current development series. All features are at an early stage. Changes may happen at any time without prior notice. Please, do not use this version in production environments.

The documentation may not reflect all changes yet.

Bug fixes

- Fixed #60605 PHP segmentation fault when `mysqlnd_ms` is enabled

Feature changes

- Introduced `node_group` filter. The filter lets you organize servers (master and slaves) in groups. Queries can be directed to a certain group of servers by prefixing the query statement with a SQL hint/comment that contains a groups configuration name. Grouping can be used for partitioning and sharding but also to optimize for cache locality. In case of sharding, think of a group name as a shard key. All queries for a given shard key will be executed on the configured shard. Please, note that sharding setups require both client and server support.

22.9.6.9.2. PECL/mysqlnd_ms 1.4 series

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

1.4.1-beta

- Release date: 08/2012
- Motto/theme: Tweaking based on user feedback

Bug fixes

- Fixed build with PHP 5.5

1.4.0-alpha

- Release date: 07/2012
- Motto/theme: Tweaking based on user feedback

Feature changes

- BC break: Renamed plugin configuration setting [ini_file](#) to [config_file](#). In early versions the plugin configuration file used ini style. Back then the configuration setting was named accordingly. It has now been renamed to reflect the newer file format and to distinguish it from PHP's own ini file (configuration directives file).
- Introduced new default charset setting [server_charset](#) to allow proper escaping before a connection is opened. This is most useful when using lazy connections, which are a default.
- Introduced [wait_for_gtid_timeout](#) setting to throttle slave reads that need session consistency. If global transaction identifier are used and the service level is set to session consistency, the plugin tries to find up-to-date slaves. The slave status check is done by a SQL statement. If nothing else is set, the slave status is checked only one can the search for more up-to-date slaves continues immediately thereafter. Setting [wait_for_gtid_timeout](#) instructs the plugin to poll a slaves status for [wait_for_gtid_timeout](#) seconds if the first execution of the SQL statement has shown that the slave is not up-to-date yet. The poll will be done once per second. This way, the plugin will wait for slaves to catch up and throttle the client.
- New failover strategy [loop_before_master](#). By default the plugin does no failover. It is possible to enable automatic failover if a connection attempt fails. Upto version 1.3 only [master](#) strategy existed to failover to a master if a slave connection fails. [loop_before_master](#) is similar but tries all other slaves before attempting to connect to the master if a slave connection fails.

The number of attempts can be limited using the [max_retries](#) option. Failed hosts can be remembered and skipped in load balancing for the rest of the web request. [max_retries](#) and [remember_failed](#) are considered experimental although decent stability is given. Syntax and semantics may change in the future without prior notice.

22.9.6.9.3. PECL/mysqlnd_ms 1.3 series

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

1.3.2-stable

- Release date: 04/2012
- Motto/theme: see 1.3.0-alpha

Bug fixes

- Fixed problem with multi-master where although in a transaction the queries to the master weren't sticky and were spread all over the masters (RR). Still not sticky for Random. [Random_once](#) is not affected.

1.3.1-beta

- Release date: 04/2012
- Motto/theme: see 1.3.0-alpha

Bug fixes

- Fixed problem with building together with QC.

1.3.0-alpha

- Release date: 04/2012
- Motto/theme: Query caching through quality-of-service concept

The 1.3 series aims to improve the performance of applications and the overall load of an asynchronous MySQL cluster, for example, a MySQL cluster using MySQL Replication. This is done by transparently replacing a slave access with a local cache access, if the application allows it by setting an appropriate quality of service flag. When using MySQL replication a slave can serve stale data. An application using MySQL replication must continue to work correctly with stale data. Given that the application is known to work correctly with stale data, the slave access can transparently be replaced with a local cache access.

[PECL/mysqlnd_qc](#) serves as a cache backend. PECL/mysqlnd_qc supports use of various storage locations, among others main memory, [APC](#) and [MEMCACHE](#).

Feature changes

- Added cache option to quality-of-service (QoS) filter.
 - New configure option `enable-mysqlnd-ms-cache-support`
 - New constant `MYSQLND_MS_HAVE_CACHE_SUPPORT`.
 - New constant `MYSQLND_MS_QOS_OPTION_CACHE` to be used with `mysqlnd_ms_set_qos`.
- Support for built-in global transaction identifier feature of MySQL 5.6.5-m8 or newer.

22.9.6.9.4. PECL/mysqlnd_ms 1.2 series

Copyright 1997-2012 the PHP Documentation Group. [3186]

1.2.1-beta

- Release date: 01/2012
- Motto/theme: see 1.2.0-alpha

Minor test changes.

1.2.0-alpha

- Release date: 11/2011
- Motto/theme: Global Transaction ID injection and quality-of-service concept

In version 1.2 the focus continues to be on supporting MySQL database clusters with asynchronous replication. The plugin tries to make using the cluster introducing a quality-of-service filter which applications can use to define what service quality they need from the cluster. Service levels provided are eventual consistency with optional maximum age/slave lag, session consistency and strong consistency.

Additionally the plugin can do client-side global transaction id injection to make manual master failover easier.

Feature changes

- Introduced quality-of-service (QoS) filter. Service levels provided by QoS filter:
 - eventual consistency, optional option slave lag
 - session consistency, optional option GTID
 - strong consistency

- Added the `mysqlnd_ms_set_qos` function to set the required connection quality at runtime. The new constants related to `mysqlnd_ms_set_qos` are:
 - `MYSQLND_MS_QOS_CONSISTENCY_STRONG`
 - `MYSQLND_MS_QOS_CONSISTENCY_SESSION`
 - `MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL`
 - `MYSQLND_MS_QOS_OPTION_GTID`
 - `MYSQLND_MS_QOS_OPTION_AGE`
- Added client-side global transaction id injection (GTID).
- New statistics related to GTID:
 - `gtid_autocommit_injections_success`
 - `gtid_autocommit_injections_failure`
 - `gtid_commit_injections_success`
 - `gtid_commit_injections_failure`
 - `gtid_implicit_commit_injections_success`
 - `gtid_implicit_commit_injections_failure`
- Added `mysqlnd_ms_get_last_gtid` to fetch the last global transaction id.
- Enabled support for multi master without slaves.

22.9.6.9.5. PECL/mysqlnd_ms 1.1 series

Copyright 1997-2012 the PHP Documentation Group. [3186]

1.1.0

- Release date: 09/2011
- Motto/theme: Cover replication basics with production quality

The 1.1 and 1.0 series expose a similar feature set. Internally, the 1.1 series has been refactored to plan for future feature additions. A new configuration file format has been introduced, and limitations have been lifted. And the code quality and quality assurance has been improved.

Feature changes

- Added the (chainable) [filter concept](#):
 - BC break: `mysqlnd_ms_set_user_pick_server` has been removed. The http://svn.php.net/viewvc/pecl/mysqlnd_ms/trunk/user filter has been introduced to replace it. The filter offers similar functionality, but see below for an explanation of the differences.
- New powerful JSON based configuration syntax.
- [Lazy connections improved](#): security relevant, and state changing commands are covered.
- Support for (native) prepared statements.

- New statistics: [use_master_guess](#), [use_slave_guess](#).
 - BC break: Semantics of statistics changed for [use_slave](#), [use_master](#). Future changes are likely. Please see, [mysqlnd_ms_get_stats](#).
- List of broadcasted messages extended by [ssl_set](#).
- Library calls now monitored to remember settings for lazy connections: [change_user](#), [select_db](#), [set_charset](#), [set_autocommit](#).
- Introduced [mysqlnd_ms.disable_rw_split](#). The configuration setting allows using the load balancing and lazy connection functionality independently of read write splitting.

Bug fixes

- Fixed PECL #22724 - Server switching ([mysqlnd_ms_query_is_select\(\)](#) case sensitive)
- Fixed PECL #22784 - Using [mysql_connect](#) and [mysql_select_db](#) did not work
- Fixed PECL #59982 - Unusable extension with `--enable-mysqlnd-ms-table-filter`. Use of the option is NOT supported. You must not use it. Added note to m4.
- Fixed Bug #60119 - `host="localhost"` lost in [mysqlnd_ms_get_last_used_connection\(\)](#)

The [mysqlnd_ms_set_user_pick_server](#) function was removed, and replaced in favor of a new [user](#) filter. You can no longer set a callback function using [mysqlnd_ms_set_user_pick_server](#) at runtime, but instead have to configure it in the plugins configuration file. The [user](#) filter will pass the same arguments to the callback as before. Therefore, you can continue to use the same procedural function as a callback. It is no longer possible to use static class methods, or class methods of an object instance, as a callback. Doing so will cause the function executing a statement handled by the plugin to emit an `E_RECOVERABLE_ERROR` level error, which might look like: "`(mysqlnd_ms) Specified callback (picker) is not a valid callback.`" Note: this may halt your application.

22.9.6.9.6. PECL/mysqlnd_ms 1.0 series

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

1.0.1-alpha

- Release date: 04/2011
- Motto/theme: bug fix release

1.0.0-alpha

- Release date: 04/2011
- Motto/theme: Cover replication basics to test user feedback

The first release of practical use. It features basic automatic read-write splitting, SQL hints to overrule automatic redirection, load balancing of slave requests, lazy connections, and optional, automatic use of the master after the first write.

The public feature set is close to that of the 1.1 release.

1.0.0-pre-alpha

- Release date: 09/2010
- Motto/theme: Proof of concept

Initial check-in. Essentially a demo of the [mysqlnd](#) plugin API.

22.9.7. Mysqlnd query result cache plugin ([mysqlnd_qc](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

The mysqlnd query result cache plugin adds easy to use client-side query caching to all PHP MySQL extensions using [mysqlnd](#).

As of version PHP 5.3.3 the MySQL native driver for PHP ([mysqlnd](#)) features an internal plugin C API. C plugins, such as the query cache plugin, can extend the functionality of [mysqlnd](#).

Mysqlnd plugins such as the query cache plugin operate transparent from a user perspective. The cache plugin supports all PHP applications and all PHP MySQL extensions ([mysqli](#), [mysql](#), [PDO_MYSQL](#)). It does not change existing APIs.

No significant application changes are required to cache a query. The cache has two operation modes. It will either cache all queries (not recommended) or only those queries marked with a certain SQL hint (recommended).

22.9.7.1. Key Features

Copyright 1997-2012 the PHP Documentation Group. [3186]

- Transparent and therefore easy to use
 - supports all PHP MySQL extensions
 - no API changes
 - very little application changes required
- Flexible invalidation strategy
 - Time-to-Live (TTL)
 - user-defined
- Storage with different scope and life-span
 - Default (Hash, process memory)
 - [APC](#)
 - MEMCACHE
 - sqlite
 - user-defined
- Built-in slam defense to prevent cache stampeding.

22.9.7.2. Limitations

Copyright 1997-2012 the PHP Documentation Group. [3186]

The current 1.0.1 release of PECL [mysqlnd_qc](#) does not support PHP 5.4. Version 1.1.0-alpha lifts this limitation.

Prepared statements and unbuffered queries are fully supported. Thus, the plugin is capable of caching all statements issued with [mysqli](#) or [PDO_MySQL](#), which are the only two PHP MySQL APIs to offer prepared statement support.

22.9.7.3. On the name

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The shortcut [mysqlnd_qc](#) stands for [mysqlnd query cache plugin](#). The name was chosen for a quick-and-dirty proof-of-concept. In the beginning the developers did not expect to continue using the code base. Sometimes PECL/[mysqlnd_qc](#) has also been called [client-side query result set cache](#).

22.9.7.4. Quickstart and Examples

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The [mysqlnd query cache plugin](#) is easy to use. This quickstart will demo typical use-cases, and provide practical advice on getting started.

It is strongly recommended to read the reference sections in addition to the quickstart. It is safe to begin with the quickstart. However, before using the plugin in mission critical environments we urge you to read additionally the background information from the reference sections.

Most of the examples use the [mysqli](#) extension because it is the most feature complete PHP MySQL extension. However, the plugin can be used with any PHP MySQL extension that is using the [mysqlnd](#) library.

22.9.7.4.1. Architecture and Concepts

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The query cache plugin is implemented as a PHP extension. It is written in C and operates under the hood of PHP. During the startup of the PHP interpreter, it gets registered as a [mysqlnd](#) plugin to replace selected [mysqlnd](#) C methods. Hereby, it can change the behaviour of any PHP MySQL extension ([mysqli](#), [PDO_MYSQL](#), [mysql](#)) compiled to use the [mysqlnd](#) library without changing the extensions API. This makes the plugin compatible with each and every PHP MySQL application. Because existing APIs are not changed, it is almost transparent to use. Please, see the [mysqlnd plugin API description](#) for a discussion of the advantages of the plugin architecture and a comparison with proxy based solutions.

Transparent to use

At PHP run time PECL/[mysqlnd_qc](#) can proxy queries send from PHP ([mysqlnd](#)) to the MySQL server. It then inspects the statement string to find whether it shall cache its results. If so, result set is cached using a storage handler and further executions of the statement are served from the cache for a user-defined period. The Time to Live (TTL) of the cache entry can either be set globally or on a per statement basis.

A statement is either cached if the plugin is instructed to cache all statements globally using a or, if the query string starts with the SQL hint (`/*qc=on*/`). The plugin is capable of caching any query issued by calling appropriate API calls of any of the existing PHP MySQL extensions.

Flexible storage: various storage handler

Various storage handler are supported to offer different scopes for cache entries. Different scopes allow for different degrees in sharing cache entries among clients.

- [default](#) (built-in): process memory, scope: process, one or more web requests depending on PHP deployment model used
- [APC](#): shared memory, scope: single server, multiple web requests

- [SQLite](#): memory or file, scope: single server, multiple web requests
- [MEMCACHE](#): main memory, scope: single or multiple server, multiple web requests
- [user](#) (built-in): user-defined - any, scope: user-defined - any

Support for the [APC](#), [SQLite](#) and [MEMCACHE](#) storage handler has to be enabled at compile time. The [default](#) and [user](#) handler are built-in. It is possible to switch between compiled-in storage handlers on a per query basis at run time. However, it is recommended to pick one storage handler and use it for all cache entries.

Built-in slam defense to avoid overloading

To avoid overload situations the cache plugin has a built-in slam defense mechanism. If a popular cache entry expires many clients using the cache entries will try to refresh the cache entry. For the duration of the refresh many clients may access the database server concurrently. In the worst case, the database server becomes overloaded and it takes more and more time to refresh the cache entry, which in turn lets more and more clients try to refresh the cache entry. To prevent this from happening the plugin has a slam defense mechanism. If slam defense is enabled and the plugin detects an expired cache entry it extends the life time of the cache entry before it refreshes the cache entry. This way other concurrent accesses to the expired cache entry are still served from the cache for a certain time. The other concurrent accesses do not trigger a concurrent refresh. Ideally, the cache entry gets refreshed by the client which extended the cache entries lifespan before other clients try to refresh the cache and potentially cause an overload situation.

Unique approach to caching

PECL/mysqlnd_qc has a unique approach to caching result sets that is superior to application based cache solutions. Application based solutions first fetch a result set into PHP variables. Then, the PHP variables are serialized for storage in a persistent cache, and then unserialized when fetching. The mysqlnd query cache stores the raw wire protocol data sent from MySQL to PHP in its cache and replays it, if still valid, on a cache hit. This way, it saves an extra serialization step for a cache put that all application based solutions have to do. It can store the raw wire protocol data in the cache without having to serialize into a PHP variable first and deserializing the PHP variable for storing in the cache again.

22.9.7.4.2. Setup

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The plugin is implemented as a PHP extension. See also the [installation instructions](#) to install the [PECL/mysqlnd_qc](#) extension.

Compile or configure the PHP MySQL extension ([mysqli](#), [PDO_MYSQL](#), [mysql](#)) that you plan to use with support for the [mysqlnd](#) library. PECL/mysqlnd_qc is a plugin for the mysqlnd library. To use the plugin with any of the existing PHP MySQL extensions (APIs), the extension has to use the mysqlnd library.

Then, load the extension into PHP and activate the plugin in the PHP configuration file using the PHP configuration directive named [mysqlnd_qc.enable_qc](#).

Example 22.291. Enabling the plugin (php.ini)

```
mysqlnd_qc.enable_qc=1
```

22.9.7.4.3. Caching queries

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

There are four ways to trigger caching of a query.

- Use of SQL hints on a per query basis
- User supplied callbacks to decide on a per query basis, for example, using `mysqlnd_qc_is_select`
- `mysqlnd_set_cache_condition` for rule based automatic per query decisions
- `mysqlnd_qc.cache_by_default = 1` to cache all queries blindly

Use of SQL hints and `mysqlnd_qc.cache_by_default = 1` are explained below. Please, refer to the function reference on `mysqlnd_qc_is_select` for a description of using a callback and, `mysqlnd_qc_set_cache_condition` on how to set rules for automatic caching.

A SQL hint is a SQL standards compliant comment. As a SQL comment it is ignored by the database. A statement is considered eligible for caching if it either begins with the SQL hint enabling caching or it is a `SELECT` statement.

An individual query which shall be cached must begin with the SQL hint `/*qc=on*/`. It is recommended to use the PHP constant `MYSQLND_QC_ENABLE_SWITCH` instead of using the string value.

- not eligible for caching and not cached: `INSERT INTO test(id) VALUES (1)`
- not eligible for caching and not cached: `SHOW ENGINES`
- eligible for caching but uncached: `SELECT id FROM test`
- eligible for caching and cached: `/*qc=on*/SELECT id FROM test`

The examples `SELECT` statement string is prefixed with the `MYSQLND_QC_ENABLE_SWITCH` SQL hint to enable caching of the statement. The SQL hint must be given at the very beginning of the statement string to enable caching.

Example 22.292. Using the `MYSQLND_QC_ENABLE_SWITCH` SQL hint

```
mysqlnd_qc.enable_qc=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");

/* Will be cached because of the SQL hint */
$start = microtime(true);
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 1");

var_dump($res->fetch_assoc());
$res->free();

printf("Total time uncached query: %.6fs\n", microtime(true) - $start);

/* Cache hit */
$start = microtime(true);
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 1");

var_dump($res->fetch_assoc());
$res->free();
```

```
printf("Total time cached query: %.6fs\n", microtime(true) - $start);
?>
```

The above examples will output something similar to:

```
array(1) {
  ["id"]=>
    string(1) "1"
}
Total time uncached query: 0.000740s
array(1) {
  ["id"]=>
    string(1) "1"
}
Total time cached query: 0.000098s
```

If nothing else is configured, as it is the case in the quickstart example, the plugin will use the built-in `default` storage handler. The `default` storage handler uses process memory to hold a cache entry. Depending on the PHP deployment model, a PHP process may serve one or more web requests. Please, consult the web server manual for details. Details make no difference for the examples given in the quickstart.

The query cache plugin will cache all queries regardless if the query string begins with the SQL hint which enables caching or not, if the PHP configuration directive `mysqlnd_qc.cache_by_default` is set to `1`. The setting `mysqlnd_qc.cache_by_default` is evaluated by the core of the query cache plugins. Neither the built-in nor user-defined storage handler can overrule the setting.

The SQL hint `/*qc=off*/` can be used to disable caching of individual queries if `mysqlnd_qc.cache_by_default = 1`. It is recommended to use the PHP constant `MYSQLND_QC_DISABLE_SWITCH` instead of using the string value.

Example 22.293. Using the `MYSQLND_QC_DISABLE_SWITCH` SQL hint

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.cache_by_default=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");

/* Will be cached although no SQL hint is present because of mysqlnd_qc.cache_by_default = 1 */
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();

$mysqli->query("DELETE FROM test WHERE id = 1");

/* Cache hit - no automatic invalidation and still valid! */
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();

/* Cache miss - query must not be cached because of the SQL hint */
```

```
$res = $mysqli->query("/*" . MYSQLND_QC_DISABLE_SWITCH . "*/SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
?>
```

The above examples will output:

```
array(1) {
  ["id"]=>
  string(1) "1"
}
array(1) {
  ["id"]=>
  string(1) "1"
}
NULL
```

PECL/mysqlnd_qc forbids caching of statements for which at least one column from the statements result set shows no table name in its meta data by default. This is usually the case for columns originating from SQL functions such as `NOW()` or `LAST_INSERT_ID()`. The policy aims to prevent pitfalls if caching by default is used.

Example 22.294. Example showing which type of statements are not cached

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.cache_by_default=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1)");

for ($i = 0; $i < 3; $i++) {

    $start = microtime(true);

    /* Note: statement will not be cached because of NOW() use */
    $res = $mysqli->query("SELECT id, NOW() AS _time FROM test");
    $row = $res->fetch_assoc();

    /* dump results */
    var_dump($row);

    printf("Total time: %.6fs\n", microtime(true) - $start);

    /* pause one second */
    sleep(1);
}
?>
```

The above examples will output something similar to:

```

array(2) {
  ["id"]=>
  string(1) "1"
  ["_time"]=>
  string(19) "2012-01-11 15:43:10"
}
Total time: 0.000540s
array(2) {
  ["id"]=>
  string(1) "1"
  ["_time"]=>
  string(19) "2012-01-11 15:43:11"
}
Total time: 0.000555s
array(2) {
  ["id"]=>
  string(1) "1"
  ["_time"]=>
  string(19) "2012-01-11 15:43:12"
}
Total time: 0.000549s

```

It is possible to enable caching for all statements including those which has columns in their result set for which MySQL reports no table, such as the statement from the example. Set `mysqlnd_qc.cache_no_table = 1` to enable caching of such statements. Please, note the difference in the measured times for the above and below examples.

Example 22.295. Enabling caching for all statements using the `mysqlnd_qc.cache_no_table` ini setting

```

mysqlnd_qc.enable_qc=1
mysqlnd_qc.cache_by_default=1
mysqlnd_qc.cache_no_table=1

```

```

<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1)");

for ($i = 0; $i < 3; $i++) {

    $start = microtime(true);

    /* Note: statement will not be cached because of NOW() use */
    $res = $mysqli->query("SELECT id, NOW() AS _time FROM test");
    $row = $res->fetch_assoc();

    /* dump results */
    var_dump($row);

    printf("Total time: %.6fs\n", microtime(true) - $start);

    /* pause one second */
    sleep(1);
}
?>

```

The above examples will output something similar to:

```
array(2) {
  ["id"]=>
  string(1) "1"
  ["_time"]=>
  string(19) "2012-01-11 15:47:45"
}
Total time: 0.000546s
array(2) {
  ["id"]=>
  string(1) "1"
  ["_time"]=>
  string(19) "2012-01-11 15:47:45"
}
Total time: 0.000187s
array(2) {
  ["id"]=>
  string(1) "1"
  ["_time"]=>
  string(19) "2012-01-11 15:47:45"
}
Total time: 0.000167s
```



Note

Although `mysqlnd_qc.cache_no_table = 1` has been created for use with `mysqlnd_qc.cache_by_default = 1` it is bound it. The plugin will evaluate the `mysqlnd_qc.cache_no_table` whenever a query is to be cached, no matter whether caching has been enabled using a SQL hint or any other measure.

22.9.7.4.4. Setting the TTL

Copyright 1997-2012 the PHP Documentation Group. [3186]

The default invalidation strategy of the query cache plugin is Time to Live (TTL). The built-in storage handlers will use the default TTL defined by the PHP configuration value `mysqlnd_qc.ttl` unless the query string contains a hint for setting a different TTL. The TTL is specified in seconds. By default cache entries expire after 30 seconds

The example sets `mysqlnd_qc.ttl=3` to cache statements for three seconds by default. Every second it updates a database table record to hold the current time and executes a `SELECT` statement to fetch the record from the database. The `SELECT` statement is cached for three seconds because it is prefixed with the SQL hint enabling caching. The output verifies that the query results are taken from the cache for the duration of three seconds before they are refreshed.

Example 22.296. Setting the TTL with the `mysqlnd_qc.ttl` ini setting

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.ttl=3
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id VARCHAR(255))");
```

```

for ($i = 0; $i < 7; $i++) {

    /* update DB row */
    if (!$mysqli->query("DELETE FROM test") ||
        !$mysqli->query("INSERT INTO test(id) VALUES (NOW())"))
        /* Of course, a real-life script should do better error handling */
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));

    /* select latest row but cache results */
    $query = "/*" . MYSQLND_QC_ENABLE_SWITCH . "*/";
    $query .= "SELECT id AS _time FROM test";
    if (!$res = $mysqli->query($query)) ||
        !($row = $res->fetch_assoc()))
    {
        printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
    }
    $res->free();
    printf("Wall time %s - DB row time %s\n", date("H:i:s"), $row['_time']);

    /* pause one second */
    sleep(1);
}
?>

```

The above examples will output something similar to:

```

Wall time 14:55:59 - DB row time 2012-01-11 14:55:59
Wall time 14:56:00 - DB row time 2012-01-11 14:55:59
Wall time 14:56:01 - DB row time 2012-01-11 14:55:59
Wall time 14:56:02 - DB row time 2012-01-11 14:56:02
Wall time 14:56:03 - DB row time 2012-01-11 14:56:02
Wall time 14:56:04 - DB row time 2012-01-11 14:56:02
Wall time 14:56:05 - DB row time 2012-01-11 14:56:05

```

As can be seen from the example, any **TTL** based cache can serve stale data. Cache entries are not automatically invalidated, if underlying data changes. Applications using the default **TTL** invalidation strategy must be able to work correctly with stale data.

A user-defined cache storage handler can implement any invalidation strategy to work around this limitation.

The default **TTL** can be overruled using the SQL hint `/*qc_tt=seconds*/`. The SQL hint must be appear immediately after the SQL hint which enables caching. It is recommended to use the PHP constant `MYSQLND_QC_TTL_SWITCH` instead of using the string value.

Example 22.297. Setting TTL with SQL hints

```

<?php
$start = microtime(true);

/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");

printf("Default TTL\tt: %d seconds\n", ini_get("mysqlnd_qc.ttl"));

/* Will be cached for 2 seconds */
$sql = sprintf("/*%s*//*%s%d*/SELECT id FROM test WHERE id = 1", MYSQLND_QC_ENABLE_SWITCH, MYSQLND_QC_TTL_

```

```
$res = $mysqli->query($sql);

var_dump($res->fetch_assoc());
$res->free();

$mysqli->query("DELETE FROM test WHERE id = 1");
sleep(1);

/* Cache hit - no automatic invalidation and still valid! */
$res = $mysqli->query($sql);
var_dump($res->fetch_assoc());
$res->free();

sleep(2);

/* Cache miss - cache entry has expired */
$res = $mysqli->query($sql);
var_dump($res->fetch_assoc());
$res->free();

printf("Script runtime\t: %d seconds\n", microtime(true) - $start);
?>
```

The above examples will output something similar to:

```
Default TTL      : 30 seconds
array(1) {
  ["id"]=>
    string(1) "1"
}
array(1) {
  ["id"]=>
    string(1) "1"
}
NULL
Script runtime   : 3 seconds
```

22.9.7.4.5. Pattern based caching

Copyright 1997-2012 the PHP Documentation Group. [3186]

An application has three options for telling PECL/`mysqlnd_qc` whether a particular statement shall be used. The most basic approach is to cache all statements by setting `mysqlnd_qc.cache_by_default = 1`. This approach is often of little practical value. But it enables users to make a quick estimation about the maximum performance gains from caching. An application designed to use a cache may be able to prefix selected statements with the appropriate SQL hints. However, altering an applications source code may not always be possible or desired, for example, to avoid problems with software updates. Therefore, PECL/`mysqlnd_qc` allows setting a callback which decides if a query is to be cached.

The callback is installed with the `mysqlnd_qc_set_is_select` function. The callback is given the statement string of every statement inspected by the plugin. Then, the callback can decide whether to cache the function. The callback is supposed to return `FALSE` if the statement shall not be cached. A return value of `TRUE` makes the plugin try to add the statement into the cache. The cache entry will be given the default TTL (`mysqlnd_qc.ttl`). If the callback returns a numerical value it is used as the TTL instead of the global default.

Example 22.298. Setting a callback with `mysqlnd_qc_set_is_select`

```
mysqld_qc.enable_qc=1
mysqld_qc.collect_statistics=1
```

```
<?php
/* callback which decides if query is cached */
function is_select($query) {
    static $patterns = array(
        /* true - use default from mysqld_qc.ttl */
        "@SELECT\s+.*\s+FROM\s+test@ismU" => true,
        /* 3 - use TTL = 3 seconds */
        "@SELECT\s+.*\s+FROM\s+news@ismU" => 3
    );

    /* check if query does match pattern */
    foreach ($patterns as $pattern => $ttl) {
        if (preg_match($pattern, $query)) {
            printf("is_select(%45s): cache\n", $query);
            return $ttl;
        }
    }
    printf("is_select(%45s): do not cache\n", $query);
    return false;
}
/* install callback */
mysqld_qc_set_is_select("is_select");

/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");

/* cache put */
$mysqli->query("SELECT id FROM test WHERE id = 1");
/* cache hit */
$mysqli->query("SELECT id FROM test WHERE id = 1");
/* cache put */
$mysqli->query("SELECT * FROM test");

$stats = mysqld_qc_get_core_stats();
printf("Cache put: %d\n", $stats['cache_put']);
printf("Cache hit: %d\n", $stats['cache_hit']);
?>
```

The above examples will output something similar to:

```
is_select(          DROP TABLE IF EXISTS test): do not cache
is_select(          CREATE TABLE test(id INT)): do not cache
is_select(  INSERT INTO test(id) VALUES (1), (2), (3)): do not cache
is_select(          SELECT id FROM test WHERE id = 1): cache
is_select(          SELECT id FROM test WHERE id = 1): cache
is_select(          SELECT * FROM test): cache
Cache put: 2
Cache hit: 1
```

The examples callback tests if a statement string matches a pattern. If this is the case, it either returns [TRUE](#) to cache the statement using the global default TTL or an alternative TTL.

To minimize application changes the callback can put into and registered in an auto prepend file.

22.9.7.4.6. Slam defense

Copyright 1997-2012 the PHP Documentation Group. [3186]

A badly designed cache can do more harm than good. In the worst case a cache can increase database server load instead of minimizing it. An overload situation can occur if a highly shared cache entry expires (cache stampeding).

Cache entries are shared and reused to a different degree depending on the storage used. The default storage handler stores cache entries in process memory. Thus, a cache entry can be reused for the life-span of a process. Other PHP processes cannot access it. If Memcache is used, a cache entry can be shared among multiple PHP processes and even among multiple machines, depending on the set up being used.

If a highly shared cache entry stored, for example, in Memcache expires, many clients get a cache miss. Many client requests can no longer be served from the cache but try to run the underlying query on the database server. Until the cache entry is refreshed, more and more clients contact the database server. In the worst case, a total loss of service is the result.

The overload can be avoided using a storage handler which limits the reuse of cache entries to few clients. Then, at the average, it's likely that only a limited number of clients will try to refresh a cache entry concurrently.

Additionally, the built-in slam defense mechanism can and should be used. If slam defense is activated an expired cache entry is given an extended life time. The first client getting a cache miss for the expired cache entry tries to refresh the cache entry within the extended life time. All other clients requesting the cache entry are temporarily served from the cache although the original `TTL` of the cache entry has expired. The other clients will not experience a cache miss before the extended life time is over.

Example 22.299. Enabling the slam defense mechanism

```
mysqlnd_qc.slam_defense=1
mysqlnd_qc.slam_defense_ttl=1
```

The slam defense mechanism is enabled with the PHP configuration directive `mysqlnd_qc.slam_defense`. The extended life time of a cache entry is set with `mysqlnd_qc.slam_defense_ttl`.

The function `mysqlnd_qc_get_core_stats` returns an array of statistics. The statistics `slam_stale_refresh` and `slam_stale_hit` are incremented if slam defense takes place.

It is not possible to give a one-fits-all recommendation on the slam defense configuration. Users are advised to monitor and test their setup and derive settings accordingly.

22.9.7.4.7. Finding cache candidates

Copyright 1997-2012 the PHP Documentation Group. [3186]

A statement should be considered for caching if it is executed often and has a long run time. Cache candidates are found by creating a list of statements sorted by the product of the number of executions multiplied by the statements run time. The function `mysqlnd_qc_get_query_trace_log` returns a query log which helps with the task.

Collecting a query trace is a slow operation. Thus, it is disabled by default. The PHP configuration directive `mysqlnd_qc.collect_query_trace` is used to enable it. The function trace contains one entry for every query issued before the function is called.

Example 22.300. Collecting a query trace

```
mysqld_qc.enable_qc=1
mysqld_qc.collect_query_trace=1
```

```
<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");

/* dummy queries to fill the query trace */
for ($i = 0; $i < 2; $i++) {
    $res = $mysqli->query("SELECT 1 AS _one FROM DUAL");
    $res->free();
}

/* dump trace */
var_dump(mysqld_qc_get_query_trace_log());
?>
```

The above examples will output:

```
array(2) {
  [0]=>
  array(8) {
    ["query"]=>
    string(26) "SELECT 1 AS _one FROM DUAL"
    ["origin"]=>
    string(102) "#0 qc.php(7): mysqli->query('SELECT 1 AS _on...')
#1 {main}"
    ["run_time"]=>
    int(0)
    ["store_time"]=>
    int(25)
    ["eligible_for_caching"]=>
    bool(false)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
    bool(false)
  }
  [1]=>
  array(8) {
    ["query"]=>
    string(26) "SELECT 1 AS _one FROM DUAL"
    ["origin"]=>
    string(102) "#0 qc.php(7): mysqli->query('SELECT 1 AS _on...')
#1 {main}"
    ["run_time"]=>
    int(0)
    ["store_time"]=>
    int(8)
    ["eligible_for_caching"]=>
    bool(false)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
```

```

    bool(false)
  }
}

```

Assorted information is given in the trace. Among them timings and the origin of the query call. The origin property holds a code backtrace to identify the source of the query. The depth of the backtrace can be limited with the PHP configuration directive `mysqlnd_qc.query_trace_bt_depth`. The default depth is 3.

Example 22.301. Setting the backtrace depth with the `mysqlnd_qc.query_trace_bt_depth` ini setting

```

mysqlnd_qc.enable_qc=1
mysqlnd_qc.collect_query_trace=1

```

```

<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");

/* dummy queries to fill the query trace */
for ($i = 0; $i < 3; $i++) {
    $res = $mysqli->query("SELECT id FROM test WHERE id = " . $mysqli->real_escape_string($i));
    $res->free();
}

$trace = mysqlnd_qc_get_query_trace_log();
$summary = array();
foreach ($trace as $entry) {
    if (!isset($summary[$entry['query']])) {
        $summary[$entry['query']] = array(
            "executions" => 1,
            "time"        => $entry['run_time'] + $entry['store_time'],
        );
    } else {
        $summary[$entry['query']]['executions']++;
        $summary[$entry['query']]['time'] += $entry['run_time'] + $entry['store_time'];
    }
}

foreach ($summary as $query => $details) {
    printf("%45s: %5dms (%dx)\n",
        $query, $details['time'], $details['executions']);
}
?>

```

The above examples will output something similar to:

```

                DROP TABLE IF EXISTS test:      0ms (1x)
                CREATE TABLE test(id INT):      0ms (1x)
    INSERT INTO test(id) VALUES (1), (2), (3):  0ms (1x)
                SELECT id FROM test WHERE id = 0: 25ms (1x)
                SELECT id FROM test WHERE id = 1: 10ms (1x)
                SELECT id FROM test WHERE id = 2:  9ms (1x)

```

22.9.7.4.8. Measuring cache efficiency

Copyright 1997-2012 the PHP Documentation Group. [3186]

PECL/`mysqlnd_qc` offers three ways to measure the cache efficiency. The function `mysqlnd_qc_get_normalized_query_trace_log` returns statistics aggregated by the normalized query string, `mysqlnd_qc_get_cache_info` gives storage handler specific information which includes a list of all cached items, depending on the storage handler. Additionally, the core of PECL/`mysqlnd_qc` collects high-level summary statistics aggregated per PHP process. The high-level statistics are returned by `mysqlnd_qc_get_core_stats`.

The functions `mysqlnd_qc_get_normalized_query_trace_log` and `mysqlnd_qc_get_core_stats` will not collect data unless data collection has been enabled through their corresponding PHP configuration directives. Data collection is disabled by default for performance considerations. It is configurable with the `mysqlnd_qc.time_statistics` option, which determines if timing information should be collected. Collection of time statistics is enabled by default but only performed if data collection as such has been enabled. Recording time statistics causes extra system calls. In most cases, the benefit of the monitoring outweighs any potential performance penalty of the additional system calls.

Example 22.302. Collecting statistics data with the `mysqlnd_qc.time_statistics` ini setting

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.collect_statistics=1
```

```
<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");

/* dummy queries */
for ($i = 1; $i <= 4; $i++) {
    $query = sprintf("/*%s*/SELECT id FROM test WHERE id = %d", MYSQLND_QC_ENABLE_SWITCH, $i % 2);
    $res = $mysqli->query($query);

    $res->free();
}

var_dump(mysqlnd_qc_get_core_stats());
?>
```

The above examples will output something similar to:

```
array(26) {
  ["cache_hit"]=>
  string(1) "2"
  ["cache_miss"]=>
  string(1) "2"
  ["cache_put"]=>
  string(1) "2"
  ["query_should_cache"]=>
  string(1) "4"
  ["query_should_not_cache"]=>
  string(1) "3"
```

```

["query_not_cached"]=>
string(1) "3"
["query_could_cache"]=>
string(1) "4"
["query_found_in_cache"]=>
string(1) "2"
["query_uncached_other"]=>
string(1) "0"
["query_uncached_no_table"]=>
string(1) "0"
["query_uncached_no_result"]=>
string(1) "0"
["query_uncached_use_result"]=>
string(1) "0"
["query_aggr_run_time_cache_hit"]=>
string(2) "28"
["query_aggr_run_time_cache_put"]=>
string(3) "900"
["query_aggr_run_time_total"]=>
string(3) "928"
["query_aggr_store_time_cache_hit"]=>
string(2) "14"
["query_aggr_store_time_cache_put"]=>
string(2) "40"
["query_aggr_store_time_total"]=>
string(2) "54"
["receive_bytes_recorded"]=>
string(3) "136"
["receive_bytes_replayed"]=>
string(3) "136"
["send_bytes_recorded"]=>
string(2) "84"
["send_bytes_replayed"]=>
string(2) "84"
["slam_stale_refresh"]=>
string(1) "0"
["slam_stale_hit"]=>
string(1) "0"
["request_counter"]=>
int(1)
["process_hash"]=>
int(1929695233)
}

```

For a quick overview, call [mysqlnd_qc_get_core_stats](#). It delivers cache usage, cache timing and traffic related statistics. Values are aggregated on a per process basis for all queries issued by any PHP MySQL API call.

Some storage handler, such as the default handler, can report cache entries, statistics related to the entries and meta data for the underlying query through the [mysqlnd_qc_get_cache_info](#) function. Please note, that the information returned depends on the storage handler. Values are aggregated on a per process basis.

Example 22.303. Example [mysqlnd_qc_get_cache_info](#) usage

```
mysqlnd_qc.enable_qc=1
```

```

<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");

```

```

$mysqlcli->query("DROP TABLE IF EXISTS test");
$mysqlcli->query("CREATE TABLE test(id INT)");
$mysqlcli->query("INSERT INTO test(id) VALUES (1), (2), (3)");

/* dummy queries to fill the query trace */
for ($i = 1; $i <= 4; $i++) {
    $query = sprintf("/*%s*/SELECT id FROM test WHERE id = %d", MYSQLND_QC_ENABLE_SWITCH, $i % 2);
    $res = $mysqlcli->query($query);

    $res->free();
}

var_dump(mysqlnd_qc_get_cache_info());
?>

```

The above examples will output something similar to:

```

array(4) {
  ["num_entries"]=>
  int(2)
  ["handler"]=>
  string(7) "default"
  ["handler_version"]=>
  string(5) "1.0.0"
  ["data"]=>
  array(2) {
    ["Localhost via UNIX socket
3306
root
test|/*qc=on*/SELECT id FROM test WHERE id = 1"]=>
    array(2) {
      ["statistics"]=>
      array(11) {
        ["rows"]=>
        int(1)
        ["stored_size"]=>
        int(71)
        ["cache_hits"]=>
        int(1)
        ["run_time"]=>
        int(391)
        ["store_time"]=>
        int(27)
        ["min_run_time"]=>
        int(16)
        ["max_run_time"]=>
        int(16)
        ["min_store_time"]=>
        int(8)
        ["max_store_time"]=>
        int(8)
        ["avg_run_time"]=>
        int(8)
        ["avg_store_time"]=>
        int(4)
      }
      ["metadata"]=>
      array(1) {
        [0]=>
        array(8) {
          ["name"]=>
          string(2) "id"
          ["orig_name"]=>
          string(2) "id"

```

```

    ["table"]=>
    string(4) "test"
    ["orig_table"]=>
    string(4) "test"
    ["db"]=>
    string(4) "test"
    ["max_length"]=>
    int(1)
    ["length"]=>
    int(11)
    ["type"]=>
    int(3)
  }
}
["Localhost via UNIX socket
3306
root
test|/*qc=on*/SELECT id FROM test WHERE id = 0"]=>
array(2) {
  ["statistics"]=>
  array(11) {
    ["rows"]=>
    int(0)
    ["stored_size"]=>
    int(65)
    ["cache_hits"]=>
    int(1)
    ["run_time"]=>
    int(299)
    ["store_time"]=>
    int(13)
    ["min_run_time"]=>
    int(11)
    ["max_run_time"]=>
    int(11)
    ["min_store_time"]=>
    int(6)
    ["max_store_time"]=>
    int(6)
    ["avg_run_time"]=>
    int(5)
    ["avg_store_time"]=>
    int(3)
  }
  ["metadata"]=>
  array(1) {
    [0]=>
    array(8) {
      ["name"]=>
      string(2) "id"
      ["orig_name"]=>
      string(2) "id"
      ["table"]=>
      string(4) "test"
      ["orig_table"]=>
      string(4) "test"
      ["db"]=>
      string(4) "test"
      ["max_length"]=>
      int(0)
      ["length"]=>
      int(11)
      ["type"]=>
      int(3)
    }
  }
}

```

```
}
}
}
```

It is possible to further break down the granularity of statistics to the level of the normalized statement string. The normalized statement string is the statements string with all parameters replaced with question marks. For example, the two statements `SELECT id FROM test WHERE id = 0` and `SELECT id FROM test WHERE id = 1` are normalized into `SELECT id FROM test WHERE id = ?`. Their both statistics are aggregated into one entry for `SELECT id FROM test WHERE id = ?`.

Example 22.304. Example `mysqld_qc_get_normalized_query_trace_log` usage

```
mysqld_qc.enable_qc=1
mysqld_qc.collect_normalized_query_trace=1
```

```
<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");

/* dummy queries to fill the query trace */
for ($i = 1; $i <= 4; $i++) {
    $query = sprintf("/*%s*/SELECT id FROM test WHERE id = %d", MYSQLND_QC_ENABLE_SWITCH, $i % 2);
    $res = $mysqli->query($query);

    $res->free();
}

var_dump(mysqld_qc_get_normalized_query_trace_log());
?>
```

The above examples will output something similar to:

```
array(4) {
  [0]=>
  array(9) {
    ["query"]=>
    string(25) "DROP TABLE IF EXISTS test"
    ["occurrences"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["avg_run_time"]=>
    int(0)
    ["min_run_time"]=>
    int(0)
    ["max_run_time"]=>
    int(0)
    ["avg_store_time"]=>
    int(0)
    ["min_store_time"]=>
    int(0)
    ["max_store_time"]=>
    int(0)
  }
}
```

```
[1]=>
array(9) {
  ["query"]=>
string(27) "CREATE TABLE test (id INT )"
  ["occurrences"]=>
int(0)
  ["eligible_for_caching"]=>
bool(false)
  ["avg_run_time"]=>
int(0)
  ["min_run_time"]=>
int(0)
  ["max_run_time"]=>
int(0)
  ["avg_store_time"]=>
int(0)
  ["min_store_time"]=>
int(0)
  ["max_store_time"]=>
int(0)
}
[2]=>
array(9) {
  ["query"]=>
string(46) "INSERT INTO test (id ) VALUES ( ? ), ( ? ), ( ? )"
  ["occurrences"]=>
int(0)
  ["eligible_for_caching"]=>
bool(false)
  ["avg_run_time"]=>
int(0)
  ["min_run_time"]=>
int(0)
  ["max_run_time"]=>
int(0)
  ["avg_store_time"]=>
int(0)
  ["min_store_time"]=>
int(0)
  ["max_store_time"]=>
int(0)
}
[3]=>
array(9) {
  ["query"]=>
string(31) "SELECT id FROM test WHERE id =?"
  ["occurrences"]=>
int(4)
  ["eligible_for_caching"]=>
bool(true)
  ["avg_run_time"]=>
int(179)
  ["min_run_time"]=>
int(11)
  ["max_run_time"]=>
int(393)
  ["avg_store_time"]=>
int(12)
  ["min_store_time"]=>
int(7)
  ["max_store_time"]=>
int(25)
}
}
```

The source distribution of PECL/mysqlnd_qc contains a directory [web/](#) in which web based monitoring scripts can be found which give an example how to write a cache monitor. Please, follow the instructions given in the source.

Since PECL/mysqlnd_qc 1.1.0 it is possible to write statistics into a log file. Please, see [mysqlnd_qc.collect_statistics_log_file](#).

22.9.7.4.9. Beyond TTL: user-defined storage

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The query cache plugin supports the use of user-defined storage handler. User-defined storage handler can use arbitrarily complex invalidation algorithms and support arbitrary storage media.

All user-defined storage handlers have to provide a certain interface. The functions of the user-defined storage handler will be called by the core of the cache plugin. The necessary interface consists of seven public functions. Both procedural and object oriented user-defined storage handler must implement the same set of functions.

Example 22.305. Using a user-defined storage handler

```
<?php
/* Enable default caching of all statements */
ini_set("mysqlnd_qc.cache_by_default", 1);

/* Procedural user defined storage handler functions */

$__cache = array();

function get_hash($host_info, $port, $user, $db, $query) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());

    return md5(sprintf("%s%s%s%s", $host_info, $port, $user, $db, $query));
}

function find_query_in_cache($key) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());

    if (isset($__cache[$key])) {
        $tmp = $__cache[$key];
        if ($tmp["valid_until"] < time()) {
            unset($__cache[$key]);
            $ret = NULL;
        } else {
            $ret = $__cache[$key]["data"];
        }
    } else {
        $ret = NULL;
    }

    return $ret;
}

function return_to_cache($key) {
    /*
     * Called on cache hit after cached data has been processed,
     * may be used for reference counting
     */
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());
}

function add_query_to_cache_if_not_exists($key, $data, $ttl, $run_time, $store_time, $row_count) {
```

```

global $__cache;
printf("\t%s(%d)\n", __FUNCTION__, func_num_args());

$__cache[$key] = array(
    "data"           => $data,
    "row_count"      => $row_count,
    "valid_until"    => time() + $ttl,
    "hits"           => 0,
    "run_time"       => $run_time,
    "store_time"     => $store_time,
    "cached_run_times" => array(),
    "cached_store_times" => array(),
);

return TRUE;
}

function query_is_select($query) {
    printf("\t%s('%s'):", __FUNCTION__, $query);

    $ret = FALSE;
    if (strstr($query, "SELECT") !== FALSE) {
        /* cache for 5 seconds */
        $ret = 5;
    }

    printf("%s\n", (FALSE === $ret) ? "FALSE" : $ret);
    return $ret;
}

function update_query_run_time_stats($key, $run_time, $store_time) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());

    if (isset($__cache[$key])) {
        $__cache[$key]['hits']++;
        $__cache[$key]["cached_run_times"][] = $run_time;
        $__cache[$key]["cached_store_times"][] = $store_time;
    }
}

function get_stats($key = NULL) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());

    if ($key && isset($__cache[$key])) {
        $stats = $__cache[$key];
    } else {
        $stats = array();
        foreach ($__cache as $key => $details) {
            $stats[$key] = array(
                'hits'           => $details['hits'],
                'bytes'          => strlen($details['data']),
                'uncached_run_time' => $details['run_time'],
                'cached_run_time' => (count($details['cached_run_times']))
                                   ? array_sum($details['cached_run_times']) / count($details['cached_run_t
                                   : 0,
            );
        }
    }

    return $stats;
}

function clear_cache() {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());

```

```

    $__cache = array();
    return TRUE;
}

/* Install procedural user-defined storage handler */
if (!mysqlnd_qc_set_user_handlers("get_hash", "find_query_in_cache",
    "return_to_cache", "add_query_to_cache_if_not_exists",
    "query_is_select", "update_query_run_time_stats", "get_stats", "clear_cache")) {

    printf("Failed to install user-defined storage handler\n");
}

/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");

printf("\nCache put/cache miss\n");

$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();

/* Delete record to verify we get our data from the cache */
$mysqli->query("DELETE FROM test WHERE id = 1");

printf("\nCache hit\n");

$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();

printf("\nDisplay cache statistics\n");
var_dump(mysqlnd_qc_get_cache_info());

printf("\nFlushing cache, cache put/cache miss");
var_dump(mysqlnd_qc_clear_cache());

$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
?>

```

The above examples will output something similar to:

```

query_is_select('DROP TABLE IF EXISTS test'): FALSE
query_is_select('CREATE TABLE test(id INT)': FALSE
query_is_select('INSERT INTO test(id) VALUES (1), (2)': FALSE

Cache put/cache miss
query_is_select('SELECT id FROM test WHERE id = 1'): 5
get_hash(5)
find_query_in_cache(1)
add_query_to_cache_if_not_exists(6)
array(1) {
  ["id"]=>
  string(1) "1"
}
query_is_select('DELETE FROM test WHERE id = 1'): FALSE

Cache hit

```

```
    query_is_select('SELECT id FROM test WHERE id = 1'): 5
    get_hash(5)
    find_query_in_cache(1)
    return_to_cache(1)
    update_query_run_time_stats(3)
array(1) {
  ["id"]=>
  string(1) "1"
}

Display cache statistics
  get_stats(0)
array(4) {
  ["num_entries"]=>
  int(1)
  ["handler"]=>
  string(4) "user"
  ["handler_version"]=>
  string(5) "1.0.0"
  ["data"]=>
  array(1) {
    ["18683c177dc89bb352b29965d112fdaa"]=>
    array(4) {
      ["hits"]=>
      int(1)
      ["bytes"]=>
      int(71)
      ["uncached_run_time"]=>
      int(398)
      ["cached_run_time"]=>
      int(4)
    }
  }
}

Flushing cache, cache put/cache miss    clear_cache(0)
bool(true)
    query_is_select('SELECT id FROM test WHERE id = 1'): 5
    get_hash(5)
    find_query_in_cache(1)
    add_query_to_cache_if_not_exists(6)
NULL
```

22.9.7.5. Installing/Configuring

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

22.9.7.5.1. Requirements

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

[PHP 5.3.3](#) or a newer version of [PHP 5.3](#). [PHP 5.4.0](#) is not yet supported.

PECL/mysqlnd_qc is a mysqlnd plugin. It plugs into the mysqlnd library. To use you plugin with a PHP MySQL extension ([mysqli](#), [mysql](#), [PDO_MYSQL](#)) it is necessary to enable use of the mysqlnd library by the PHP MySQL extension in question. Please refer to the extensions manual sections for details.

For using [APC](#) storage handler: [APC 3.1.3p1-beta](#) or newer.

For using [MEMCACHE](#) storage handler: [libmemcache 0.38](#) or newer.

For using `sqlite` storage handler: `sqlite3` bundled with PHP.

22.9.7.5.2. Installation

Copyright 1997-2012 the PHP Documentation Group. [3186]

This [PECL](#) extension is not bundled with PHP.

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: http://pecl.php.net/package/mysqlnd_qc

A DLL for this PECL extension is currently unavailable. See also the [building on Windows](#) section.

22.9.7.5.3. Runtime Configuration

Copyright 1997-2012 the PHP Documentation Group. [3186]

The behaviour of these functions is affected by settings in `php.ini`.

Table 22.73. mysqlnd_qc Configure Options

| Name | Default | Changeable | Changelog |
|--|-----------------------------------|----------------|-----------|
| <code>mysqlnd_qc.enable_qc</code> | 1 | PHP_INI_SYSTEM | |
| <code>mysqlnd_qc.ttl</code> | 30 | PHP_INI_ALL | |
| <code>mysqlnd_qc.cache_by_default</code> | 0 | PHP_INI_ALL | |
| <code>mysqlnd_qc.cache_no_table</code> | 0 | PHP_INI_ALL | |
| <code>mysqlnd_qc.use_request_time</code> | 0 | PHP_INI_ALL | |
| <code>mysqlnd_qc.time_statistics</code> | 1 | PHP_INI_ALL | |
| <code>mysqlnd_qc.collect_statistics</code> | 0 | PHP_INI_ALL | |
| <code>mysqlnd_qc.collect_statistics_temp_file</code> | <code>tmp/mysqlnd_qc.stats</code> | PHP_INI_SYSTEM | |
| <code>mysqlnd_qc.collect_query_trace</code> | 0 | PHP_INI_SYSTEM | |
| <code>mysqlnd_qc.query_trace_buffer_depth</code> | 3 | PHP_INI_SYSTEM | |
| <code>mysqlnd_qc.collect_normalized_query_trace</code> | 0 | PHP_INI_SYSTEM | |
| <code>mysqlnd_qc.ignore_sql_comments</code> | 0 | PHP_INI_ALL | |
| <code>mysqlnd_qc.slam_defense</code> | 0 | PHP_INI_SYSTEM | |
| <code>mysqlnd_qc.slam_defense_timeout</code> | 30 | PHP_INI_SYSTEM | |
| <code>mysqlnd_qc.std_data_copy</code> | 0 | PHP_INI_SYSTEM | |
| <code>mysqlnd_qc.apc_prefix</code> | <code>qc_</code> | PHP_INI_ALL | |
| <code>mysqlnd_qc.memc_server</code> | 127.0.0.1 | PHP_INI_ALL | |
| <code>mysqlnd_qc.memc_port</code> | 11211 | PHP_INI_ALL | |
| <code>mysqlnd_qc.sqlite_data_file</code> | memory: | PHP_INI_ALL | |

Here's a short explanation of the configuration directives.

- `mysqlnd_qc.enable_qc` integer Enables or disables the plugin. If disabled the extension will not plug into `mysqlnd` to proxy internal `mysqlnd` C API calls.
- `mysqlnd_qc.ttl` integer Default Time-to-Live (TTL) for cache entries in seconds.

Mysqlnd query result cache plugin (`mysqlnd_qc`)

| | |
|--|---|
| <code>mysqlnd_qc.cache_by_default</code>
integer | Cache all queries regardless if they begin with the SQL hint that enables caching of a query or not. Storage handler cannot overrule the setting. It is evaluated by the core of the plugin. |
| <code>mysqlnd_qc.cache_no_table</code>
integer | Whether to cache queries with no table name in any of columns meta data of their result set, for example, <code>SELECT SLEEP(1)</code> , <code>SELECT NOW()</code> , <code>SELECT SUBSTRING()</code> . |
| <code>mysqlnd_qc.use_request_time</code>
integer | Use PHP global request time to avoid <code>gettimeofday()</code> system calls? If using <code>APC</code> storage handler it should be set to the value of <code>apc.use_request_time</code> , if not warnings will be generated. |
| <code>mysqlnd_qc.time_statistics</code>
integer | Collect run time and store time statistics using <code>gettimeofday()</code> system call? Data will be collected only if you also set <code>mysqlnd_qc.collect_statistics = 1</code> , |
| <code>mysqlnd_qc.collect_statistics</code>
integer | Collect statistics for <code>mysqlnd_qc_get_core_stats</code> ? Does not influence storage handler statistics! Handler statistics can be an integral part of the handler internal storage format. Therefore, collection of some handler statistics cannot be disabled. |
| <code>mysqlnd_qc.collect_statistics_log-file</code>
integer | If <code>mysqlnd_qc.collect_statistics</code> and <code>mysqlnd_qc.collect_statistics_log_file</code> are set, the plugin will dump statistics into the specified log file at every 10th web request during PHP request shutdown. The log file needs to be writable by the web server user.

Since 1.1.0. |
| <code>mysqlnd_qc.collect_query_trace</code>
integer | Collect query back traces? |
| <code>mysqlnd_qc.query_trace_bt_max</code>
integer | Maximum depth/level of a query code backtrace. |
| <code>mysqlnd_qc.ignore_sql_comments</code>
integer | Whether to remove SQL comments from a query string before hashing it to generate a cache key. Disable if you do not want two statemts such as <code>SELECT /*my_source_ip=123*/ id FROM test</code> and <code>SELECT /*my_source_ip=456*/ id FROM test</code> to refer to the same cache entry.

Since 1.1.0. |
| <code>mysqlnd_qc.slam_defense</code>
integer | Activates handler based slam defense (cache stampeding protection) if available. Supported by <code>Default</code> and <code>APC</code> storage handler |
| <code>mysqlnd_qc.slam_defense_ttl</code>
integer | TTL for stale cache entries which are served while another client updates the entries. Supported by <code>APC</code> storage handler. |
| <code>mysqlnd_qc.collect_normalized</code>
integer | Collect aggregated normalized query traces? The setting has no effect by default. You compile the extension using the define <code>NORM_QUERY_TRACE_LOG</code> to make use of the setting. |
| <code>mysqlnd_qc.std_data_copy</code>
integer | Default storage handler: copy cached wire data? EXPERIMENTAL – use default setting! |
| <code>mysqlnd_qc.apc_prefix</code>
string | The <code>APC</code> storage handler stores data in the <code>APC</code> user cache. The setting sets a prefix to be used for cache entries. |

| | |
|--|--|
| <code>mysqlnd_qc.memc_server</code>
string | MEMCACHE storage handler: memcache server host. |
| <code>mysqlnd_qc.memc_port</code>
integer | MEMCACHE storage handler: memcached server port. |
| <code>mysqlnd_qc.sqlite_data_file</code>
string | sqlite storage handler: data file. Any setting but <code>:memory:</code> may be of little practical value. |

22.9.7.6. Predefined Constants

Copyright 1997-2012 the PHP Documentation Group. [3186]

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

SQL hint related

Example 22.306. Using SQL hint constants

The query cache is controlled by SQL hints. SQL hints are used to enable and disable caching. SQL hints can be used to set the `TTL` of a query.

The SQL hints recognized by the query cache can be manually changed at compile time. This makes it possible to use `mysqlnd_qc` in environments in which the default SQL hints are already taken and interpreted by other systems. Therefore it is recommended to use the SQL hint string constants instead of manually adding the default SQL hints to the query string.

```
<?php
/* Use constants for maximum portability */
$query = "/*" . MYSQLND_QC_ENABLE_SWITCH . "*/SELECT id FROM test";

/* Valid but less portable: default TTL */
$query = "/*qc=on*/SELECT id FROM test";

/* Valid but less portable: per statement TTL */
$query = "/*qc=on*//*qc_ttl=5*/SELECT id FROM test";

printf("MYSQLND_QC_ENABLE_SWITCH: %s\n", MYSQLND_QC_ENABLE_SWITCH);
printf("MYSQLND_QC_DISABLE_SWITCH: %s\n", MYSQLND_QC_DISABLE_SWITCH);
printf("MYSQLND_QC_TTL_SWITCH: %s\n", MYSQLND_QC_TTL_SWITCH);
?>
```

The above examples will output:

```
MYSQLND_QC_ENABLE_SWITCH: qc=on
MYSQLND_QC_DISABLE_SWITCH: qc=off
MYSQLND_QC_TTL_SWITCH: qc_ttl=
```

`MYSQLND_QC_ENABLE_SWITCH` SQL hint used to enable caching of a query.
(string)

`MYSQLND_QC_DISABLE_SWITCH` SQL hint used to disable caching of a query if
(string) `mysqlnd_qc.cache_by_default = 1`.

[MYSQLND_QC_TTL_SWITCH](#) SQL hint used to set the TTL of a result set.
(string)

[MYSQLND_QC_SERVER_ID_SWITCH](#) This SQL hint should not be used in general.
(string)

It is needed by [PECL/mysqlnd_ms](#) to group cache entries for one statement but originating from different physical connections. If the hint is used connection settings such as user, hostname and charset are not considered for generating a cache key of a query. Instead the given value and the query string are used as input to the hashing function that generates the key.

[PECL/mysqlnd_ms](#) may, if instructed, cache results from MySQL Replication slaves. Because it can hold many connections to the slave the cache key shall not be formed from the user, hostname or other settings that may vary for the various slave connections. Instead, [PECL/mysqlnd_ms](#) provides an identifier which refers to the group of slave connections that shall be enabled to share cache entries no matter which physical slave connection was to generate the cache entry.

Use of this feature outside of [PECL/mysqlnd_ms](#) is not recommended.

[mysqlnd_qc_set_cache_condition](#) related

Example 22.307. Example [mysqlnd_qc_set_cache_condition](#) usage

The function [mysqlnd_qc_set_cache_condition](#) allows setting conditions for automatic caching of statements which don't begin with the SQL hints necessary to manually enable caching.

```
<?php
/* Cache all accesses to tables with the name "new%" in schema/database "db_example" for 1 second */
if (!mysqlnd_qc_set_cache_condition(MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN, "db_example.new%", 1)) {
    die("Failed to set cache condition!");
}

$mysqli = new mysqli("host", "user", "password", "db_example", "port");
/* cached although no SQL hint given */
$mysqli->query("SELECT id, title FROM news");

$pdo_mysql = new PDO("mysql:host=host;dbname=db_example;port=port", "user", "password");
/* not cached: no SQL hint, no pattern match */
$pdo_mysql->query("SELECT id, title FROM latest_news");
/* cached: TTL 1 second, pattern match */
$pdo_mysql->query("SELECT id, title FROM news");
?>
```

[MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN](#) Used as a parameter of [mysqlnd_qc_set_cache_condition](#) to set conditions for schema based automatic caching.
(int)

22.9.7.7. [mysqlnd_qc](#) Functions

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.7.7.1. [mysqlnd_qc_clear_cache](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_qc_clear_cache](#)

Flush all cache contents

Description

```
bool mysqlnd_qc_clear_cache();
```

Flush all cache contents.

Flushing the cache is a storage handler responsibility. All built-in storage handler but the `memcache` storage handler support flushing the cache. The `memcache` storage handler cannot flush its cache contents.

User-defined storage handler may or may not support the operation.

Parameters

This function has no parameters.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

A return value of `FALSE` indicates that flushing all cache contents has failed or the operation is not supported by the active storage handler. Applications must not expect that calling the function will always flush the cache.

22.9.7.7.2. `mysqlnd_qc_get_available_handlers`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_qc_get_available_handlers`

Returns a list of available storage handler

Description

```
array mysqlnd_qc_get_available_handlers();
```

Which storage are available depends on the compile time configuration of the query cache plugin. The `default` storage handler is always available. All other storage handler must be enabled explicitly when building the extension.

Parameters

This function has no parameters.

Return Values

Returns an array of available built-in storage handler. For each storage handler the version number and version string is given.

Examples

Example 22.308. `mysqlnd_qc_get_available_handlers` example

```
<?php
var_dump(mysqlnd_qc_get_available_handlers());
?>
```

The above examples will output:

```
array(5) {
  ["default"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
  ["user"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
  ["APC"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
  ["MEMCACHE"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
  ["sqlite"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
}
```

See Also

[Installation](#)

[mysqlnd_qc_set_storage_handler](#)

22.9.7.7.3. [mysqlnd_qc_get_cache_info](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_qc_get_cache_info](#)

Returns information on the current handler, the number of cache entries and cache entries, if available

Description

```
array mysqlnd_qc_get_cache_info();
```

Parameters

This function has no parameters.

Return Values

Returns information on the current handler, the number of cache entries and cache entries, if available. If and what data will be returned for the cache entries is subject to the active storage handler. Storage handler are free to return any data. Storage handler are recommended to return at least the data provided by the default handler, if technically possible.

The scope of the information is the PHP process. Depending on the PHP deployment model a process may serve one or more web requests.

Values are aggregated for all cache activities on a per storage handler basis. It is not possible to tell how much queries originating from `mysqli`, `PDO_MySQL` or `mysql`.API calls have contributed to the aggregated data values. Use `mysqlnd_qc_get_core_stats` to get timing data aggregated for all storage handlers.

Array of cache information

| | |
|-------------------------------------|---|
| <code>handler</code> string | <p>The active storage handler.</p> <p>All storage handler. Since 1.0.0.</p> |
| <code>handler_version</code> string | <p>The version of the active storage handler.</p> <p>All storage handler. Since 1.0.0.</p> |
| <code>num_entries</code> int | <p>The number of cache entries. The value depends on the storage handler in use.</p> <p>The default, APC and SQLite storage handler provide the actual number of cache entries.</p> <p>The MEMCACHE storage handler always returns 0. MEMCACHE does not support counting the number of cache entries.</p> <p>If a user defined handler is used, the number of entries of the <code>data</code> property is reported.</p> <p>Since 1.0.0.</p> |
| <code>data</code> array | <p>The version of the active storage handler.</p> <p>Additional storage handler dependent data on the cache entries. Storage handler are requested to provide similar and comparable information. A user defined storage handler is free to return any data.</p> <p>Since 1.0.0.</p> <p>The following information is provided by the default storage handler for the <code>data</code> property.</p> <p>The <code>data</code> property holds a hash. The hash is indexed by the internal cache entry identifier of the storage handler. The cache entry identifier is human-readable and contains the query string leading to the cache entry. Please, see also the example below. The following data is given for every cache entry.</p> |
| <code>statistics</code> array | <p>Statistics of the cache entry.</p> <p>Since 1.0.0.</p> |

| Property | Description | Version |
|-----------------------------|---|--------------|
| <code>rows</code> | Number of rows of the cached result set. | Since 1.0.0. |
| <code>store_size</code> | The size of the cached result set in bytes. This is the size of the payload. The value is not suited for calculating the total memory consumption of all cache entries including the administrative overhead of the cache entries. | Since 1.0.0. |
| <code>cache_hits</code> | How often the cached entry has been returned. | Since 1.0.0. |
| <code>run_time</code> | Run time of the statement to which the cache entry belongs. This is the run time of the uncached statement. It is the time between sending the statement to MySQL receiving a reply from MySQL. Run time saved by using the query cache plugin can be calculated like this: <code>cache_hits * ((run_time - avg_run_time) + (store_time - avg_store_time))</code> . | Since 1.0.0. |
| <code>store_time</code> | Store time of the statements result set to which the cache entry belongs. This is the time it took to fetch and store the results of the uncached statement. | Since 1.0.0. |
| <code>min_run_time</code> | Minimum run time of the cached statement. How long it took to find the statement in the cache. | Since 1.0.0. |
| <code>min_store_time</code> | Minimum store time of the cached statement. The time taken for fetching the cached result set from the storage medium and decoding | Since 1.0.0. |

| Property | Description | Version |
|------------------------|---|--------------|
| <code>avg_run</code> | Average run time of the cached statement. | Since 1.0.0. |
| <code>avg_store</code> | Average store time of the cached statement. | Since 1.0.0. |
| <code>max_run</code> | Average run time of the cached statement. | Since 1.0.0. |
| <code>max_store</code> | Average store time of the cached statement. | Since 1.0.0. |
| <code>validity</code> | Timestamp when the cache entry expires. | Since 1.1.0. |

`metadata` array

Metadata of the cache entry. This is the metadata provided by MySQL together with the result set of the statement in question. Different versions of the MySQL server may return different metadata. Unlike with some of the PHP MySQL extensions no attempt is made to hide MySQL server version dependencies and version details from the caller. Please, refer to the MySQL C API documentation that belongs to the MySQL server in use for further details.

The metadata list contains one entry for every column.

Since 1.0.0.

| Property | Description | Version |
|-------------------------|---|--------------|
| <code>name</code> | The field name. Depending on the MySQL version this may be the fields alias name. | Since 1.0.0. |
| <code>org_name</code> | The field name. | Since 1.0.0. |
| <code>table</code> | The table name. If an alias name was used for the table, this usually holds the alias name. | Since 1.0.0. |
| <code>org_table</code> | The table name. | Since 1.0.0. |
| <code>db</code> | The database/schema name. | Since 1.0.0. |
| <code>max_length</code> | The maximum width of the field. Details may | Since 1.0.0. |

| Property | Description | Version |
|---------------------|--|--------------|
| | vary by MySQL server version. | |
| <code>length</code> | The width of the field. Details may vary by MySQL server version. | Since 1.0.0. |
| <code>type</code> | The data type of the field. Details may vary by the MySQL server in use. This is the MySQL C API type constants value. It is recommended to use type constants provided by the <code>mysqli</code> extension to test for its meaning. You should not test for certain type values by comparing with certain numbers. | Since 1.0.0. |

The APC storage handler returns the same information for the `data` property but no `metadata`. The `metadata` of a cache entry is set to `NULL`.

The MEMCACHE storage handler does not fill the `data` property. Statistics are not available on a per cache entry basis with the MEMCACHE storage handler.

A user defined storage handler is free to provide any data.

Examples

Example 22.309. `mysqlnd_qc_get_cache_info` example

The example shows the output from the built-in default storage handler. Other storage handler may report different data.

```
<?php
/* Populate the cache, e.g. using mysqli */
$mysqli = new mysqli("host", "user", "password", "schema");
$mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/SELECT id FROM test");

/* Display cache information */
var_dump(mysqlnd_qc_get_cache_info());
?>
```

The above examples will output:

```
array(4) {
  ["num_entries"]=>
  int(1)
  ["handler"]=>
  string(7) "default"
  ["handler_version"]=>
```

```
string(5) "1.0.0"
["data"]=>
array(1) {
  ["Localhost via UNIX socket 3306 user schema/*qc=on*/SELECT id FROM test"]=>
  array(2) {
    ["statistics"]=>
    array(11) {
      ["rows"]=>
      int(6)
      ["stored_size"]=>
      int(101)
      ["cache_hits"]=>
      int(0)
      ["run_time"]=>
      int(471)
      ["store_time"]=>
      int(27)
      ["min_run_time"]=>
      int(0)
      ["max_run_time"]=>
      int(0)
      ["min_store_time"]=>
      int(0)
      ["max_store_time"]=>
      int(0)
      ["avg_run_time"]=>
      int(0)
      ["avg_store_time"]=>
      int(0)
    }
  }
  ["metadata"]=>
  array(1) {
    [0]=>
    array(8) {
      ["name"]=>
      string(2) "id"
      ["orig_name"]=>
      string(2) "id"
      ["table"]=>
      string(4) "test"
      ["orig_table"]=>
      string(4) "test"
      ["db"]=>
      string(4) "schema"
      ["max_length"]=>
      int(1)
      ["length"]=>
      int(11)
      ["type"]=>
      int(3)
    }
  }
}
```

See Also

[mysqlnd_qc_get_core_stats](#)

22.9.7.7.4. [mysqlnd_qc_get_core_stats](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_qc_get_core_stats`

Statistics collected by the core of the query cache

Description

```
array mysqlnd_qc_get_core_stats();
```

Returns an array of statistics collected by the core of the cache plugin. The same data fields will be reported for any storage handler because the data is collected by the core.

The [PHP](#) configuration setting `mysqlnd_qc.collect_statistics` controls the collection of statistics. The collection of statistics is disabled by default for performance reasons. Disabling the collection of statistics will also disable the collection of time related statistics.

The [PHP](#) configuration setting `mysqlnd_qc.collect_time_statistics` controls the collection of time related statistics.

The scope of the core statistics is the [PHP](#) process. Depending on your deployment model a [PHP](#) process may handle one or multiple requests.

Statistics are aggregated for all cache entries and all storage handler. It is not possible to tell how much queries originating from `mysqli`, `PDO_MySQL` or `mysql` API calls have contributed to the aggregated data values.

Parameters

This function has no parameters.

Return Values

Array of core statistics

| Statistic | Description | Version |
|-------------------------|--|--------------|
| <code>cache_hit</code> | Statement is considered cacheable and cached data has been reused. Statement is considered cacheable and a cache miss happened but the statement got cached by someone else while we process it and thus we can fetch the result from the refreshed cache. | Since 1.0.0. |
| <code>cache_miss</code> | Statement is considered cacheable... <ul style="list-style-type: none"> • ... and has been added to the cache • ... but the PHP configuration directive setting of <code>mysqlnd_qc.cache_no_table = 1</code> has prevented caching. • ... but an unbuffered result set is requested. | Since 1.0.0. |

| Statistic | Description | Version |
|-------------------------------------|---|--------------|
| | <ul style="list-style-type: none"> ... but a buffered result set was empty. | |
| <code>cache_put</code> | Statement is considered cacheable and has been added to the cache. Take care when calculating derived statistics. Storage handler with a storage life time beyond process scope may report <code>cache_put = 0</code> together with <code>cache_hit > 0</code> , if another process has filled the cache. You may want to use <code>num_entries</code> from <code>mysqlnd_qc_get_cache_info</code> if the handler supports it (<code>default</code> , <code>APC</code>). | Since 1.0.0. |
| <code>query_should_cache</code> | Statement is considered cacheable based on query string analysis. The statement may or may not be added to the cache. See also <code>cache_put</code> . | Since 1.0.0. |
| <code>query_should_not_cache</code> | Statement is considered not cacheable based on query string analysis. | Since 1.0.0. |
| <code>query_not_cached</code> | Statement is considered not cacheable or it is considered cacheable but the storage handler has not returned a hash key for it. | Since 1.0.0. |
| <code>query_could_cache</code> | Statement is considered cacheable... <ul style="list-style-type: none"> ... and statement has been run without errors ... and meta data shows at least one column in the result set <p>The statement may or may not be in the cache already. It may or may not be added to the cache later on.</p> | Since 1.0.0. |
| <code>query_found_in_cache</code> | Statement is considered cacheable and we have found it in the cache but we have not replayed the cached data yet and we have not send the result set to the client yet. This is not considered a cache hit because the client might not fetch the result or the cached data may be faulty. | Since 1.0.0. |

| Statistic | Description | Version |
|--|--|--------------|
| <code>query_uncached_other</code> | Statement is considered cacheable and it may or may not be in the cache already but either replaying cached data has failed, no result set is available or some other error has happened. | |
| <code>query_uncached_no_table</code> | Statement has not been cached because the result set has at least one column which has no table name in its meta data. An example of such a query is <code>SELECT SLEEP(1)</code> . To cache those statements you have to change default value of the PHP configuration directive <code>mysqLnd_qc.cache_no_table</code> and set <code>mysqLnd_qc.cache_no_table = 1</code> . Often, it is not desired to cache such statements. | Since 1.0.0. |
| <code>query_uncached_use_result</code> | Statement would have been cached if a buffered result set had been used. The situation is also considered as a cache miss and <code>cache_miss</code> will be incremented as well. | Since 1.0.0. |
| <code>query_aggr_run_time_cache</code> | Aggregated run time (ms) of all cached queries. Cached queries are those which have incremented <code>cache_hit</code> . | Since 1.0.0. |
| <code>query_aggr_run_time_cache</code> | Aggregated run time (ms) of all uncached queries that have been put into the cache. See also <code>cache_put</code> . | Since 1.0.0. |
| <code>query_aggr_run_time_total</code> | Aggregated run time (ms) of all uncached and cached queries that have been inspected and executed by the query cache. | Since 1.0.0. |
| <code>query_aggr_store_time_cache</code> | Aggregated store time (ms) of all cached queries. Cached queries are those which have incremented <code>cache_hit</code> . | Since 1.0.0. |
| <code>query_aggr_store_time_cache</code> | Aggregated store time (ms) of all uncached queries that have been put into the cache. See also <code>cache_put</code> . | Since 1.0.0. |
| <code>query_aggr_store_time_total</code> | Aggregated store time (ms) of all uncached and cached queries | Since 1.0.0. |

| Statistic | Description | Version |
|-------------------------------------|---|--------------|
| | that have been inspected and executed by the query cache. | |
| <code>receive_bytes_recorded</code> | Recorded incoming network traffic (<code>bytes</code>) send from MySQL to PHP. The traffic may or may not have been added to the cache. The traffic is the total for all queries regardless if cached or not. | Since 1.0.0. |
| <code>receive_bytes_replayed</code> | Network traffic replayed during cache. This is the total amount of incoming traffic saved because of the usage of the query cache plugin. | Since 1.0.0. |
| <code>send_bytes_recorded</code> | Recorded outgoing network traffic (<code>bytes</code>) send from MySQL to PHP. The traffic may or may not have been added to the cache. The traffic is the total for all queries regardless if cached or not. | Since 1.0.0. |
| <code>send_bytes_replayed</code> | Network traffic replayed during cache. This is the total amount of outgoing traffic saved because of the usage of the query cache plugin. | Since 1.0.0. |
| <code>slam_stale_refresh</code> | Number of cache misses which triggered serving stale data until the client causing the cache miss has refreshed the cache entry. | Since 1.0.0. |
| <code>slam_stale_hit</code> | Number of cache hits while a stale cache entry gets refreshed. | Since 1.0.0. |

Examples

Example 22.310. `mysqlnd_qc_get_core_stats` example

```
<?php
/* Enable collection of statistics - default: disabled */
ini_set("mysqlnd_qc.collect_statistics", 1);

/* Enable collection of all timing related statistics -
default: enabled but overruled by mysqlnd_qc.collect_statistics = 0 */
ini_set("mysqlnd_qc.collect_time_statistics", 1);

/* Populate the cache, e.g. using mysqli */
$mysqli = new mysqli('host', 'user', 'password', 'schema');

/* Cache miss and cache put */
$mysqli->query("/*qc=on*/SELECT id FROM test");
/* Cache hit */
$mysqli->query("/*qc=on*/SELECT id FROM test");

/* Display core statistics */
```

```
var_dump(mysqlnd_qc_get_core_stats());
?>
```

The above examples will output:

```
array(26) {
  ["cache_hit"]=>
  string(1) "1"
  ["cache_miss"]=>
  string(1) "1"
  ["cache_put"]=>
  string(1) "1"
  ["query_should_cache"]=>
  string(1) "2"
  ["query_should_not_cache"]=>
  string(1) "0"
  ["query_not_cached"]=>
  string(1) "0"
  ["query_could_cache"]=>
  string(1) "2"
  ["query_found_in_cache"]=>
  string(1) "1"
  ["query_uncached_other"]=>
  string(1) "0"
  ["query_uncached_no_table"]=>
  string(1) "0"
  ["query_uncached_no_result"]=>
  string(1) "0"
  ["query_uncached_use_result"]=>
  string(1) "0"
  ["query_aggr_run_time_cache_hit"]=>
  string(1) "4"
  ["query_aggr_run_time_cache_put"]=>
  string(3) "395"
  ["query_aggr_run_time_total"]=>
  string(3) "399"
  ["query_aggr_store_time_cache_hit"]=>
  string(1) "2"
  ["query_aggr_store_time_cache_put"]=>
  string(1) "8"
  ["query_aggr_store_time_total"]=>
  string(2) "10"
  ["receive_bytes_recorded"]=>
  string(2) "65"
  ["receive_bytes_replayed"]=>
  string(2) "65"
  ["send_bytes_recorded"]=>
  string(2) "29"
  ["send_bytes_replayed"]=>
  string(2) "29"
  ["slam_stale_refresh"]=>
  string(1) "0"
  ["slam_stale_hit"]=>
  string(1) "0"
  ["request_counter"]=>
  int(1)
  ["process_hash"]=>
  int(3547549858)
}
```

See Also

[Runtime configuration](#)
[mysqlnd_qc.collect_statistics](#)
[mysqlnd_qc.time_statistics](#)
[mysqlnd_qc_get_cache_info](#)

22.9.7.7.5. `mysqlnd_qc_get_normalized_query_trace_log`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_qc_get_normalized_query_trace_log`

Returns a normalized query trace log for each query inspected by the query cache

Description

```
array mysqlnd_qc_get_normalized_query_trace_log();
```

Returns a normalized query trace log for each query inspected by the query cache. The collection of the trace log is disabled by default. To collect the trace log you have to set the PHP configuration directive `mysqlnd_qc.collect_normalized_query_trace` to 1

Entries in the trace log are grouped by the normalized query statement. The normalized query statement is the query statement with all statement parameter values being replaced with a question mark. For example, the two statements `SELECT id FROM test WHERE id = 1` and `SELECT id FROM test WHERE id = 2` are normalized as `SELECT id FROM test WHERE id = ?`. Whenever a statement is inspected by the query cache which matches the normalized statement pattern, its statistics are grouped by the normalized statement string.

Parameters

This function has no parameters.

Return Values

An array of query log. Every list entry contains the normalized query string and further detail information.

| Key | Description |
|---------------------------------|---|
| <code>query</code> | Normalized statement string. |
| <code>occurrence</code> | How many statements have matched the normalized statement string in addition to the one which has created the log entry. The value is zero if a statement has been normalized, its normalized representation has been added to the log but no further queries inspected by PECL/mysqlnd_qc have the same normalized statement string. |
| <code>eligible_for_cache</code> | Whether the statement could be cached. An statement eligible for caching has not necessarily been cached. It not possible to tell for sure if or how many cached statement have contributed to the aggregated normalized statement log entry. However, comparing the minimum and average run time one can make an educated guess. |
| <code>avg_run_time</code> | The average run time of all queries contributing to the query log entry. The run time is the time between sending the query statement to MySQL and receiving an answer from MySQL. |
| <code>avg_store_time</code> | The average store time of all queries contributing to the query log entry. The store time is the time needed to fetch a statements result set from the server to the client and, storing it on the client. |
| <code>min_run_time</code> | The minimum run time of all queries contributing to the query log entry. |
| <code>min_store_time</code> | The minimum store time of all queries contributing to the query log entry. |
| <code>max_run_time</code> | The maximum run time of all queries contributing to the query log entry. |

| Key | Description |
|---------------------------------|--|
| <code>max_store_time_max</code> | The maximum store time of all queries contributing to the query log entry. |

Examples

Example 22.311. `mysqlnd_qc_get_normalized_query_trace_log` example

```
mysqlnd_qc.collect_normalized_query_trace=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");

/* not cached */
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();

/* cache put */
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 2");
var_dump($res->fetch_assoc());
$res->free();

/* cache hit */
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 2");
var_dump($res->fetch_assoc());
$res->free();

var_dump(mysqlnd_qc_get_normalized_query_trace_log());
?>
```

The above examples will output:

```
array(1) {
  ["id"]=>
  string(1) "1"
}
array(1) {
  ["id"]=>
  string(1) "2"
}
array(1) {
  ["id"]=>
  string(1) "2"
}
array(4) {
  [0]=>
  array(9) {
    ["query"]=>
    string(25) "DROP TABLE IF EXISTS test"
    ["occurrences"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["avg_run_time"]=>
    int(0)
  }
}
```

```
    ["min_run_time"]=>
    int(0)
    ["max_run_time"]=>
    int(0)
    ["avg_store_time"]=>
    int(0)
    ["min_store_time"]=>
    int(0)
    ["max_store_time"]=>
    int(0)
  }
[1]=>
array(9) {
    ["query"]=>
    string(27) "CREATE TABLE test (id INT )"
    ["occurrences"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["avg_run_time"]=>
    int(0)
    ["min_run_time"]=>
    int(0)
    ["max_run_time"]=>
    int(0)
    ["avg_store_time"]=>
    int(0)
    ["min_store_time"]=>
    int(0)
    ["max_store_time"]=>
    int(0)
  }
[2]=>
array(9) {
    ["query"]=>
    string(40) "INSERT INTO test (id ) VALUES ( ? ), ( ? )"
    ["occurrences"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["avg_run_time"]=>
    int(0)
    ["min_run_time"]=>
    int(0)
    ["max_run_time"]=>
    int(0)
    ["avg_store_time"]=>
    int(0)
    ["min_store_time"]=>
    int(0)
    ["max_store_time"]=>
    int(0)
  }
[3]=>
array(9) {
    ["query"]=>
    string(31) "SELECT id FROM test WHERE id =?"
    ["occurrences"]=>
    int(2)
    ["eligible_for_caching"]=>
    bool(true)
    ["avg_run_time"]=>
    int(159)
    ["min_run_time"]=>
    int(12)
    ["max_run_time"]=>
    int(307)
```

```
[ "avg_store_time" ]=>
int(10)
[ "min_store_time" ]=>
int(8)
[ "max_store_time" ]=>
int(13)
}
}
```

See Also

- [Runtime configuration](#)
- [mysqlnd_qc.collect_normalized_query_trace](#)
- [mysqlnd_qc.time_statistics](#)
- [mysqlnd_qc_get_query_trace_log](#)

22.9.7.7.6. mysqlnd_qc_get_query_trace_log

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_qc_get_query_trace_log](#)

Returns a backtrace for each query inspected by the query cache

Description

```
array mysqlnd_qc_get_query_trace_log();
```

Returns a backtrace for each query inspected by the query cache. The collection of the backtrace is disabled by default. To collect the backtrace you have to set the PHP configuration directive `mysqlnd_qc.collect_query_trace` to 1

The maximum depth of the backtrace is limited to the depth set with the PHP configuration directive `mysqlnd_qc.query_trace_bt_depth`.

Parameters

This function has no parameters.

Return Values

An array of query backtrace. Every list entry contains the query string, a backtrace and further detail information.

| Key | Description |
|-------------------------|---|
| <code>query</code> | Query string. |
| <code>origin</code> | Code backtrace. |
| <code>run_time</code> | Query run time in milliseconds. The collection of all times and the necessary <code>gettimeofday</code> system calls can be disabled by setting the PHP configuration directive <code>mysqlnd_qc.time_statistics</code> to 0 |
| <code>store_time</code> | Query result set store time in milliseconds. The collection of all times and the necessary <code>gettimeofday</code> system calls can be disabled by setting the PHP configuration directive <code>mysqlnd_qc.time_statistics</code> to 0 |
| <code>eligible</code> | <code>TRUE</code> if query is cacheable otherwise <code>FALSE</code> . |
| <code>no_table</code> | <code>TRUE</code> if the query has generated a result set and at least one column from the result set has no table name set in its metadata. This is usually the case with queries which one probably do not |

| Key | Description |
|-----------------------------------|--|
| | want to cache such as <code>SELECT SLEEP(1)</code> . By default any such query will not be added to the cache. See also PHP configuration directive <code>mysqlnd_qc.cache_no_table</code> . |
| <code>was_added</code> | <code>TRUE</code> if the query result has been put into the cache, otherwise <code>FALSE</code> . |
| <code>was_already_in_cache</code> | <code>TRUE</code> if the query result would have been added to the cache if it was not already in the cache (cache hit). Otherwise <code>FALSE</code> . |

Examples

Example 22.312. `mysqlnd_qc_get_query_trace_log` example

```
mysqlnd_qc.collect_query_trace=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");

/* not cached */
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();

/* cache put */
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 2");
var_dump($res->fetch_assoc());
$res->free();

/* cache hit */
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 2");
var_dump($res->fetch_assoc());
$res->free();

var_dump(mysqlnd_qc_get_query_trace_log());
?>
```

The above examples will output:

```
array(1) {
  ["id"]=>
  string(1) "1"
}
array(1) {
  ["id"]=>
  string(1) "2"
}
array(1) {
  ["id"]=>
  string(1) "2"
}
array(6) {
  [0]=>
  array(8) {
    ["query"]=>
    string(25) "DROP TABLE IF EXISTS test"
```

```
["origin"]=>
string(102) "#0 qc.php(4): mysqli->query('DROP TABLE IF E...')
#1 {main}"
["run_time"]=>
int(0)
["store_time"]=>
int(0)
["eligible_for_caching"]=>
bool(false)
["no_table"]=>
bool(false)
["was_added"]=>
bool(false)
["was_already_in_cache"]=>
bool(false)
}
[1]=>
array(8) {
["query"]=>
string(25) "CREATE TABLE test(id INT)"
["origin"]=>
string(102) "#0 qc.php(5): mysqli->query('CREATE TABLE te...')
#1 {main}"
["run_time"]=>
int(0)
["store_time"]=>
int(0)
["eligible_for_caching"]=>
bool(false)
["no_table"]=>
bool(false)
["was_added"]=>
bool(false)
["was_already_in_cache"]=>
bool(false)
}
[2]=>
array(8) {
["query"]=>
string(36) "INSERT INTO test(id) VALUES (1), (2)"
["origin"]=>
string(102) "#0 qc.php(6): mysqli->query('INSERT INTO tes...')
#1 {main}"
["run_time"]=>
int(0)
["store_time"]=>
int(0)
["eligible_for_caching"]=>
bool(false)
["no_table"]=>
bool(false)
["was_added"]=>
bool(false)
["was_already_in_cache"]=>
bool(false)
}
[3]=>
array(8) {
["query"]=>
string(32) "SELECT id FROM test WHERE id = 1"
["origin"]=>
string(102) "#0 qc.php(9): mysqli->query('SELECT id FROM ...')
#1 {main}"
["run_time"]=>
int(0)
["store_time"]=>
int(25)
```

```

    ["eligible_for_caching"]=>
    bool(false)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
    bool(false)
  }
[4]=>
array(8) {
    ["query"]=>
    string(41) "/*qc=on*/SELECT id FROM test WHERE id = 2"
    ["origin"]=>
    string(103) "#0 qc.php(14): mysqli->query('/*qc=on*/SELECT...')
#1 {main}"
    ["run_time"]=>
    int(311)
    ["store_time"]=>
    int(13)
    ["eligible_for_caching"]=>
    bool(true)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(true)
    ["was_already_in_cache"]=>
    bool(false)
  }
[5]=>
array(8) {
    ["query"]=>
    string(41) "/*qc=on*/SELECT id FROM test WHERE id = 2"
    ["origin"]=>
    string(103) "#0 qc.php(19): mysqli->query('/*qc=on*/SELECT...')
#1 {main}"
    ["run_time"]=>
    int(13)
    ["store_time"]=>
    int(8)
    ["eligible_for_caching"]=>
    bool(true)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
    bool(true)
  }
}

```

See Also

[Runtime configuration](#)
[mysqlnd_qc.collect_query_trace](#)
[mysqlnd_qc.query_trace_bt_depth](#)
[mysqlnd_qc.time_statistics](#)
[mysqlnd_qc.cache_no_table](#)
[mysqlnd_qc_get_normalized_query_trace_log](#)

22.9.7.7.7. `mysqlnd_qc_set_cache_condition`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_qc_set_cache_condition`

Set conditions for automatic caching

Description

```
bool mysqlnd_qc_set_cache_condition(
    int condition_type,
    mixed condition,
    mixed condition_option);
```

Sets a condition for automatic caching of statements which do not contain the necessary SQL hints to enable caching of them.

Parameters

`condition_type`

Type of the condition. The only allowed value is `MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN`.

`condition`

Parameter for the condition set with `condition_type`. Parameter type and structure depend on `condition_type`

If `condition_type` equals `MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN` `condition` must be a string. The string sets a pattern. Statements are cached if table and database meta data entry of their result sets match the pattern. The pattern is checked for a match with the `db` and `org_table` meta data entries provided by the underlying MySQL client server library. Please, check the MySQL Reference manual for details about the two entries. The `db` and `org_table` values are concatenated with a dot (`.`) before matched against `condition`. Pattern matching supports the wildcards `%` and `_`. The wildcard `%` will match one or many arbitrary characters. `_` will match one arbitrary character. The escape symbol is backslash.

`condition_option`

Option for `condition`. Type and structure depend on `condition_type`.

If `condition_type` equals `MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN` `condition_options` is the TTL to be used.

Examples

Example 22.313. `mysqlnd_qc_set_cache_condition` example

```
<?php
/* Cache all accesses to tables with the name "new%" in schema/database "db_example" for 1 second */
if (!mysqlnd_qc_set_cache_condition(MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN, "db_example.new%", 1)) {
    die("Failed to set cache condition!");
}

$mysqli = new mysqli("host", "user", "password", "db_example", "port");
/* cached although no SQL hint given */
$mysqli->query("SELECT id, title FROM news");

$pdo_mysql = new PDO("mysql:host=host;dbname=db_example;port=port", "user", "password");
/* not cached: no SQL hint, no pattern match */
$pdo_mysql->query("SELECT id, title FROM latest_news");
```

```
/* cached: TTL 1 second, pattern match */
$pdo_mysql->query("SELECT id, title FROM news");
?>
```

Return Values

Returns TRUE on success or FALSE on FAILURE.

See Also

[Quickstart: pattern based caching](#)

22.9.7.7.8. `mysqlnd_qc_set_is_select`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_qc_set_is_select`

Installs a callback which decides whether a statement is cached

Description

```
mixed mysqlnd_qc_set_is_select(
    string callback);
```

Installs a callback which decides whether a statement is cached.

There are several ways of hinting PECL/mysqlnd_qc to cache a query. By default, PECL/mysqlnd_qc attempts to cache a if caching of all statements is enabled or the query string begins with a certain SQL hint. The plugin internally calls a function named `is_select()` to find out. This internal function can be replaced with a user-defined callback. Then, the user-defined callback is responsible to decide whether the plugin attempts to cache a statement. Because the internal function is replaced with the callback, the callback gains full control. The callback is free to ignore the configuration setting `mysqlnd_qc.cache_by_default` and SQL hints.

The callback is invoked for every statement inspected by the plugin. It is given the statements string as a parameter. The callback returns `FALSE` if the statement shall not be cached. It returns `TRUE` to make the plugin attempt to cache the statements result set, if any. A so-created cache entry is given the default TTL set with the PHP configuration directive `mysqlnd_qc.ttl`. If a different TTL shall be used, the callback returns a numeric value to be used as the TTL.

The internal `is_select` function is part of the internal cache storage handler interface. Thus, a user-defined storage handler offers the same capabilities.

Parameters

This function has no parameters.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 22.314. `mysqlnd_qc_set_is_select` example

```
<?php
```

```

/* callback which decides if query is cached */
function is_select($query) {
    static $patterns = array(
        /* true - use default from mysqlnd_qc.ttl */
        "@SELECT\s+.*\s+FROM\s+test@ismU" => true,
        /* 3 - use TTL = 3 seconds */
        "@SELECT\s+.*\s+FROM\s+news@ismU" => 3
    );
    /* check if query does match pattern */
    foreach ($patterns as $pattern => $ttl) {
        if (preg_match($pattern, $query)) {
            printf("is_select(%45s): cache\n", $query);
            return $ttl;
        }
    }
    printf("is_select(%45s): do not cache\n", $query);
    return false;
}
mysqlnd_qc_set_is_select("is_select");

/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");

/* cache put */
$mysqli->query("SELECT id FROM test WHERE id = 1");
/* cache hit */
$mysqli->query("SELECT id FROM test WHERE id = 1");
/* cache put */
$mysqli->query("SELECT * FROM test");
?>

```

The above examples will output:

```

is_select(                DROP TABLE IF EXISTS test): do not cache
is_select(                CREATE TABLE test(id INT)): do not cache
is_select(  INSERT INTO test(id) VALUES (1), (2), (3)): do not cache
is_select(                SELECT id FROM test WHERE id = 1): cache
is_select(                SELECT id FROM test WHERE id = 1): cache
is_select(                SELECT * FROM test): cache

```

See Also

[Runtime configuration](#)
[mysqlnd_qc.ttl](#)
[mysqlnd_qc.cache_by_default](#)
[mysqlnd_qc_set_user_handlers](#)

22.9.7.7.9. [mysqlnd_qc_set_storage_handler](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_qc_set_storage_handler](#)

Change current storage handler

Description

```
bool mysqlnd_qc_set_storage_handler(
    string handler);
```

Sets the storage handler used by the query cache. A list of available storage handler can be obtained from [mysqlnd_qc_get_available_handlers](#). Which storage are available depends on the compile time configuration of the query cache plugin. The `default` storage handler is always available. All other storage handler must be enabled explicitly when building the extension.

Parameters

handler Handler can be of type string representing the name of a built-in storage handler or an object of type `mysqlnd_qc_handler_default`. The names of the built-in storage handler are `default`, `APC`, `MEMCACHE`, `sqlite`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

If changing the storage handler fails a catchable fatal error will be thrown. The query cache cannot operate if the previous storage handler has been shutdown but no new storage handler has been installed.

Examples

Example 22.315. [mysqlnd_qc_set_storage_handler](#) example

The example shows the output from the built-in default storage handler. Other storage handler may report different data.

```
<?php
var_dump(mysqlnd_qc_set_storage_handler("memcache"));

if (true === mysqlnd_qc_set_storage_handler("default"))
    printf("Default storage handler activated");

/* Catchable fatal error */
var_dump(mysqlnd_qc_set_storage_handler("unknown"));
?>
```

The above examples will output:

```
bool(true)
Default storage handler activated
Catchable fatal error: mysqlnd_qc_set_storage_handler(): Unknown handler 'unknown' in (file) on line (line
```

See Also

[Installation](#)

[mysqlnd_qc_get_available_handlers](#)

22.9.7.7.10. [mysqlnd_qc_set_user_handlers](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_qc_set_user_handlers](#)

Sets the callback functions for a user-defined procedural storage handler

Description

```
bool mysqlnd_qc_set_user_handlers(  
    string get_hash,  
    string find_query_in_cache,  
    string return_to_cache,  
    string add_query_to_cache_if_not_exists,  
    string query_is_select,  
    string update_query_run_time_stats,  
    string get_stats,  
    string clear_cache);
```

Sets the callback functions for a user-defined procedural storage handler.

Parameters

| | |
|--|--|
| get_hash | Name of the user function implementing the storage handler get_hash functionality. |
| find_query_in_cache | Name of the user function implementing the storage handler find_in_cache functionality. |
| return_to_cache | Name of the user function implementing the storage handler return_to_cache functionality. |
| add_query_to_cache_if_not_exists | Name of the user function implementing the storage handler add_query_to_cache_if_not_exists functionality. |
| query_is_select | Name of the user function implementing the storage handler query_is_select functionality. |
| update_query_run_time_stats | Name of the user function implementing the storage handler update_query_run_time_stats functionality. |
| get_stats | Name of the user function implementing the storage handler get_stats functionality. |
| clear_cache | Name of the user function implementing the storage handler clear_cache functionality. |

Return Values

Returns TRUE on success or FALSE on FAILURE.

See Also

[Procedural user-defined storage handler example](#)

22.9.7.8. Change History

Copyright 1997-2012 the PHP Documentation Group. [3186]

This change history is a high level summary of selected changes that may impact applications and/or break backwards compatibility.

See also the [CHANGES](#) file in the source distribution for a complete list of changes.

22.9.7.8.1. PECL/mysqlnd_qc 1.1 series

Copyright 1997-2012 the PHP Documentation Group. [3186]

1.1.0

- Release date: under development
- Motto/theme: PHP 5.4 compatibility, schema pattern based caching and mysqlnd_ms support

Feature changes

- APC storage handler update
 - Fix build for APC 3.1.9+
 - Note: Use of the APC storage handler is currently not recommended due to stability issues of APC itself.
- New PHP configuration directives
 - [mysqlnd_qc.collect_statistics_log_file](#). Aggregated cache statistics log file written after every 10th request served by the PHP process.
 - [mysqlnd_qc.ignore_sql_comments](#). Control whether SQL comments are ignored for cache key hash generation.
- New constants and SQL hints
 - [MYSQLND_QC_SERVER_ID_SWITCH](#) allows grouping of cache entries from different physical connections. This is needed by PECL/mysqlnd_ms.
 - [MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN](#) to be used with [mysqlnd_qc_set_cache_condition](#).
- New function [mysqlnd_qc_set_cache_condition](#) for built-in schema pattern based caching. Likely to support a wider range of conditions in the future.
- Report [valid_until](#) timestamp for cache entries of the default handler through [mysqlnd_qc_get_cache_info](#).
- Include charset number for cache entry hashing. This should prevent serving result sets which have the wrong charset.
API change: [get_hash_key](#) expects new "charsetnr" (int) parameter after "port".
- API change: changing [is_select\(\)](#) signature from `bool is_select()` to mixed `is_select()`. Mixed can be either boolean or array(long ttl, string server_id). This is needed by PECL/mysqlnd_ms.

Other

- Support acting as a cache backend for [PECL/mysqlnd_ms](#) 1.3.0-beta or later to transparently replace MySQL Replication slave reads with cache accesses, if the user explicitly allows.

Bug fixes

- Fixed Bug #59959 (config.m4, wrong library - 64bit memcached handler builds) (Credits: Remi Collet)

22.9.7.8.2. PECL/mysqlnd_qc 1.0 series

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

1.0.1-stable

- Release date: 12/2010
- Motto/theme: Prepared statement support

Added support for Prepared statements and unbuffered queries.

1.0.0-beta

- Release date: 07/2010
- Motto/theme: TTL-based cache with various storage options (Memcache, APC, SQLite, user-defined)

Initial public release of the transparent TTL-based query result cache. Flexible storage of cached results. Various storage media supported.

22.9.8. Mysqlnd user handler plugin ([mysqlnd_uh](#))

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The mysqlnd user handler plugin ([mysqlnd_uh](#)) allows users to set hooks for most internal calls of the MySQL native driver for PHP ([mysqlnd](#)). Mysqlnd and its plugins, including PECL/mysqlnd_uh, operate on a layer beneath the PHP MySQL extensions. A mysqlnd plugin can be considered as a proxy between the PHP MySQL extensions and the MySQL server as part of the PHP executable on the client-side. Because the plugins operates on their own layer below the PHP MySQL extensions, they can monitor and change application actions without requiring application changes. If the PHP MySQL extensions ([mysqli](#), [mysql](#), [PDO_MYSQL](#)) are compiled to use [mysqlnd](#) this can be used for:

- Monitoring
 - Queries executed by any of the PHP MySQL extensions
 - Prepared statements executing by any of the PHP MySQL extensions
- Auditing
 - Detection of database usage
 - SQL injection protection using black and white lists
- Assorted
 - Load Balancing connections

The MySQL native driver for PHP ([mysqlnd](#)) features an internal plugin C API. C plugins, such as the mysqlnd user handler plugin, can extend the functionality of [mysqlnd](#). PECL/mysqlnd_uh makes parts of the internal plugin C API available to the PHP user for plugin development with PHP.



Status

The mysqlnd user handler plugin is in alpha status. Take appropriate care before using it in production environments.

22.9.8.1. Security considerations

Copyright 1997-2012 the PHP Documentation Group. [3186]

PECL/`mysqlnd_uh` gives users access to MySQL user names, MySQL password used by any of the PHP MySQL extensions to connect to MySQL. It allows monitoring of all queries and prepared statements exposing the statement string to the user. Therefore, the extension should be installed with care. The `PHP_INI_SYSTEM` configuration setting `mysqlnd_uh.enable` can be used to prevent users from hooking `mysqlnd` calls.

Code obfuscators and similar technologies are not suitable to prevent monitoring of `mysqlnd` library activities if PECL/`mysqlnd_uh` is made available and the user can install a proxy, for example, using `auto_prepend_file`.

22.9.8.2. Documentation note

Copyright 1997-2012 the PHP Documentation Group. [3186]

Many of the `mysqlnd_uh` functions are briefly described because the `mysqli` extension is a thin abstraction layer on top of the MySQL C API that the `mysqlnd` library provides. Therefore, the corresponding `mysqli` documentation (along with the MySQL reference manual) can be consulted to receive more information about a particular function.

22.9.8.3. On the name

Copyright 1997-2012 the PHP Documentation Group. [3186]

The shortcut `mysqlnd_uh` stands for `mysqlnd user handler`, and has been the name since early development.

22.9.8.4. Quickstart and Examples

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqlnd` user handler plugin can be understood as a client-side proxy for all PHP MySQL extensions (`mysqli`, `mysql`, `PDO_MYSQL`), if they are compiled to use the `mysqlnd` library. The extensions use the `mysqlnd` library internally, at the C level, to communicate with the MySQL server. PECL/`mysqlnd_uh` allows it to hook many `mysqlnd` calls. Therefore, most activities of the PHP MySQL extensions can be monitored.

Because monitoring happens at the level of the library, at a layer below the application, it is possible to monitor applications without changing them.

On the C level, the `mysqlnd` library is structured in modules or classes. The extension hooks almost all methods of the `mysqlnd` internal `connection` class and exposes them through the user space class `MysqlndUhConnection`. Some few methods of the `mysqlnd` internal `statement` class are made available to the PHP user with the class `MysqlndUhPreparedStatement`. By subclassing the classes `MysqlndUhConnection` and `MysqlndUhPreparedStatement` users get access to `mysqlnd` internal function calls.



Note

The internal `mysqlnd` function calls are not designed to be exposed to the PHP user. Manipulating their activities may cause PHP to crash or leak memory. Often, this is not considered a bug. Please, keep in mind that you are accessing C library functions through PHP which are expected to take certain actions, which you may not be able to emulate in user space. Therefore, it is strongly

recommended to always call the parent method implementation when subclassing `MysqlndUhConnection` or `MysqlndUhPreparedStatement`. To prevent the worst case, the extension performs some sanity checks. Please, see also the [Mysqlnd_uh Configure Options](#).

22.9.8.4.1. Setup

Copyright 1997-2012 the PHP Documentation Group. [3186]

The plugin is implemented as a PHP extension. See the [installation instructions](#) to install the [PECL/mysqlnd_uh](#) extension. Then, load the extension into PHP and activate the plugin in the PHP configuration file using the PHP configuration directive named `mysqlnd_uh.enable`. The below example shows the default settings of the extension.

Example 22.316. Enabling the plugin (`php.ini`)

```
mysqlnd_uh.enable=1
mysqlnd_uh.report_wrong_types=1
```

22.9.8.4.2. How it works

Copyright 1997-2012 the PHP Documentation Group. [3186]

This describes the background and inner workings of the `mysqlnd_uh` extension.

Two classes are provided by the extension: `MysqlndUhConnection` and `MysqlndUhPreparedStatement`. `MysqlndUhConnection` lets you access almost all methods of the `mysqlnd` internal `connection` class. The latter exposes some selected methods of the `mysqlnd` internal `statement` class. For example, `MysqlndUhConnection::connect` maps to the `mysqlnd` library C function `mysqlnd_conn__connect`.

As a `mysqlnd` plugin, the `PECL/mysqlnd_uh` extension replaces `mysqlnd` library C functions with its own functions. Whenever a PHP MySQL extension compiled to use `mysqlnd` calls a `mysqlnd` function, the functions installed by the plugin are executed instead of the original `mysqlnd` ones. For example, `mysqli_connect` invokes `mysqlnd_conn__connect`, so the connect function installed by `PECL/mysqlnd_uh` will be called. The functions installed by `PECL/mysqlnd_uh` are the methods of the built-in classes.

The built-in PHP classes and their methods do nothing but call their `mysqlnd` C library counterparts, to behave exactly like the original `mysqlnd` function they replace. The code below illustrates in pseudo-code what the extension does.

Example 22.317. Pseudo-code: what a built-in class does

```
class MysqlndUhConnection {
    public function connect(($conn, $host, $user, $passwd, $db, $port, $socket, $mysql_flags) {
        MYSQLND* c_mysqlnd_connection = convert_from_php_to_c($conn);
        ...
        return call_c_function(mysqlnd_conn__connect(c_mysqlnd_connection, ...));
    }
}
```

The built-in classes behave like a transparent proxy. It is possible for you to replace the proxy with your own. This is done by subclassing `MysqlndUhConnection` or `MysqlndUhPreparedStatement` to

extend the functionality of the proxy, followed by registering a new proxy object. Proxy objects are installed by `mysqlnd_uh_set_connection_proxy` and `mysqlnd_uh_set_statement_proxy`.

Example 22.318. Installing a proxy

```
<?php
class proxy extends MysqlndUhConnection {
    public function connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The above example will output:

```
proxy::connect(array (
    0 => NULL,
    1 => 'localhost',
    2 => 'root',
    3 => '',
    4 => 'test',
    5 => 3306,
    6 => NULL,
    7 => 131072,
))
proxy::connect returns true
```

22.9.8.4.3. Installing a proxy

Copyright 1997-2012 the PHP Documentation Group. [3186]

The extension provides two built-in classes: `MysqlndUhConnection` and `MysqlndUhPreparedStatement`. The classes are used for hooking `mysqlnd` library calls. Their methods correspond to `mysqlnd` internal functions. By default they act like a transparent proxy and do nothing but call their `mysqlnd` counterparts. By subclassing the classes you can install your own proxy to monitor `mysqlnd`.

See also the [How it works](#) guide to learn about the inner workings of this extension.

Connection proxies are objects of the type `MysqlndUhConnection`. Connection proxy objects are installed by `mysqlnd_uh_set_connection_proxy`. If you install the built-in class `MysqlndUhConnection` as a proxy, nothing happens. It behaves like a transparent proxy.

Example 22.319. Proxy registration, `mysqlnd_uh.enable=1`

```
<?php
mysqlnd_uh_set_connection_proxy(new MysqlndUhConnection());
$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The `PHP_INI_SYSTEM` configuration setting `mysqlnd_uh.enable` controls whether a proxy may be set. If disabled, the extension will throw errors of type `E_WARNING`

Example 22.320. Proxy installation disabled

```
mysqlnd_uh.enable=0
```

```
<?php
mysqlnd_uh_set_connection_proxy(new MySQLndUhConnection());
$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The above example will output:

```
PHP Warning: MySQLndUhConnection::__construct(): (MySQLnd User Handler) The plugin has been disabled by settin
PHP Warning: mysqlnd_uh_set_connection_proxy(): (MySQLnd User Handler) The plugin has been disabled by settin
```

To monitor `mysqlnd`, you have to write your own proxy object subclassing `MySQLndUhConnection`. Please, see the function reference for a the list of methods that can be subclassed. Alternatively, you can use reflection to inspect the built-in `MySQLndUhConnection`.

Create a new class `proxy`. Derive it from the built-in class `MySQLndUhConnection`. Replace the `MySQLndUhConnection::connect` method. Print out the host parameter value passed to the method. Make sure that you call the parent implementation of the `connect` method. Failing to do so may give unexpected and undesired results, including memory leaks and crashes.

Register your proxy and open three connections using the PHP MySQL extensions `mysqli`, `mysql`, `PDO_MYSQL`. If the extensions have been compiled to use the `mysqlnd` library, the `proxy::connect` method will be called three times, once for each connection opened.

Example 22.321. Connection proxy

```
<?php
class proxy extends MySQLndUhConnection {
    public function connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags) {
        printf("Connection opened to '%s'\n", $host);
        /* Always call the parent implementation! */
        return parent::connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags);
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysql = mysql_connect("localhost", "root", "");
$pdo = new PDO("mysql:host=localhost;dbname=test", "root", "");
?>
```

The above example will output:

```
Connection opened to 'localhost'
Connection opened to 'localhost'
```

```
Connection opened to 'localhost'
```

The use of prepared statement proxies follows the same pattern: create a proxy object of the type [MysqlndUhPreparedStatement](#) and install the proxy using [mysqlnd_uh_set_statement_proxy](#).

Example 22.322. Prepared statement proxy

```
<?php
class stmt_proxy extends MysqlndUhPreparedStatement {
    public function prepare($res, $query) {
        printf("%s(%s)\n", __METHOD__, $query);
        return parent::prepare($res, $query);
    }
}
mysqlnd_uh_set_statement_proxy(new stmt_proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$stmt = $mysqli->prepare("SELECT 'mysqlnd hacking made easy' AS _msg FROM DUAL");
?>
```

The above example will output:

```
stmt_proxy::prepare(SELECT 'mysqlnd hacking made easy' AS _msg FROM DUAL)
```

22.9.8.4.4. Basic query monitoring

Copyright 1997-2012 the PHP Documentation Group. [3186]

Basic monitoring of a query statement is easy with PECL/mysqlnd_uh. Combined with [debug_print_backtrace](#) it can become a powerful tool, for example, to find the origin of certain statement. This may be desired when searching for slow queries but also after database refactoring to find code still accessing deprecated databases or tables. The latter may be a complicated matter to do otherwise, especially if the application uses auto-generated queries.

Example 22.323. Basic Monitoring

```
<?php
class conn_proxy extends MysqlndUhConnection {
    public function query($res, $query) {
        debug_print_backtrace();
        return parent::query($res, $query);
    }
}
class stmt_proxy extends MysqlndUhPreparedStatement {
    public function prepare($res, $query) {
        debug_print_backtrace();
        return parent::prepare($res, $query);
    }
}
mysqlnd_uh_set_connection_proxy(new conn_proxy());
mysqlnd_uh_set_statement_proxy(new stmt_proxy());

printf("Proxies installed...\n");
$pdo = new PDO("mysql:host=localhost;dbname=test", "root", "");
var_dump($pdo->query("SELECT 1 AS _one FROM DUAL")->fetchAll(PDO::FETCH_ASSOC));
```

```
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->prepare("SELECT 1 AS _two FROM DUAL");
?>
```

The above example will output:

```
#0 conn_proxy->query(Resource id #19, SELECT 1 AS _one FROM DUAL)
#1 PDO->query(SELECT 1 AS _one FROM DUAL) called at [example.php:19]
array(1) {
  [0]=>
  array(1) {
    ["_one"]=>
    string(1) "1"
  }
}
#0 stmt_proxy->prepare(Resource id #753, SELECT 1 AS _two FROM DUAL)
#1 mysqli->prepare(SELECT 1 AS _two FROM DUAL) called at [example.php:22]
```

For basic query monitoring you should install a connection and a prepared statement proxy. The connection proxy should subclass [MysqlndUhConnection::query](#). All database queries not using native prepared statements will call this method. In the example the [query](#) function is invoked by a PDO call. By default, [PDO_MYSQL](#) is using prepared statement emulation.

All native prepared statements are prepared with the [prepare](#) method of [mysqlnd](#) exported through [MysqlndUhPreparedStatement::prepare](#). Subclass [MysqlndUhPreparedStatement](#) and overwrite [prepare](#) for native prepared statement monitoring.

22.9.8.5. Installing/Configuring

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.8.5.1. Requirements

Copyright 1997-2012 the PHP Documentation Group. [3186]

[PHP 5.3.3](#) or later. It is recommended to use [PHP 5.4.0](#) or later to get access to the latest [mysqlnd](#) features.

The [mysqlnd_uh](#) user handler plugin supports all PHP applications and all available PHP MySQL extensions ([mysqli](#), [mysql](#), [PDO_MYSQL](#)). The PHP MySQL extension must be configured to use [mysqlnd](#) in order to be able to use the [mysqlnd_uh](#) plugin for [mysqlnd](#).

The alpha versions makes use of some [mysqli](#) features. You must enable [mysqli](#) to compile the plugin. This requirement may be removed in the future. Note, that this requirement does not restrict you to use the plugin only with [mysqli](#). You can use the plugin to monitor [mysql](#), [mysqli](#) and [PDO_MYSQL](#).

22.9.8.5.2. Installation

Copyright 1997-2012 the PHP Documentation Group. [3186]

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: <http://pecl.php.net/package/mysqlnd-uh>

PECL/[mysqlnd_uh](#) is currently not available on Windows. The source code of the extension makes use of [C99](#) constructs not allowed with PHP Windows builds.

22.9.8.5.3. Runtime Configuration

Copyright 1997-2012 the PHP Documentation Group. [3186]

The behaviour of these functions is affected by settings in `php.ini`.

Table 22.74. MySQLnd_uh Configure Options

| Name | Default | Changeable | Changelog |
|--|---------|----------------|-----------|
| <code>mysqlnd_uh.enable</code> | 1 | PHP_INI_SYSTEM | |
| <code>mysqlnd_uh.report_wrong_types</code> | 1 | PHP_INI_ALL | |

Here's a short explanation of the configuration directives.

`mysqlnd_uh.enable` integer Enables or disables the plugin. If set to disabled, the extension will not allow users to plug into `mysqlnd` to hook `mysqlnd` calls.

`mysqlnd_uh.report_wrong_types` integer Whether to report wrong return value types of user hooks as `E_WARNING` level errors. This is recommended for detecting errors.

22.9.8.5.4. Resource Types

Copyright 1997-2012 the PHP Documentation Group. [3186]

This extension has no resource types defined.

22.9.8.6. Predefined Constants

Copyright 1997-2012 the PHP Documentation Group. [3186]

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Most of the constants refer to details of the MySQL Client Server Protocol. Please, refer to the MySQL reference manual to learn about their meaning. To avoid content duplication, only short descriptions are given.

`MySQLndUhConnection::simpleCommand` related

The following constants can be used to detect what command is to be send through `MySQLndUhConnection::simpleCommand`.

`MYSQLND_UH_MYSQLND_COM_SLEEP` MySQL Client Server protocol command: `COM_SLEEP`.
(integer)

`MYSQLND_UH_MYSQLND_COM_QUIT` MySQL Client Server protocol command: `COM_QUIT`.
(integer)

`MYSQLND_UH_MYSQLND_COM_INIT_DB` MySQL Client Server protocol command: `COM_INIT_DB`.
(integer)

`MYSQLND_UH_MYSQLND_COM_QUERY` MySQL Client Server protocol command: `COM_QUERY`.
(integer)

`MYSQLND_UH_MYSQLND_COM_FIELD_LIST` MySQL Client Server protocol command: `COM_FIELD_LIST`.
(integer)

`MYSQLND_UH_MYSQLND_COM_CREATE_DB` MySQL Client Server protocol command: `COM_CREATE_DB`.
(integer)

`MYSQLND_UH_MYSQLND_COM_DROP_DB` MySQL Client Server protocol command: `COM_DROP_DB`.
(integer)

`MYSQLND_UH_MYSQLND_COM_REFRESH` MySQL Client Server protocol command: `COM_REFRESH`.
(integer)

`MYSQLND_UH_MYSQLND_COM_SHUTDOWN` MySQL Client Server protocol command: `COM_SHUTDOWN`.
(integer)

`MYSQLND_UH_MYSQLND_COM_STATISTICS` MySQL Client Server protocol command: `COM_STATISTICS`.
(integer)

`MYSQLND_UH_MYSQLND_COM_PROCESS_INFO` MySQL Client Server protocol command: `COM_PROCESS_INFO`.
(integer)

`MYSQLND_UH_MYSQLND_COM_CONNECT` MySQL Client Server protocol command: `COM_CONNECT`.
(integer)

`MYSQLND_UH_MYSQLND_COM_PROCESS_KILL` MySQL Client Server protocol command: `COM_PROCESS_KILL`.
(integer)

`MYSQLND_UH_MYSQLND_COM_DEBUG` MySQL Client Server protocol command: `COM_DEBUG`.
(integer)

`MYSQLND_UH_MYSQLND_COM_PING` MySQL Client Server protocol command: `COM_PING`.
(integer)

`MYSQLND_UH_MYSQLND_COM_TIME` MySQL Client Server protocol command: `COM_TIME`.
(integer)

`MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT` MySQL Client Server protocol command: `COM_DELAYED_INSERT`.
(integer)

`MYSQLND_UH_MYSQLND_COM_CHANGE_USER` MySQL Client Server protocol command: `COM_CHANGE_USER`.
(integer)

`MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP` MySQL Client Server protocol command: `COM_BINLOG_DUMP`.
(integer)

`MYSQLND_UH_MYSQLND_COM_TABLE_DUMP` MySQL Client Server protocol command: `COM_TABLE_DUMP`.
(integer)

`MYSQLND_UH_MYSQLND_COM_CONNECT_OUT` MySQL Client Server protocol command: `COM_CONNECT_OUT`.
(integer)

`MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVE` MySQL Client Server protocol command: `COM_REGISTER_SLAVE`.
(integer)

`MYSQLND_UH_MYSQLND_COM_STMT_PREPARE` MySQL Client Server protocol command: `COM_STMT_PREPARE`.
(integer)

`MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE` MySQL Client Server protocol command: `COM_STMT_EXECUTE`.
(integer)

`MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA` MySQL Client Server protocol command:
(integer) `COM_STMT_SEND_LONG_DATA`.

`MYSQLND_UH_MYSQLND_COM_STMT_CLOSE` MySQL Client Server protocol command: COM_STMT_CLOSE.
(integer)

`MYSQLND_UH_MYSQLND_COM_STMT_RESET` MySQL Client Server protocol command: COM_STMT_RESET.
(integer)

`MYSQLND_UH_MYSQLND_COM_SET_OPTION` MySQL Client Server protocol command: COM_SET_OPTION.
(integer)

`MYSQLND_UH_MYSQLND_COM_STMT_FETCH` MySQL Client Server protocol command: COM_STMT_FETCH.
(integer)

`MYSQLND_UH_MYSQLND_COM_DAEMON` MySQL Client Server protocol command: COM_DAEMON.
(integer)

`MYSQLND_UH_MYSQLND_COM_END` MySQL Client Server protocol command: COM_END.
(integer)

The following constants can be used to analyze the `ok_packet` argument of `MySQLndUhConnection::simpleCommand`.

`MYSQLND_UH_MYSQLND_PROT_GREETING` MySQL Client Server protocol packet: greeting.
(integer)

`MYSQLND_UH_MYSQLND_PROT_AUTHENTICATION` MySQL Client Server protocol packet: authentication.
(integer)

`MYSQLND_UH_MYSQLND_PROT_OK` MySQL Client Server protocol packet: OK.
(integer)

`MYSQLND_UH_MYSQLND_PROT_EOF` MySQL Client Server protocol packet: EOF.
(integer)

`MYSQLND_UH_MYSQLND_PROT_COMMAND` MySQL Client Server protocol packet: command.
(integer)

`MYSQLND_UH_MYSQLND_PROT_RESULTSET_HEADER` MySQL Client Server protocol packet: result set header.
(integer)

`MYSQLND_UH_MYSQLND_PROT_RESULTSET_FIELD` MySQL Client Server protocol packet: resultset field.
(integer)

`MYSQLND_UH_MYSQLND_PROT_ROW` MySQL Client Server protocol packet: row.
(integer)

`MYSQLND_UH_MYSQLND_PROT_STATS` MySQL Client Server protocol packet: stats.
(integer)

`MYSQLND_UH_MYSQLND_PREPARE_RESPONSE` MySQL Client Server protocol packet: prepare response.
(integer)

`MYSQLND_UH_MYSQLND_CHG_USER_RESPONSE` MySQL Client Server protocol packet: change user response.
(integer)

`MYSQLND_UH_MYSQLND_PROT_LAST` No practical meaning. Last entry marker of internal C data structure list.
(integer)

`MySQLndUhConnection::close` *related*

The following constants can be used to detect why a connection has been closed through `MySqlndUhConnection::close()`.

`MYSQLND_UH_MYSQLND_CLOSE_USER` User has called `mysqlnd` to close the connection.
(integer)

`MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT` Implicitly closed, for example, during garbage connection.
(integer)

`MYSQLND_UH_MYSQLND_CLOSE_CONNECTION_ERROR` Connection error.
(integer)

`MYSQLND_UH_MYSQLND_CLOSE_INVALID` No practical meaning. Last entry marker of internal C data structure list.
(integer)

MySqlndUhConnection::setServerOption() related

The following constants can be used to detect which option is set through `MySqlndUhConnection::setServerOption()`.

`MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENT_SUPPORT` Option: enables multi statement support.
(integer)

`MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENT_SUPPORT_DISABLED` Option: disables multi statement support.
(integer)

MySqlndUhConnection::setClientOption related

The following constants can be used to detect which option is set through `MySqlndUhConnection::setClientOption`.

`MYSQLND_UH_MYSQLND_OPTION_CONNECTION_TIMEOUT` Option: connection timeout.
(integer)

`MYSQLND_UH_MYSQLND_OPTION_COMPRESSED_PROTOCOL` Option: whether the MySQL compressed protocol is to be used.
(integer)

`MYSQLND_UH_MYSQLND_OPTION_NAMED_PIPE` Option: named pipe to use for connection (Windows).
(integer)

`MYSQLND_UH_MYSQLND_OPTION_INIT_COMMAND` Option: init command to execute upon connect.
(integer)

`MYSQLND_UH_MYSQLND_READ_DEFAULT_FILE` Option: MySQL server default file to read upon connect.
(integer)

`MYSQLND_UH_MYSQLND_READ_DEFAULT_FILE_GROUP` Option: MySQL server default file group to read upon connect.
(integer)

`MYSQLND_UH_MYSQLND_SET_CHARSET_DIR` Option: charset description files directory.
(integer)

`MYSQLND_UH_MYSQLND_SET_CHARSET_NAME` Option: charset name.
(integer)

`MYSQLND_UH_MYSQLND_OPT_LOCAL_INFILE` Option: Whether to allow `LOAD DATA LOCAL INFILE` use.
(integer)

- `MYSQLND_UH_MYSQLND_OPT_PROTOCOL` Option: supported protocol version.
(integer)
- `MYSQLND_UH_MYSQLND_SHARED_MEMORY_BASE_NAME` Option: shared memory base name for shared memory connections.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_READ_TIMEOUT` Option: connection read timeout.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_WRITE_TIMEOUT` Option: connection write timeout.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_USE_UNBUFFERED_RESULT_SETS` Option: unbuffered result sets.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_USE_EMBEDDED_SERVER` Embedded server related.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_USE_EMBEDDED_SERVER_IN` Embedded server related.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_GUEST_CONNECTION` TODO
(integer)
- `MYSQLND_UH_MYSQLND_SET_CLIENT_IP` TODO
(integer)
- `MYSQLND_UH_MYSQLND_SECURE` TODO
(integer)
- `MYSQLND_UH_MYSQLND_REPORT_DATA_TRUNCATION` Option: Whether to report data truncation.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_RECONNECT` Option: Whether to reconnect automatically.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_SSL_FILE_SERVER_CERT` Option: TODO
(integer)
- `MYSQLND_UH_MYSQLND_OPT_NET_BUFFER_SIZE_FOR_COMMANDS` Option: mysqlnd network buffer size for commands.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_NET_BUFFER_SIZE_FOR_READING_FROM_SERVER` Option: mysqlnd network buffer size for reading from the server.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_SSL_KEY` Option: SSL key.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_SSL_CERTIFICATE` Option: SSL certificate.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_SSL_CA` Option: SSL CA.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_SSL_PATH_TO_CA` Option: Path to SSL CA.
(integer)
- `MYSQLND_UH_MYSQLND_OPT_SSL_CIPHER` Option: SSL cipher.
(integer)

`MYSQLND_UH_MYSQLND_OPT_SSL_CIPHER` Option: SSL passphrase.
(integer)

`MYSQLND_UH_SERVER_OPTION_PLUGIN_DIRECTORY` Option: server plugin directory.
(integer)

`MYSQLND_UH_SERVER_OPTION_DEFAULT_AUTHENTICATION_METHOD` Option: default authentication method.
(integer)

`MYSQLND_UH_SERVER_OPTION_SECURE_CLIENT_IP` **TODO**
(integer)

`MYSQLND_UH_MYSQLND_OPT_MAX_ALLOWED_PACKET_SIZE` Option: maximum allowed packet size. Available as of PHP 5.4.0.
(integer)

`MYSQLND_UH_MYSQLND_OPT_AUTOMATIC_RETRY` Option: **TODO**. Available as of PHP 5.4.0.
(integer)

`MYSQLND_UH_MYSQLND_OPT_INTEGER_FLOAT` Option: make mysqlnd return integer and float columns as long even when using the MySQL Client Server text protocol. Only available with a custom build of mysqlnd.
(integer)

Other

The plugins version number can be obtained using `MYSQLND_UH_VERSION` or `MYSQLND_UH_VERSION_ID . MYSQLND_UH_VERSION` is the string representation of the numerical version number `MYSQLND_UH_VERSION_ID` , which is an integer such as 10000. Developers can calculate the version number as follows.

| Version (part) | Example |
|------------------------------------|-----------------|
| Major*10000 | 1*10000 = 10000 |
| Minor*100 | 0*100 = 0 |
| Patch | 0 = 0 |
| <code>MYSQLND_UH_VERSION_ID</code> | 10000 |

`MYSQLND_UH_VERSION` (string) Plugin version string, for example, "1.0.0-alpha".

`MYSQLND_UH_VERSION_ID` (integer) Plugin version number, for example, 10000.

22.9.8.7. The MysqlndUhConnection class (`MysqlndUhConnection`)

Copyright 1997-2012 the PHP Documentation Group. [3186]

```
MysqlndUhConnection {
MysqlndUhConnection

    Methods

    public bool MysqlndUhConnection::changeUser(
        mysqlnd_connection connection,
        string user,
        string password,
        string database,
        bool silent,
```

```

    int passwd_len);

public string MySqlndUhConnection::charsetName(
    mysqlnd_connection connection);

public bool MySqlndUhConnection::close(
    mysqlnd_connection connection,
    int close_type);

public bool MySqlndUhConnection::connect(
    mysqlnd_connection connection,
    string host,
    string use",
    string password,
    string database,
    int port,
    string socket,
    int mysql_flags);

public MySqlndUhConnection::__construct();

public bool MySqlndUhConnection::endPSession(
    mysqlnd_connection connection);

public string MySqlndUhConnection::escapeString(
    mysqlnd_connection connection,
    string escape_string);

public int MySqlndUhConnection::getAffectedRows(
    mysqlnd_connection connection);

public int MySqlndUhConnection::getErrorNumber(
    mysqlnd_connection connection);

public string MySqlndUhConnection::getErrorString(
    mysqlnd_connection connection);

public int MySqlndUhConnection::getFieldCount(
    mysqlnd_connection connection);

public string MySqlndUhConnection::getHostInformation(
    mysqlnd_connection connection);

public int MySqlndUhConnection::getLastInsertId(
    mysqlnd_connection connection);

public void MySqlndUhConnection::getLastMessage(
    mysqlnd_connection connection);

public string MySqlndUhConnection::getProtocolInformation(
    mysqlnd_connection connection);

public string MySqlndUhConnection::getServerInformation(
    mysqlnd_connection connection);

public string MySqlndUhConnection::getServerStatistics(
    mysqlnd_connection connection);

public int MySqlndUhConnection::getServerVersion(
    mysqlnd_connection connection);

public string MySqlndUhConnection::getSqlstate(
    mysqlnd_connection connection);

public array MySqlndUhConnection::getStatistics(
    mysqlnd_connection connection);

public int MySqlndUhConnection::getThreadId(

```

```

mysqLnd_connection connection);

public int MysqLndUhConnection::getWarningCount(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::init(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::killConnection(
    mysqLnd_connection connection,
    int pid);

public array MysqLndUhConnection::listFields(
    mysqLnd_connection connection,
    string table,
    string achtung_wild);

public void MysqLndUhConnection::listMethod(
    mysqLnd_connection connection,
    string query,
    string achtung_wild,
    string parl);

public bool MysqLndUhConnection::moreResults(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::nextResult(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::ping(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::query(
    mysqLnd_connection connection,
    string query);

public bool MysqLndUhConnection::queryReadResultsetHeader(
    mysqLnd_connection connection,
    mysqLnd_statement mysqLnd_stmt);

public bool MysqLndUhConnection::reapQuery(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::refreshServer(
    mysqLnd_connection connection,
    int options);

public bool MysqLndUhConnection::restartPSession(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::selectDb(
    mysqLnd_connection connection,
    string database);

public bool MysqLndUhConnection::sendClose(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::sendQuery(
    mysqLnd_connection connection,
    string query);

public bool MysqLndUhConnection::serverDumpDebugInformation(
    mysqLnd_connection connection);

public bool MysqLndUhConnection::setAutocommit(
    mysqLnd_connection connection,
    int mode);

```

```

public bool MySQLndUhConnection::setCharset(
    mysqlnd_connection connection,
    string charset);

public bool MySQLndUhConnection::setClientOption(
    mysqlnd_connection connection,
    int option,
    int value);

public void MySQLndUhConnection::setServerOption(
    mysqlnd_connection connection,
    int option);

public void MySQLndUhConnection::shutdownServer(
    string MYSQLND_UH_RES_MYSQLND_NAME,
    string "level");

public bool MySQLndUhConnection::simpleCommand(
    mysqlnd_connection connection,
    int command,
    string arg,
    int ok_packet,
    bool silent,
    bool ignore_upsert_status);

public bool MySQLndUhConnection::simpleCommandHandleResponse(
    mysqlnd_connection connection,
    int ok_packet,
    bool silent,
    int command,
    bool ignore_upsert_status);

public bool MySQLndUhConnection::sslSet(
    mysqlnd_connection connection,
    string key,
    string cert,
    string ca,
    string capath,
    string cipher);

public resource MySQLndUhConnection::stmtInit(
    mysqlnd_connection connection);

public resource MySQLndUhConnection::storeResult(
    mysqlnd_connection connection);

public bool MySQLndUhConnection::txCommit(
    mysqlnd_connection connection);

public bool MySQLndUhConnection::txRollback(
    mysqlnd_connection connection);

public resource MySQLndUhConnection::useResult(
    mysqlnd_connection connection);
}

```

22.9.8.7.1. `MySQLndUhConnection::changeUser`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MySQLndUhConnection::changeUser`

Changes the user of the specified `mysqlnd` database connection

Description

```
public bool MysqlndUhConnection::changeUser(
    mysqlnd_connection connection,
    string user,
    string password,
    string database,
    bool silent,
    int passwd_len);
```

Changes the user of the specified mysqlnd database connection

Parameters

| | |
|-------------------------|---|
| <code>connection</code> | Mysqlnd connection handle. Do not modify! |
| <code>user</code> | The MySQL user name. |
| <code>password</code> | The MySQL password. |
| <code>database</code> | The MySQL database to change to. |
| <code>silent</code> | Controls if mysqlnd is allowed to emit errors or not. |
| <code>passwd_len</code> | Length of the MySQL password. |

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.324. `MysqlndUhConnection::changeUser` example

```
<?php
class proxy extends MysqlndUhConnection {
    /* Hook mysqlnd's connection::change_user call */
    public function changeUser($res, $user, $passwd, $db, $silent, $passwd_len) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::changeUser($res, $user, $passwd, $db, $silent, $passwd_len);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
/* Install proxy/hooks to be used with all future mysqlnd connection */
mysqlnd_uh_set_connection_proxy(new proxy());

/* Create mysqli connection which is using the mysqlnd library */
$mysqli = new mysqli("localhost", "root", "", "test");

/* Example of a user API call which triggers the hooked mysqlnd call */
var_dump($mysqli->change_user("root", "bar", "test"));
?>
```

The above example will output:

```
proxy::changeUser(array (
    0 => NULL,
    1 => 'root',
    2 => 'bar',
    3 => 'test',
    4 => false,
```

```

    5 => 3,
  ))
proxy::changeUser returns false
bool(false)

```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_change_user](#)

22.9.8.7.2. [MysqlndUhConnection::charsetName](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::charsetName](#)

Returns the default character set for the database connection

Description

```

public string MysqlndUhConnection::charsetName(
    mysqlnd_connection connection);

```

Returns the default character set for the database connection.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

The default character set.

Examples

Example 22.325. [MysqlndUhConnection::charsetName](#) example

```

<?php
class proxy extends MysqlndUhConnection {
    public function charsetName($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::charsetName($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump(mysqli_character_set_name($mysqli));
?>

```

The above example will output:

```

proxy::charsetName(array (
    0 => NULL,
))

```

```
proxy::charsetName returns 'latin1'  
string(6) "latin1"
```

See Also

`mysqlnd_uh_set_connection_proxy`
`mysqli_character_set_name`

22.9.8.7.3. `MysqlndUhConnection::close`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MysqlndUhConnection::close`

Closes a previously opened database connection

Description

```
public bool MysqlndUhConnection::close(  
    mysqlnd_connection connection,  
    int close_type);
```

Closes a previously opened database connection.



Note

Failing to call the parent implementation may cause memory leaks or crash PHP. This is not considered a bug. Please, keep in mind that the `mysqlnd` library functions have never been designed to be exposed to the user space.

Parameters

connection

The connection to be closed. Do not modify!

close_type

Why the connection is to be closed. The value of *close_type* is one of `MYSQLND_UH_MYSQLND_CLOSE_EXPLICIT`, `MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT`, `MYSQLND_UH_MYSQLND_CLOSE_DISCONNECTED` or `MYSQLND_UH_MYSQLND_CLOSE_LAST`. The latter should never be seen, unless the default behaviour of the `mysqlnd` library has been changed by a plugin.

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.326. `MysqlndUhConnection::close` example

```
<?php  
function close_type_to_string($close_type) {  
    $mapping = array(  
        MYSQLND_UH_MYSQLND_CLOSE_DISCONNECTED => "MYSQLND_UH_MYSQLND_CLOSE_DISCONNECTED",  
        MYSQLND_UH_MYSQLND_CLOSE_EXPLICIT => "MYSQLND_UH_MYSQLND_CLOSE_EXPLICIT",  
        MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT => "MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT",  
        MYSQLND_UH_MYSQLND_CLOSE_LAST => "MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT"  
    );  
}
```

```
return (isset($mapping[$close_type])) ? $mapping[$close_type] : 'unknown';
}

class proxy extends MysqlndUhConnection {
    public function close($res, $close_type) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("close_type = %s\n", close_type_to_string($close_type));
        /* WARNING: you must call the parent */
        $ret = parent::close($res, $close_type);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->close();
?>
```

The above example will output:

```
proxy::close(array (
    0 => NULL,
    1 => 0,
))
close_type = MYSQLND_UH_MYSQLND_CLOSE_EXPLICIT
proxy::close returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_close](#)
[mysql_close](#)

22.9.8.7.4. `MysqlndUhConnection::connect`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MysqlndUhConnection::connect`

Open a new connection to the MySQL server

Description

```
public bool MysqlndUhConnection::connect(
    mysqlnd_connection connection,
    string host,
    string use",
    string password,
    string database,
    int port,
    string socket,
    int mysql_flags);
```

Open a new connection to the MySQL server.

Parameters

`connection` Mysqlnd connection handle. Do not modify!

| | |
|--------------------------|---|
| <code>host</code> | Can be either a host name or an IP address. Passing the NULL value or the string "localhost" to this parameter, the local host is assumed. When possible, pipes will be used instead of the TCP/IP protocol. |
| <code>user</code> | The MySQL user name. |
| <code>password</code> | If not provided or <code>NULL</code> , the MySQL server will attempt to authenticate the user against those user records which have no password only. This allows one username to be used with different permissions (depending on if a password as provided or not). |
| <code>database</code> | If provided will specify the default database to be used when performing queries. |
| <code>port</code> | Specifies the port number to attempt to connect to the MySQL server. |
| <code>socket</code> | Specifies the socket or named pipe that should be used. If <code>NULL</code> , mysqlnd will default to <code>/tmp/mysql.sock</code> . |
| <code>mysql_flags</code> | Connection options. |

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.327. `MysqlndUhConnection::connect` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The above example will output:

```
proxy::connect(array (
    0 => NULL,
    1 => 'localhost',
    2 => 'root',
    3 => '',
    4 => 'test',
    5 => 3306,
    6 => NULL,
    7 => 131072,
))
proxy::connect returns true
```

See Also

`mysqlnd_uh_set_connection_proxy`
`mysqli_connect`
`mysql_connect`

22.9.8.7.5. `MySQLndUhConnection::__construct`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MySQLndUhConnection::__construct`

The `__construct` purpose

Description

```
public MySQLndUhConnection::__construct();
```



Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

22.9.8.7.6. `MySQLndUhConnection::endPSession`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MySQLndUhConnection::endPSession`

End a persistent connection

Description

```
public bool MySQLndUhConnection::endPSession(
    mysqlnd_connection connection);
```

End a persistent connection



Warning

This function is currently not documented; only its argument list is available.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.328. `MySQLndUhConnection::endPSession` example

```
<?php
class proxy extends MySQLndUhConnection {
    public function endPSession($conn) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::endPSession($conn);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("p:localhost", "root", "", "test");
$mysqli->close();
?>
```

The above example will output:

```
proxy::endPSession(array (
    0 => NULL,
))
proxy::endPSession returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

22.9.8.7.7. `MySQLndUhConnection::escapeString`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::escapeString](#)

Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection

Description

```
public string MySQLndUhConnection::escapeString(
    mysqlnd_connection connection,
    string escape_string);
```

Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection.

Parameters

[MYSQLND_UH_RES_MYSQLND_NAME](#) MySQLnd connection handle. Do not modify!

[escape_string](#) The string to be escaped.

Return Values

The escaped string.

Examples

Example 22.329. [MySQLndUhConnection::escapeString](#) example

```
<?php
class proxy extends MySQLndUhConnection {
    public function escapeString($res, $string) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::escapeString($res, $string);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->set_charset("latin1");
$mysqli->real_escape_string("test0'test");
?>
```

The above example will output:

```
proxy::escapeString(array (
    0 => NULL,
    1 => 'test0\'test',
))
proxy::escapeString returns 'test0\\\'test'
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_real_escape_string](#)
[mysql_real_escape_string](#)

22.9.8.7.8. [MySQLndUhConnection::getAffectedRows](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::getAffectedRows](#)

Gets the number of affected rows in a previous MySQL operation

Description

```
public int MySQLndUhConnection::getAffectedRows(
    mysqlnd_connection connection);
```

Gets the number of affected rows in a previous MySQL operation.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

Number of affected rows.

Examples

Example 22.330. `MysqlndUhConnection::getAffectedRows` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getAffectedRows($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getAffectedRows($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1)");
var_dump($mysqli->affected_rows);
?>
```

The above example will output:

```
proxy::getAffectedRows(array (
    0 => NULL,
))
proxy::getAffectedRows returns 1
int(1)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_affected_rows](#)
[mysql_affected_rows](#)

22.9.8.7.9. `MysqlndUhConnection::getErrorNumber`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getErrorNumber](#)

Returns the error code for the most recent function call

Description

```
public int MysqlndUhConnection::getErrorNumber(
    mysqlnd_connection connection);
```

Returns the error code for the most recent function call.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Error code for the most recent function call.

Examples

[MysqlndUhConnection::getErrorNumber](#) is not only executed after the invocation of a user space API call which maps directly to it but also called internally.

Example 22.331. [MysqlndUhConnection::getErrorNumber](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getErrorNumber($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getErrorNumber($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

printf("connect...\n");
$mysqli = new mysqli("localhost", "root", "", "test");
printf("query...\n");
$mysqli->query("PLEASE_LET_THIS_BE_INVALID_SQL");
printf("errno...\n");
var_dump($mysqli->errno);
printf("close...\n");
$mysqli->close();
?>
```

The above example will output:

```
connect...
proxy::getErrorNumber(array (
    0 => NULL,
))
proxy::getErrorNumber returns 0
query...
errno...
proxy::getErrorNumber(array (
    0 => NULL,
))
proxy::getErrorNumber returns 1064
int(1064)
close...
```

See Also

- [mysqlnd_uh_set_connection_proxy](#)
- [MysqlndUhConnection::getErrorString](#)
- [mysqli_errno](#)
- [mysql_errno](#)

22.9.8.7.10. [MysqlndUhConnection::getErrorString](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getErrorString](#)

Returns a string description of the last error

Description

```
public string MysqlndUhConnection::getErrorString(
    mysqlnd_connection connection);
```

Returns a string description of the last error.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Error string for the most recent function call.

Examples

[MysqlndUhConnection::getErrorString](#) is not only executed after the invocation of a user space API call which maps directly to it but also called internally.

Example 22.332. [MysqlndUhConnection::getErrorString](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getErrorString($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getErrorString($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

printf("connect...\n");
$mysqli = new mysqli("localhost", "root", "", "test");
printf("query...\n");
$mysqli->query("WILL_I_EVER_LEARN_SQL?");
printf("errno...\n");
var_dump($mysqli->error);
printf("close...\n");
$mysqli->close();
?>
```

The above example will output:

```
connect...
proxy::getErrorString(array (
    0 => NULL,
))
proxy::getErrorString returns ''
query...
errno...
proxy::getErrorString(array (
    0 => NULL,
))
proxy::getErrorString returns 'You have an error in your SQL syntax; check the manual that corresponds to your
string(168) "You have an error in your SQL syntax; check the manual that corresponds to your MySQL server vers
close...
```

See Also

```
mysqlnd_uh_set_connection_proxy
MysqlndUhConnection::getErrorNumber
mysqli_error
mysql_error
```

22.9.8.7.11. [MysqlndUhConnection::getFieldCount](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getFieldCount](#)

Returns the number of columns for the most recent query

Description

```
public int MysqlndUhConnection::getFieldCount(
    mysqlnd_connection connection);
```

Returns the number of columns for the most recent query.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

Number of columns.

Examples

[MysqlndUhConnection::getFieldCount](#) is not only executed after the invocation of a user space API call which maps directly to it but also called internally.

Example 22.333. [MysqlndUhConnection::getFieldCount](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getFieldCount($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getFieldCount($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("WILL_I_EVER_LEARN_SQL?");
var_dump($mysqli->field_count);
$mysqli->query("SELECT 1, 2, 3 FROM DUAL");
var_dump($mysqli->field_count);
?>
```

The above example will output:

```
proxy::getFieldCount(array (
    0 => NULL,
))
proxy::getFieldCount returns 0
```

```
int(0)
proxy::getFieldCount(array (
  0 => NULL,
))
proxy::getFieldCount returns 3
proxy::getFieldCount(array (
  0 => NULL,
))
proxy::getFieldCount returns 3
int(3)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_field_count](#)

22.9.8.7.12. [MysqlndUhConnection::getHostInformation](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getHostInformation](#)

Returns a string representing the type of connection used

Description

```
public string MysqlndUhConnection::getHostInformation(
    mysqlnd_connection connection);
```

Returns a string representing the type of connection used.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Connection description.

Examples

Example 22.334. [MysqlndUhConnection::getHostInformation](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getHostInformation($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getHostInformation($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->host_info);
?>
```

The above example will output:

```
proxy::getHostInformation(array (
    0 => NULL,
))
proxy::getHostInformation returns 'Localhost via UNIX socket'
string(25) "Localhost via UNIX socket"
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_host_info](#)
[mysql_get_host_info](#)

22.9.8.7.13. [MySQLndUhConnection::getLastInsertId](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::getLastInsertId](#)

Returns the auto generated id used in the last query.

Description

```
public int MySQLndUhConnection::getLastInsertId(
    mysqlnd_connection connection);
```

Returns the auto generated id used in the last query.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

Last insert id.

Examples

Example 22.335. [MySQLndUhConnection::getLastInsertId](#) example

```
<?php
class proxy extends MySQLndUhConnection {
    public function getLastInsertId($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getLastInsertId($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT AUTO_INCREMENT PRIMARY KEY, col VARCHAR(255))");
$mysqli->query("INSERT INTO test(col) VALUES ('a')");
var_dump($mysqli->insert_id);
?>
```

The above example will output:

```
proxy::getLastInsertId(array (
  0 => NULL,
))
proxy::getLastInsertId returns 1
int(1)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_insert_id](#)
[mysql_insert_id](#)

22.9.8.7.14. [MysqlndUhConnection::getLastMessage](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getLastMessage](#)

Retrieves information about the most recently executed query

Description

```
public void MysqlndUhConnection::getLastMessage(
    mysqlnd_connection connection);
```

Retrieves information about the most recently executed query.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Last message. Trying to return a string longer than 511 bytes will cause an error of the type [E_WARNING](#) and result in the string being truncated.

Examples

Example 22.336. [MysqlndUhConnection::getLastMessage](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getLastMessage($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getLastMessage($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->info);
$mysqli->query("DROP TABLE IF EXISTS test");
var_dump($mysqli->info);
?>
```

The above example will output:

```
proxy::getLastMessage(array (
  0 => NULL,
))
proxy::getLastMessage returns ''
string(0) ""
proxy::getLastMessage(array (
  0 => NULL,
))
proxy::getLastMessage returns ''
string(0) ""
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_info](#)
[mysql_info](#)

22.9.8.7.15. [MysqlndUhConnection::getProtocolInformation](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getProtocolInformation](#)

Returns the version of the MySQL protocol used

Description

```
public string MysqlndUhConnection::getProtocolInformation(
    mysqlnd_connection connection);
```

Returns the version of the MySQL protocol used.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

The protocol version.

Examples

Example 22.337. [MysqlndUhConnection::getProtocolInformation](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getProtocolInformation($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getProtocolInformation($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->protocol_version);
?>
```

The above example will output:

```
proxy::getProtocolInformation(array (
  0 => NULL,
))
proxy::getProtocolInformation returns 10
int(10)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_proto_info](#)
[mysql_get_proto_info](#)

22.9.8.7.16. [MysqlndUhConnection::getServerInformation](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getServerInformation](#)

Returns the version of the MySQL server

Description

```
public string MysqlndUhConnection::getServerInformation(
    mysqlnd_connection connection);
```

Returns the version of the MySQL server.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

The server version.

Examples

Example 22.338. [MysqlndUhConnection::getServerInformation](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getServerInformation($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getServerInformation($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->server_info);
?>
```

The above example will output:

```
proxy::getServerInformation(array (
    0 => NULL,
))
proxy::getServerInformation returns '5.1.45-debug-log'
string(16) "5.1.45-debug-log"
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_server_info](#)
[mysql_get_server_info](#)

22.9.8.7.17. `MysqlndUhConnection::getServerStatistics`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MysqlndUhConnection::getServerStatistics`

Gets the current system status

Description

```
public string MysqlndUhConnection::getServerStatistics(
    mysqlnd_connection connection);
```

Gets the current system status.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

The system status message.

Examples

Example 22.339. `MysqlndUhConnection::getServerStatistics` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getServerStatistics($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getServerStatistics($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump(mysqli_stat($mysqli));
?>
```

The above example will output:

```
proxy::getServerStatistics(array (
    0 => NULL,
))
proxy::getServerStatistics returns 'Uptime: 2059995 Threads: 1 Questions: 126157 Slow queries: 0 Opens: 63
string(140) "Uptime: 2059995 Threads: 1 Questions: 126157 Slow queries: 0 Opens: 6377 Flush tables: 1 Op
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_stat](#)
[mysql_stat](#)

22.9.8.7.18. [MySqlndUhConnection::getServerVersion](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySqlndUhConnection::getServerVersion](#)

Returns the version of the MySQL server as an integer

Description

```
public int MySqlndUhConnection::getServerVersion(
    mysqlnd_connection connection);
```

Returns the version of the MySQL server as an integer.

Parameters

connection MySqlnd connection handle. Do not modify!

Return Values

The MySQL version.

Examples

Example 22.340. [MySqlndUhConnection::getServerVersion](#) example

```
<?php
class proxy extends MySqlndUhConnection {
    public function getServerVersion($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getServerVersion($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->server_version);
?>
```

The above example will output:

```
proxy::getServerVersion(array (
    0 => NULL,
```

```
))
proxy::getServerVersion returns 50145
int(50145)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_server_version](#)
[mysql_get_server_version](#)

22.9.8.7.19. [MySQLndUhConnection::getSqlstate](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::getSqlstate](#)

Returns the SQLSTATE error from previous MySQL operation

Description

```
public string MySQLndUhConnection::getSqlstate(
    mysqlnd_connection connection);
```

Returns the SQLSTATE error from previous MySQL operation.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

The SQLSTATE code.

Examples

Example 22.341. [MySQLndUhConnection::getSqlstate](#) example

```
<?php
class proxy extends MySQLndUhConnection {
    public function getSqlstate($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getSqlstate($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->sqlstate);
$mysqli->query("AN_INVALID_REQUEST_TO_PROVOKE_AN_ERROR");
var_dump($mysqli->sqlstate);
?>
```

The above example will output:

```
proxy::getSqlstate(array (
    0 => NULL,
```

```

))
proxy::getSqlstate returns '00000'
string(5) "00000"
proxy::getSqlstate(array (
  0 => NULL,
))
proxy::getSqlstate returns '42000'
string(5) "42000"

```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_sql_state](#)

22.9.8.7.20. MysqlndUhConnection::getStatistics

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getStatistics](#)

Returns statistics about the client connection.

Description

```

public array MysqlndUhConnection::getStatistics(
    mysqlnd_connection connection);

```

Returns statistics about the client connection.

**Warning**

This function is currently not documented; only its argument list is available.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Connection statistics collected by mysqlnd.

Examples**Example 22.342. MysqlndUhConnection::getStatistics example**

```

<?php
class proxy extends MysqlndUhConnection {
    public function getStatistics($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getStatistics($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->get_connection_stats());
?>

```

The above example will output:

```
proxy::getStatistics(array (
  0 => NULL,
))
proxy::getStatistics returns array (
  'bytes_sent' => '73',
  'bytes_received' => '77',
  'packets_sent' => '2',
  'packets_received' => '2',
  'protocol_overhead_in' => '8',
  'protocol_overhead_out' => '8',
  'bytes_received_ok_packet' => '0',
  'bytes_received_eof_packet' => '0',
  'bytes_received_rset_header_packet' => '0',
  'bytes_received_rset_field_meta_packet' => '0',
  'bytes_received_rset_row_packet' => '0',
  'bytes_received_prepare_response_packet' => '0',
  'bytes_received_change_user_packet' => '0',
  'packets_sent_command' => '0',
  'packets_received_ok' => '0',
  'packets_received_eof' => '0',
  'packets_received_rset_header' => '0',
  'packets_received_rset_field_meta' => '0',
  'packets_received_rset_row' => '0',
  'packets_received_prepare_response' => '0',
  'packets_received_change_user' => '0',
  'result_set_queries' => '0',
  'non_result_set_queries' => '0',
  'no_index_used' => '0',
  'bad_index_used' => '0',
  'slow_queries' => '0',
  'buffered_sets' => '0',
  'unbuffered_sets' => '0',
  'ps_buffered_sets' => '0',
  'ps_unbuffered_sets' => '0',
  'flushed_normal_sets' => '0',
  'flushed_ps_sets' => '0',
  'ps_prepared_never_executed' => '0',
  'ps_prepared_once_executed' => '0',
  'rows_fetched_from_server_normal' => '0',
  'rows_fetched_from_server_ps' => '0',
  'rows_buffered_from_client_normal' => '0',
  'rows_buffered_from_client_ps' => '0',
  'rows_fetched_from_client_normal_buffered' => '0',
  'rows_fetched_from_client_normal_unbuffered' => '0',
  'rows_fetched_from_client_ps_buffered' => '0',
  'rows_fetched_from_client_ps_unbuffered' => '0',
  'rows_fetched_from_client_ps_cursor' => '0',
  'rows_affected_normal' => '0',
  'rows_affected_ps' => '0',
  'rows_skipped_normal' => '0',
  'rows_skipped_ps' => '0',
  'copy_on_write_saved' => '0',
  'copy_on_write_performed' => '0',
  'command_buffer_too_small' => '0',
  'connect_success' => '1',
  'connect_failure' => '0',
  'connection_reused' => '0',
  'reconnect' => '0',
  'pconnect_success' => '0',
  'active_connections' => '1',
  'active_persistent_connections' => '0',
  'explicit_close' => '0',
```

```
'implicit_close' => '0',
'disconnect_close' => '0',
'in_middle_of_command_close' => '0',
'explicit_free_result' => '0',
'implicit_free_result' => '0',
'explicit_stmt_close' => '0',
'implicit_stmt_close' => '0',
'mem_emalloc_count' => '0',
'mem_emalloc_amount' => '0',
'mem_ecalloc_count' => '0',
'mem_ecalloc_amount' => '0',
'mem_erealloc_count' => '0',
'mem_erealloc_amount' => '0',
'mem_efree_count' => '0',
'mem_efree_amount' => '0',
'mem_malloc_count' => '0',
'mem_malloc_amount' => '0',
'mem_calloc_count' => '0',
'mem_calloc_amount' => '0',
'mem_realloc_count' => '0',
'mem_realloc_amount' => '0',
'mem_free_count' => '0',
'mem_free_amount' => '0',
'mem_strdup_count' => '0',
'mem_strndup_count' => '0',
'mem_estndup_count' => '0',
'mem_strdup_count' => '0',
'proto_text_fetched_null' => '0',
'proto_text_fetched_bit' => '0',
'proto_text_fetched_tinyint' => '0',
'proto_text_fetched_short' => '0',
'proto_text_fetched_int24' => '0',
'proto_text_fetched_int' => '0',
'proto_text_fetched_bigint' => '0',
'proto_text_fetched_decimal' => '0',
'proto_text_fetched_float' => '0',
'proto_text_fetched_double' => '0',
'proto_text_fetched_date' => '0',
'proto_text_fetched_year' => '0',
'proto_text_fetched_time' => '0',
'proto_text_fetched_datetime' => '0',
'proto_text_fetched_timestamp' => '0',
'proto_text_fetched_string' => '0',
'proto_text_fetched_blob' => '0',
'proto_text_fetched_enum' => '0',
'proto_text_fetched_set' => '0',
'proto_text_fetched_geometry' => '0',
'proto_text_fetched_other' => '0',
'proto_binary_fetched_null' => '0',
'proto_binary_fetched_bit' => '0',
'proto_binary_fetched_tinyint' => '0',
'proto_binary_fetched_short' => '0',
'proto_binary_fetched_int24' => '0',
'proto_binary_fetched_int' => '0',
'proto_binary_fetched_bigint' => '0',
'proto_binary_fetched_decimal' => '0',
'proto_binary_fetched_float' => '0',
'proto_binary_fetched_double' => '0',
'proto_binary_fetched_date' => '0',
'proto_binary_fetched_year' => '0',
'proto_binary_fetched_time' => '0',
'proto_binary_fetched_datetime' => '0',
'proto_binary_fetched_timestamp' => '0',
'proto_binary_fetched_string' => '0',
'proto_binary_fetched_blob' => '0',
'proto_binary_fetched_enum' => '0',
'proto_binary_fetched_set' => '0',
```

```

'proto_binary_fetched_geometry' => '0',
'proto_binary_fetched_other' => '0',
'init_command_executed_count' => '0',
'init_command_failed_count' => '0',
'com_quit' => '0',
'com_init_db' => '0',
'com_query' => '0',
'com_field_list' => '0',
'com_create_db' => '0',
'com_drop_db' => '0',
'com_refresh' => '0',
'com_shutdown' => '0',
'com_statistics' => '0',
'com_process_info' => '0',
'com_connect' => '0',
'com_process_kill' => '0',
'com_debug' => '0',
'com_ping' => '0',
'com_time' => '0',
'com_delayed_insert' => '0',
'com_change_user' => '0',
'com_binlog_dump' => '0',
'com_table_dump' => '0',
'com_connect_out' => '0',
'com_register_slave' => '0',
'com_stmt_prepare' => '0',
'com_stmt_execute' => '0',
'com_stmt_send_long_data' => '0',
'com_stmt_close' => '0',
'com_stmt_reset' => '0',
'com_stmt_set_option' => '0',
'com_stmt_fetch' => '0',
'com_daemon' => '0',
'bytes_received_real_data_normal' => '0',
'bytes_received_real_data_ps' => '0',
)
array(160) {
  ["bytes_sent"]=>
  string(2) "73"
  ["bytes_received"]=>
  string(2) "77"
  ["packets_sent"]=>
  string(1) "2"
  ["packets_received"]=>
  string(1) "2"
  ["protocol_overhead_in"]=>
  string(1) "8"
  ["protocol_overhead_out"]=>
  string(1) "8"
  ["bytes_received_ok_packet"]=>
  string(1) "0"
  ["bytes_received_eof_packet"]=>
  string(1) "0"
  ["bytes_received_rset_header_packet"]=>
  string(1) "0"
  ["bytes_received_rset_field_meta_packet"]=>
  string(1) "0"
  ["bytes_received_rset_row_packet"]=>
  string(1) "0"
  ["bytes_received_prepare_response_packet"]=>
  string(1) "0"
  ["bytes_received_change_user_packet"]=>
  string(1) "0"
  ["packets_sent_command"]=>
  string(1) "0"
  ["packets_received_ok"]=>
  string(1) "0"

```

```

["packets_received_eof"]=>
string(1) "0"
["packets_received_rset_header"]=>
string(1) "0"
["packets_received_rset_field_meta"]=>
string(1) "0"
["packets_received_rset_row"]=>
string(1) "0"
["packets_received_prepare_response"]=>
string(1) "0"
["packets_received_change_user"]=>
string(1) "0"
["result_set_queries"]=>
string(1) "0"
["non_result_set_queries"]=>
string(1) "0"
["no_index_used"]=>
string(1) "0"
["bad_index_used"]=>
string(1) "0"
["slow_queries"]=>
string(1) "0"
["buffered_sets"]=>
string(1) "0"
["unbuffered_sets"]=>
string(1) "0"
["ps_buffered_sets"]=>
string(1) "0"
["ps_unbuffered_sets"]=>
string(1) "0"
["flushed_normal_sets"]=>
string(1) "0"
["flushed_ps_sets"]=>
string(1) "0"
["ps_prepared_never_executed"]=>
string(1) "0"
["ps_prepared_once_executed"]=>
string(1) "0"
["rows_fetched_from_server_normal"]=>
string(1) "0"
["rows_fetched_from_server_ps"]=>
string(1) "0"
["rows_buffered_from_client_normal"]=>
string(1) "0"
["rows_buffered_from_client_ps"]=>
string(1) "0"
["rows_fetched_from_client_normal_buffered"]=>
string(1) "0"
["rows_fetched_from_client_normal_unbuffered"]=>
string(1) "0"
["rows_fetched_from_client_ps_buffered"]=>
string(1) "0"
["rows_fetched_from_client_ps_unbuffered"]=>
string(1) "0"
["rows_fetched_from_client_ps_cursor"]=>
string(1) "0"
["rows_affected_normal"]=>
string(1) "0"
["rows_affected_ps"]=>
string(1) "0"
["rows_skipped_normal"]=>
string(1) "0"
["rows_skipped_ps"]=>
string(1) "0"
["copy_on_write_saved"]=>
string(1) "0"
["copy_on_write_performed"]=>

```

```
string(1) "0"
["command_buffer_too_small"]=>
string(1) "0"
["connect_success"]=>
string(1) "1"
["connect_failure"]=>
string(1) "0"
["connection_reused"]=>
string(1) "0"
["reconnect"]=>
string(1) "0"
["pconnect_success"]=>
string(1) "0"
["active_connections"]=>
string(1) "1"
["active_persistent_connections"]=>
string(1) "0"
["explicit_close"]=>
string(1) "0"
["implicit_close"]=>
string(1) "0"
["disconnect_close"]=>
string(1) "0"
["in_middle_of_command_close"]=>
string(1) "0"
["explicit_free_result"]=>
string(1) "0"
["implicit_free_result"]=>
string(1) "0"
["explicit_stmt_close"]=>
string(1) "0"
["implicit_stmt_close"]=>
string(1) "0"
["mem_emalloc_count"]=>
string(1) "0"
["mem_emalloc_amount"]=>
string(1) "0"
["mem_ecalloc_count"]=>
string(1) "0"
["mem_ecalloc_amount"]=>
string(1) "0"
["mem_erealloc_count"]=>
string(1) "0"
["mem_erealloc_amount"]=>
string(1) "0"
["mem_efree_count"]=>
string(1) "0"
["mem_efree_amount"]=>
string(1) "0"
["mem_malloc_count"]=>
string(1) "0"
["mem_malloc_amount"]=>
string(1) "0"
["mem_calloc_count"]=>
string(1) "0"
["mem_calloc_amount"]=>
string(1) "0"
["mem_realloc_count"]=>
string(1) "0"
["mem_realloc_amount"]=>
string(1) "0"
["mem_free_count"]=>
string(1) "0"
["mem_free_amount"]=>
string(1) "0"
["mem_strdup_count"]=>
string(1) "0"
```

```

["mem_strndup_count"]=>
string(1) "0"
["mem_estndup_count"]=>
string(1) "0"
["mem_strdup_count"]=>
string(1) "0"
["proto_text_fetched_null"]=>
string(1) "0"
["proto_text_fetched_bit"]=>
string(1) "0"
["proto_text_fetched_tinyint"]=>
string(1) "0"
["proto_text_fetched_short"]=>
string(1) "0"
["proto_text_fetched_int24"]=>
string(1) "0"
["proto_text_fetched_int"]=>
string(1) "0"
["proto_text_fetched_bigint"]=>
string(1) "0"
["proto_text_fetched_decimal"]=>
string(1) "0"
["proto_text_fetched_float"]=>
string(1) "0"
["proto_text_fetched_double"]=>
string(1) "0"
["proto_text_fetched_date"]=>
string(1) "0"
["proto_text_fetched_year"]=>
string(1) "0"
["proto_text_fetched_time"]=>
string(1) "0"
["proto_text_fetched_datetime"]=>
string(1) "0"
["proto_text_fetched_timestamp"]=>
string(1) "0"
["proto_text_fetched_string"]=>
string(1) "0"
["proto_text_fetched_blob"]=>
string(1) "0"
["proto_text_fetched_enum"]=>
string(1) "0"
["proto_text_fetched_set"]=>
string(1) "0"
["proto_text_fetched_geometry"]=>
string(1) "0"
["proto_text_fetched_other"]=>
string(1) "0"
["proto_binary_fetched_null"]=>
string(1) "0"
["proto_binary_fetched_bit"]=>
string(1) "0"
["proto_binary_fetched_tinyint"]=>
string(1) "0"
["proto_binary_fetched_short"]=>
string(1) "0"
["proto_binary_fetched_int24"]=>
string(1) "0"
["proto_binary_fetched_int"]=>
string(1) "0"
["proto_binary_fetched_bigint"]=>
string(1) "0"
["proto_binary_fetched_decimal"]=>
string(1) "0"
["proto_binary_fetched_float"]=>
string(1) "0"
["proto_binary_fetched_double"]=>

```

```

string(1) "0"
["proto_binary_fetched_date"]=>
string(1) "0"
["proto_binary_fetched_year"]=>
string(1) "0"
["proto_binary_fetched_time"]=>
string(1) "0"
["proto_binary_fetched_datetime"]=>
string(1) "0"
["proto_binary_fetched_timestamp"]=>
string(1) "0"
["proto_binary_fetched_string"]=>
string(1) "0"
["proto_binary_fetched_blob"]=>
string(1) "0"
["proto_binary_fetched_enum"]=>
string(1) "0"
["proto_binary_fetched_set"]=>
string(1) "0"
["proto_binary_fetched_geometry"]=>
string(1) "0"
["proto_binary_fetched_other"]=>
string(1) "0"
["init_command_executed_count"]=>
string(1) "0"
["init_command_failed_count"]=>
string(1) "0"
["com_quit"]=>
string(1) "0"
["com_init_db"]=>
string(1) "0"
["com_query"]=>
string(1) "0"
["com_field_list"]=>
string(1) "0"
["com_create_db"]=>
string(1) "0"
["com_drop_db"]=>
string(1) "0"
["com_refresh"]=>
string(1) "0"
["com_shutdown"]=>
string(1) "0"
["com_statistics"]=>
string(1) "0"
["com_process_info"]=>
string(1) "0"
["com_connect"]=>
string(1) "0"
["com_process_kill"]=>
string(1) "0"
["com_debug"]=>
string(1) "0"
["com_ping"]=>
string(1) "0"
["com_time"]=>
string(1) "0"
["com_delayed_insert"]=>
string(1) "0"
["com_change_user"]=>
string(1) "0"
["com_binlog_dump"]=>
string(1) "0"
["com_table_dump"]=>
string(1) "0"
["com_connect_out"]=>
string(1) "0"

```

```
["com_register_slave"]=>
string(1) "0"
["com_stmt_prepare"]=>
string(1) "0"
["com_stmt_execute"]=>
string(1) "0"
["com_stmt_send_long_data"]=>
string(1) "0"
["com_stmt_close"]=>
string(1) "0"
["com_stmt_reset"]=>
string(1) "0"
["com_stmt_set_option"]=>
string(1) "0"
["com_stmt_fetch"]=>
string(1) "0"
["com_deamon"]=>
string(1) "0"
["bytes_received_real_data_normal"]=>
string(1) "0"
["bytes_received_real_data_ps"]=>
string(1) "0"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_connection_stats](#)

22.9.8.7.21. [MysqlndUhConnection::getThreadId](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::getThreadId](#)

Returns the thread ID for the current connection

Description

```
public int MysqlndUhConnection::getThreadId(
    mysqlnd_connection connection);
```

Returns the thread ID for the current connection.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Connection thread id.

Examples

Example 22.343. [MysqlndUhConnection::getThreadId](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getThreadId($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
    }
}
```

```

$ret = parent::getThreadId($res);
printf("%s returns %s\n", __METHOD__, var_export($ret, true));
return $ret;
}
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->thread_id);
?>

```

The above example will output:

```

proxy::getThreadId(array (
  0 => NULL,
))
proxy::getThreadId returns 27646
int(27646)

```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_thread_id](#)
[mysql_thread_id](#)

22.9.8.7.22. [MySQLndUhConnection::getWarningCount](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::getWarningCount](#)

Returns the number of warnings from the last query for the given link

Description

```

public int MySQLndUhConnection::getWarningCount(
    mysqlnd_connection connection);

```

Returns the number of warnings from the last query for the given link.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

Number of warnings.

Examples

Example 22.344. [MySQLndUhConnection::getWarningCount](#) example

```

<?php
class proxy extends MySQLndUhConnection {
    public function getWarningCount($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getWarningCount($res);
    }
}

```

```

printf("%s returns %s\n", __METHOD__, var_export($ret, true));
return $ret;
}
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->warning_count);
?>

```

The above example will output:

```

proxy::getWarningCount(array (
  0 => NULL,
))
proxy::getWarningCount returns 0
int(0)

```

See Also

[mysqlnd_uh_set_connection_proxy](#)

[mysqli_warning_count](#)

22.9.8.7.23. [MysqlndUhConnection::init](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::init](#)

Initialize mysqlnd connection

Description

```

public bool MysqlndUhConnection::init(
    mysqlnd_connection connection);

```

Initialize mysqlnd connection. This is an mysqlnd internal call to initialize the connection object.



Note

Failing to call the parent implementation may cause memory leaks or crash PHP. This is not considered a bug. Please, keep in mind that the [mysqlnd](#) library functions have never been designed to be exposed to the user space.

Parameters

connection

Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.345. [MysqlndUhConnection::init](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function init($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::init($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The above example will output:

```
proxy::init(array (
    0 => NULL,
))
proxy::init returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

22.9.8.7.24. [MysqlndUhConnection::killConnection](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::killConnection](#)

Asks the server to kill a MySQL thread

Description

```
public bool MysqlndUhConnection::killConnection(
    mysqlnd_connection connection,
    int pid);
```

Asks the server to kill a MySQL thread.

Parameters

| | |
|-------------------|---|
| <i>connection</i> | Mysqlnd connection handle. Do not modify! |
| <i>pid</i> | Thread Id of the connection to be killed. |

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.346. [MysqlndUhConnection::kill](#) example

```
<?php
```

```
class proxy extends MySQLndUhConnection {
    public function killConnection($res, $pid) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::killConnection($res, $pid);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->kill($mysqli->thread_id);
?>
```

The above example will output:

```
proxy::killConnection(array (
    0 => NULL,
    1 => 27650,
))
proxy::killConnection returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_kill](#)

22.9.8.7.25. MySQLndUhConnection::listFields

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::listFields](#)

List MySQL table fields

Description

```
public array MySQLndUhConnection::listFields(
    MySQLndConnection connection,
    string table,
    string achtung_wild);
```

List MySQL table fields.



Warning

This function is currently not documented; only its argument list is available.

Parameters

| | |
|-------------------|---|
| <i>connection</i> | MySQLnd connection handle. Do not modify! |
| <i>table</i> | The name of the table that's being queried. |
| <i>pattern</i> | Name pattern. |

Return Values

Examples

Example 22.347. `MySQLndUhConnection::listFields` example

```
<?php
class proxy extends MySQLndUhConnection {
    public function listFields($res, $table, $pattern) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::listFields($res, $table, $pattern);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysql = mysql_connect("localhost", "root", "");
mysql_select_db("test", $mysql);
mysql_query("DROP TABLE IF EXISTS test_a", $mysql);
mysql_query("CREATE TABLE test_a(id INT, col1 VARCHAR(255))", $mysql);
$res = mysql_list_fields("test", "test_a", $mysql);
printf("num_rows = %d\n", mysql_num_rows($res));
while ($row = mysql_fetch_assoc($res))
    var_dump($row);
?>
```

The above example will output:

```
proxy::listFields(array (
    0 => NULL,
    1 => 'test_a',
    2 => '',
))
proxy::listFields returns NULL
num_rows = 0
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysql_list_fields](#)

22.9.8.7.26. `MySQLndUhConnection::listMethod`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::listMethod](#)

Wrapper for assorted list commands

Description

```
public void MySQLndUhConnection::listMethod(
    mysqlnd_connection connection,
    string query,
    string achtung_wild,
    string par1);
```

Wrapper for assorted list commands.

**Warning**

This function is currently not documented; only its argument list is available.

Parameters

| | |
|---------------------|---|
| <i>connection</i> | Mysqlnd connection handle. Do not modify! |
| <i>query</i> | SHOW command to be executed. |
| <i>achtung_wild</i> | |
| <i>par1</i> | |

Return Values**Return Values**

TODO

Examples**Example 22.348. `MysqlndUhConnection::listMethod` example**

```
<?php
class proxy extends MysqlndUhConnection {
    public function listMethod($res, $query, $pattern, $par1) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::listMethod($res, $query, $pattern, $par1);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysql = mysql_connect("localhost", "root", "");
$res = mysql_list_dbs($mysql);
printf("num_rows = %d\n", mysql_num_rows($res));
while ($row = mysql_fetch_assoc($res))
    var_dump($row);
?>
```

The above example will output:

```
proxy::listMethod(array (
    0 => NULL,
    1 => 'SHOW DATABASES',
    2 => '',
    3 => '',
))
proxy::listMethod returns NULL
num_rows = 6
array(1) {
    ["Database"]=>
        string(18) "information_schema"
}
array(1) {
    ["Database"]=>
```

```

    string(5) "mysql"
}
array(1) {
    ["Database"]=>
        string(8) "oxid_new"
}
array(1) {
    ["Database"]=>
        string(7) "phptest"
}
array(1) {
    ["Database"]=>
        string(7) "pushphp"
}
array(1) {
    ["Database"]=>
        string(4) "test"
}

```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysql_list_dbs](#)

22.9.8.7.27. [MysqlndUhConnection::moreResults](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::moreResults](#)

Check if there are any more query results from a multi query

Description

```

public bool MysqlndUhConnection::moreResults(
    mysqlnd_connection connection);

```

Check if there are any more query results from a multi query.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.349. [MysqlndUhConnection::moreResults](#) example

```

<?php
class proxy extends MysqlndUhConnection {
    public function moreResults($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::moreResults($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}

```

```
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->multi_query("SELECT 1 AS _one; SELECT 2 AS _two");
do {
    $res = $mysqli->store_result();
    var_dump($res->fetch_assoc());
    printf("%s\n", str_repeat("-", 40));
} while ($mysqli->more_results() && $mysqli->next_result());
?>
```

The above example will output:

```
array(1) {
  ["_one"]=>
  string(1) "1"
}
-----
proxy::moreResults(array (
  0 => NULL,
))
proxy::moreResults returns true
proxy::moreResults(array (
  0 => NULL,
))
proxy::moreResults returns true
array(1) {
  ["_two"]=>
  string(1) "2"
}
-----
proxy::moreResults(array (
  0 => NULL,
))
proxy::moreResults returns false
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_more_results](#)

22.9.8.7.28. [MysqlndUhConnection::nextResult](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::nextResult](#)

Prepare next result from `multi_query`

Description

```
public bool MysqlndUhConnection::nextResult(
    mysqlnd_connection connection);
```

Prepare next result from `multi_query`.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.350. [MysqlndUhConnection::nextResult](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function nextResult($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::nextResult($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->multi_query("SELECT 1 AS _one; SELECT 2 AS _two");
do {
    $res = $mysqli->store_result();
    var_dump($res->fetch_assoc());
    printf("%s\n", str_repeat("-", 40));
} while ($mysqli->more_results() && $mysqli->next_result());
?>
```

The above example will output:

```
array(1) {
  ["_one"]=>
  string(1) "1"
}
-----
proxy::nextResult(array (
  0 => NULL,
))
proxy::nextResult returns true
array(1) {
  ["_two"]=>
  string(1) "2"
}
-----
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_next_result](#)

22.9.8.7.29. [MysqlndUhConnection::ping](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::ping](#)

Pings a server connection, or tries to reconnect if the connection has gone down

Description

```
public bool MysqlndUhConnection::ping(
    mysqlnd_connection connection);
```

Pings a server connection, or tries to reconnect if the connection has gone down.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.351. [MysqlndUhConnection::ping](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function ping($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::ping($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->ping();
?>
```

The above example will output:

```
proxy::ping(array (
  0 => NULL,
))
proxy::ping returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_ping](#)
[mysql_ping](#)

22.9.8.7.30. [MysqlndUhConnection::query](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::query](#)

Performs a query on the database

Description

```
public bool MysqlndUhConnection::query(
    mysqlnd_connection connection,
    string query);
```

Performs a query on the database (COM_QUERY).

Parameters

connection Mysqlnd connection handle. Do not modify!

query The query string.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.352. [MysqlndUhConnection::query](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function query($res, $query) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $query = "SELECT 'How about query rewriting?'";
        $ret = parent::query($res, $query);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$res = $mysqli->query("SELECT 'Welcome mysqlnd_uh!' FROM DUAL");
var_dump($res->fetch_assoc());
?>
```

The above example will output:

```
proxy::query(array (
  0 => NULL,
  1 => 'SELECT \'Welcome mysqlnd_uh!\' FROM DUAL',
))
proxy::query returns true
array(1) {
  ["How about query rewriting?"]=>
    string(26) "How about query rewriting?"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_query](#)
[mysql_query](#)

22.9.8.7.31. [MysqlndUhConnection::queryReadResultsetHeader](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::queryReadResultsetHeader](#)

Read a result set header

Description

```
public bool MysqlndUhConnection::queryReadResultsetHeader(
    mysqlnd_connection connection,
    mysqlnd_statement mysqlnd_stmt);
```

Read a result set header.

Parameters

| | |
|------------------------------|---|
| connection | Mysqlnd connection handle. Do not modify! |
| mysqlnd_stmt | Mysqlnd statement handle. Do not modify! Set to <code>NULL</code> , if function is not used in the context of a prepared statement. |

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.353. [MysqlndUhConnection::queryReadResultsetHeader](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function queryReadResultsetHeader($res, $stmt) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::queryReadResultsetHeader($res, $stmt);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$res = $mysqli->query("SELECT 'Welcome mysqlnd_uh!' FROM DUAL");
var_dump($res->fetch_assoc());
?>
```

The above example will output:

```
proxy::queryReadResultsetHeader(array (
    0 => NULL,
    1 => NULL,
))
proxy::queryReadResultsetHeader returns true
array(1) {
    ["Welcome mysqlnd_uh!"]=>
    string(19) "Welcome mysqlnd_uh!"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

22.9.8.7.32. `MysqlndUhConnection::reapQuery`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MysqlndUhConnection::reapQuery`

Get result from async query

Description

```
public bool MysqlndUhConnection::reapQuery(
    mysqlnd_connection connection);
```

Get result from async query.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.354. `MysqlndUhConnection::reapQuery` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function reapQuery($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::reapQuery($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$conn1 = new mysqli("localhost", "root", "", "test");
$conn2 = new mysqli("localhost", "root", "", "test");

$conn1->query("SELECT 1 as 'one', SLEEP(1) AS _sleep FROM DUAL", MYSQLI_ASYNC | MYSQLI_USE_RESULT);
$conn2->query("SELECT 1.1 as 'one dot one' FROM DUAL", MYSQLI_ASYNC | MYSQLI_USE_RESULT);

$links = array(
    $conn1->thread_id => array('link' => $conn1, 'processed' => false),
    $conn2->thread_id => array('link' => $conn2, 'processed' => false)
);

$saved_errors = array();
do {
    $poll_links = $poll_errors = $poll_reject = array();
    foreach ($links as $thread_id => $link) {
        if (!$link['processed']) {
            $poll_links[] = $link['link'];
            $poll_errors[] = $link['link'];
            $poll_reject[] = $link['link'];
        }
    }
}
if (0 == count($poll_links))
    break;

if (0 == ($num_ready = mysqli_poll($poll_links, $poll_errors, $poll_reject, 0, 20000)))
    continue;
```

```

if (!empty($poll_errors)) {
    die(var_dump($poll_errors));
}

foreach ($poll_links as $link) {
    $thread_id = mysqli_thread_id($link);
    $links[$thread_id]['processed'] = true;

    if (is_object($res = mysqli_reap_async_query($link))) {
        // result set object
        while ($row = mysqli_fetch_assoc($res)) {
            // eat up all results
            var_dump($row);
        }
        mysqli_free_result($res);
    } else {
        // either there is no result (no SELECT) or there is an error
        if (mysqli_errno($link) > 0) {
            $saved_errors[$thread_id] = mysqli_errno($link);
            printf("%s' caused %d\n", $links[$thread_id]['query'], mysqli_errno($link));
        }
    }
}
} while (true);
?>

```

The above example will output:

```

proxy::reapQuery(array (
    0 => NULL,
))
proxy::reapQuery returns true
array(1) {
    ["one dot one"]=>
        string(3) "1.1"
}
proxy::reapQuery(array (
    0 => NULL,
))
proxy::reapQuery returns true
array(2) {
    ["one"]=>
        string(1) "1"
    ["_sleep"]=>
        string(1) "0"
}

```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_real_async_query](#)

22.9.8.7.33. [MysqlndUhConnection::refreshServer](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::refreshServer](#)

Flush or reset tables and caches

Description

```
public bool MySQLndUhConnection::refreshServer(
    mysqlnd_connection connection,
    int options);
```

Flush or reset tables and caches.



Warning

This function is currently not documented; only its argument list is available.

Parameters

connection MySQLnd connection handle. Do not modify!

options What to refresh.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.355. [MySQLndUhConnection::refreshServer](#) example

```
<?php
class proxy extends MySQLndUhConnection {
    public function refreshServer($res, $option) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::refreshServer($res, $option);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
mysqli_refresh($mysqli, 1);
?>
```

The above example will output:

```
proxy::refreshServer(array (
    0 => NULL,
    1 => 1,
))
proxy::refreshServer returns false
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

22.9.8.7.34. [MySQLndUhConnection::restartPSession](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

Description

```
public bool MySQLndUhConnection::selectDb(
    mysqlnd_connection connection,
    string database);
```

Selects the default database for database queries.

Parameters

connection MySQLnd connection handle. Do not modify!

database The database name.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.357. [MySQLndUhConnection::selectDb](#) example

```
<?php
class proxy extends MySQLndUhConnection {
    public function selectDb($res, $database) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::selectDb($res, $database);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->select_db("mysql");
?>
```

The above example will output:

```
proxy::selectDb(array (
    0 => NULL,
    1 => 'mysql',
))
proxy::selectDb returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_select_db](#)
[mysql_select_db](#)

22.9.8.7.36. [MySQLndUhConnection::sendClose](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::sendClose](#)

Sends a close command to MySQL

Description

```
public bool MysqlndUhConnection::sendClose(
    mysqlnd_connection connection);
```

Sends a close command to MySQL.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.358. [MysqlndUhConnection::sendClose](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function sendClose($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::sendClose($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->close();
?>
```

The above example will output:

```
proxy::sendClose(array (
    0 => NULL,
))
proxy::sendClose returns true
proxy::sendClose(array (
    0 => NULL,
))
proxy::sendClose returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

22.9.8.7.37. [MysqlndUhConnection::sendQuery](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::sendQuery](#)

Sends a query to MySQL

Description

```
public bool MysqlndUhConnection::sendQuery(
    mysqlnd_connection connection,
    string query);
```

Sends a query to MySQL.

Parameters

connection MySQLnd connection handle. Do not modify!

query The query string.

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples

Example 22.359. [MysqlndUhConnection::sendQuery](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function sendQuery($res, $query) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::sendQuery($res, $query);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("SELECT 1");
?>
```

The above example will output:

```
proxy::sendQuery(array (
    0 => NULL,
    1 => 'SELECT 1',
))
proxy::sendQuery returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

22.9.8.7.38. [MysqlndUhConnection::serverDumpDebugInformation](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::serverDumpDebugInformation](#)

Dump debugging information into the log for the MySQL server

Description

```
public bool MysqlndUhConnection::serverDumpDebugInformation(
```

```
mysqlnd_connection connection);
```

Dump debugging information into the log for the MySQL server.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.360. [MysqlndUhConnection::serverDumpDebugInformation](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function serverDumpDebugInformation($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::serverDumpDebugInformation($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->dump_debug_info();
?>
```

The above example will output:

```
proxy::serverDumpDebugInformation(array (
    0 => NULL,
))
proxy::serverDumpDebugInformation returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_dump_debug_info](#)

22.9.8.7.39. [MysqlndUhConnection::setAutocommit](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::setAutocommit](#)

Turns on or off auto-committing database modifications

Description

```
public bool MysqlndUhConnection::setAutocommit(
    mysqlnd_connection connection,
    int mode);
```

Turns on or off auto-committing database modifications

Parameters

| | |
|-------------------|---|
| <i>connection</i> | Mysqlnd connection handle. Do not modify! |
| <i>mode</i> | Whether to turn on auto-commit or not. |

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples

Example 22.361. `MysqlndUhConnection::setAutocommit` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function setAutocommit($res, $mode) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::setAutocommit($res, $mode);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->autocommit(false);
$mysqli->autocommit(true);
?>
```

The above example will output:

```
proxy::setAutocommit(array (
    0 => NULL,
    1 => 0,
))
proxy::setAutocommit returns true
proxy::setAutocommit(array (
    0 => NULL,
    1 => 1,
))
proxy::setAutocommit returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_autocommit](#)

22.9.8.7.40. `MysqlndUhConnection::setCharset`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MysqlndUhConnection::setCharset`

Sets the default client character set

Description

```
public bool MySQLndUhConnection::setCharset(
    mysqlnd_connection connection,
    string charset);
```

Sets the default client character set.

Parameters

connection MySQLnd connection handle. Do not modify!

charset The charset to be set as default.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.362. [MySQLndUhConnection::setCharset](#) example

```
<?php
class proxy extends MySQLndUhConnection {
    public function setCharset($res, $charset) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::setCharset($res, $charset);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->set_charset("latin1");
?>
```

The above example will output:

```
proxy::setCharset(array (
    0 => NULL,
    1 => 'latin1',
))
proxy::setCharset returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_set_charset](#)

22.9.8.7.41. [MySQLndUhConnection::setClientOption](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::setClientOption](#)

Sets a client option

Description

```
public bool MySQLndUhConnection::setClientOption(
    mysqlnd_connection connection,
    int option,
    int value);
```

Sets a client option.

Parameters

`connection` MySQLnd connection handle. Do not modify!

`option` The option to be set.

`value` Optional option value, if required.

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.363. `MySQLndUhConnection::setClientOption` example

```
<?php
function client_option_to_string($option) {
    static $mapping = array(
        MYSQLND_UH_MYSQLND_OPTION_OPT_CONNECT_TIMEOUT => "MYSQLND_UH_MYSQLND_OPTION_OPT_CONNECT_TIMEOUT",
        MYSQLND_UH_MYSQLND_OPTION_OPT_COMPRESS => "MYSQLND_UH_MYSQLND_OPTION_OPT_COMPRESS",
        MYSQLND_UH_MYSQLND_OPTION_OPT_NAMED_PIPE => "MYSQLND_UH_MYSQLND_OPTION_OPT_NAMED_PIPE",
        MYSQLND_UH_MYSQLND_OPTION_INIT_COMMAND => "MYSQLND_UH_MYSQLND_OPTION_INIT_COMMAND",
        MYSQLND_UH_MYSQLND_READ_DEFAULT_FILE => "MYSQLND_UH_MYSQLND_READ_DEFAULT_FILE",
        MYSQLND_UH_MYSQLND_READ_DEFAULT_GROUP => "MYSQLND_UH_MYSQLND_READ_DEFAULT_GROUP",
        MYSQLND_UH_MYSQLND_SET_CHARSET_DIR => "MYSQLND_UH_MYSQLND_SET_CHARSET_DIR",
        MYSQLND_UH_MYSQLND_SET_CHARSET_NAME => "MYSQLND_UH_MYSQLND_SET_CHARSET_NAME",
        MYSQLND_UH_MYSQLND_OPT_LOCAL_INFILE => "MYSQLND_UH_MYSQLND_OPT_LOCAL_INFILE",
        MYSQLND_UH_MYSQLND_OPT_PROTOCOL => "MYSQLND_UH_MYSQLND_OPT_PROTOCOL",
        MYSQLND_UH_MYSQLND_SHARED_MEMORY_BASE_NAME => "MYSQLND_UH_MYSQLND_SHARED_MEMORY_BASE_NAME",
        MYSQLND_UH_MYSQLND_OPT_READ_TIMEOUT => "MYSQLND_UH_MYSQLND_OPT_READ_TIMEOUT",
        MYSQLND_UH_MYSQLND_OPT_WRITE_TIMEOUT => "MYSQLND_UH_MYSQLND_OPT_WRITE_TIMEOUT",
        MYSQLND_UH_MYSQLND_OPT_USE_RESULT => "MYSQLND_UH_MYSQLND_OPT_USE_RESULT",
        MYSQLND_UH_MYSQLND_OPT_USE_REMOTE_CONNECTION => "MYSQLND_UH_MYSQLND_OPT_USE_REMOTE_CONNECTION",
        MYSQLND_UH_MYSQLND_OPT_USE_EMBEDDED_CONNECTION => "MYSQLND_UH_MYSQLND_OPT_USE_EMBEDDED_CONNECTION",
        MYSQLND_UH_MYSQLND_OPT_GUESS_CONNECTION => "MYSQLND_UH_MYSQLND_OPT_GUESS_CONNECTION",
        MYSQLND_UH_MYSQLND_SET_CLIENT_IP => "MYSQLND_UH_MYSQLND_SET_CLIENT_IP",
        MYSQLND_UH_MYSQLND_SECURE_AUTH => "MYSQLND_UH_MYSQLND_SECURE_AUTH",
        MYSQLND_UH_MYSQLND_REPORT_DATA_TRUNCATION => "MYSQLND_UH_MYSQLND_REPORT_DATA_TRUNCATION",
        MYSQLND_UH_MYSQLND_OPT_RECONNECT => "MYSQLND_UH_MYSQLND_OPT_RECONNECT",
        MYSQLND_UH_MYSQLND_OPT_SSL_VERIFY_SERVER_CERT => "MYSQLND_UH_MYSQLND_OPT_SSL_VERIFY_SERVER_CERT",
        MYSQLND_UH_MYSQLND_OPT_NET_CMD_BUFFER_SIZE => "MYSQLND_UH_MYSQLND_OPT_NET_CMD_BUFFER_SIZE",
        MYSQLND_UH_MYSQLND_OPT_NET_READ_BUFFER_SIZE => "MYSQLND_UH_MYSQLND_OPT_NET_READ_BUFFER_SIZE",
        MYSQLND_UH_MYSQLND_OPT_SSL_KEY => "MYSQLND_UH_MYSQLND_OPT_SSL_KEY",
        MYSQLND_UH_MYSQLND_OPT_SSL_CERT => "MYSQLND_UH_MYSQLND_OPT_SSL_CERT",
        MYSQLND_UH_MYSQLND_OPT_SSL_CA => "MYSQLND_UH_MYSQLND_OPT_SSL_CA",
        MYSQLND_UH_MYSQLND_OPT_SSL_CAPATH => "MYSQLND_UH_MYSQLND_OPT_SSL_CAPATH",
        MYSQLND_UH_MYSQLND_OPT_SSL_CIPHER => "MYSQLND_UH_MYSQLND_OPT_SSL_CIPHER",
        MYSQLND_UH_MYSQLND_OPT_SSL_PASSPHRASE => "MYSQLND_UH_MYSQLND_OPT_SSL_PASSPHRASE",
        MYSQLND_UH_SERVER_OPTION_PLUGIN_DIR => "MYSQLND_UH_SERVER_OPTION_PLUGIN_DIR",
        MYSQLND_UH_SERVER_OPTION_DEFAULT_AUTH => "MYSQLND_UH_SERVER_OPTION_DEFAULT_AUTH",
        MYSQLND_UH_SERVER_OPTION_SET_CLIENT_IP => "MYSQLND_UH_SERVER_OPTION_SET_CLIENT_IP"
    );
    if (version_compare(PHP_VERSION, '5.3.99-dev', '>')) {
        $mapping[MYSQLND_UH_MYSQLND_OPT_MAX_ALLOWED_PACKET] = "MYSQLND_UH_MYSQLND_OPT_MAX_ALLOWED_PACKET";
        $mapping[MYSQLND_UH_MYSQLND_OPT_AUTH_PROTOCOL] = "MYSQLND_UH_MYSQLND_OPT_AUTH_PROTOCOL";
    }
}
```

```
if (defined("MYSQLND_UH_MYSQLND_OPT_INT_AND_FLOAT_NATIVE")) {
    /* special mysqlnd build */
    $mapping["MYSQLND_UH_MYSQLND_OPT_INT_AND_FLOAT_NATIVE"] = "MYSQLND_UH_MYSQLND_OPT_INT_AND_FLOAT_NATIVE";
}
return (isset($mapping[$option])) ? $mapping[$option] : 'unknown';
}

class proxy extends MysqlndUhConnection {
    public function setClientOption($res, $option, $value) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("Option '%s' set to %s\n", client_option_to_string($option), var_export($value, true));
        $ret = parent::setClientOption($res, $option, $value);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The above example will output:

```
proxy::setClientOption(array (
  0 => NULL,
  1 => 210,
  2 => 3221225472,
))
Option 'MYSQLND_UH_MYSQLND_OPT_MAX_ALLOWED_PACKET' set to 3221225472
proxy::setClientOption returns true
proxy::setClientOption(array (
  0 => NULL,
  1 => 211,
  2 => 'mysql_native_password',
))
Option 'MYSQLND_UH_MYSQLND_OPT_AUTH_PROTOCOL' set to 'mysql_native_password'
proxy::setClientOption returns true
proxy::setClientOption(array (
  0 => NULL,
  1 => 8,
  2 => 1,
))
Option 'MYSQLND_UH_MYSQLND_OPT_LOCAL_INFILE' set to 1
proxy::setClientOption returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_real_connect](#)
[mysqli_options](#)

22.9.8.7.42. [MysqlndUhConnection::setServerOption](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::setServerOption](#)

Sets a server option

Description

```
public void MySqlndUhConnection::setServerOption(
    mysqlnd_connection connection,
    int option);
```

Sets a server option.

Parameters

connection MySqlnd connection handle. Do not modify!

option The option to be set.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.364. [MySqlndUhConnection::setServerOption](#) example

```
<?php
function server_option_to_string($option) {
    $ret = 'unknown';
    switch ($option) {
        case MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_ON:
            $ret = 'MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_ON';
            break;
        case MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_OFF:
            $ret = 'MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_ON';
            break;
    }
    return $ret;
}

class proxy extends MySqlndUhConnection {
    public function setServerOption($res, $option) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("Option '%s' set\n", server_option_to_string($option));
        $ret = parent::setServerOption($res, $option);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}

mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->multi_query("SELECT 1; SELECT 2");
?>
```

The above example will output:

```
proxy::setServerOption(array (
    0 => NULL,
    1 => 0,
))
Option 'MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_ON' set
proxy::setServerOption returns true
```

See Also

```
mysqlnd_uh_set_connection_proxy
mysqli_real_connect
mysqli_options
mysqli_multi_query
```

22.9.8.7.43. `MysqlndUhConnection::shutdownServer`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MysqlndUhConnection::shutdownServer`

The `shutdownServer` purpose

Description

```
public void MysqlndUhConnection::shutdownServer(
    string MYSQLND_UH_RES_MYSQLND_NAME,
    string "level");
```



Warning

This function is currently not documented; only its argument list is available.

Parameters

`MYSQLND_UH_RES_MYSQLND_NAME`

`"level"`

Return Values

22.9.8.7.44. `MysqlndUhConnection::simpleCommand`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MysqlndUhConnection::simpleCommand`

Sends a basic `COM_*` command

Description

```
public bool MysqlndUhConnection::simpleCommand(
    mysqlnd_connection connection,
    int command,
    string arg,
    int ok_packet,
    bool silent,
    bool ignore_upsert_status);
```

Sends a basic `COM_*` command to MySQL.

Parameters

| | |
|-------------------------|---|
| <code>connection</code> | Mysqlnd connection handle. Do not modify! |
| <code>command</code> | The COM command to be send. |
| <code>arg</code> | Optional COM command arguments. |

| | |
|-----------------------------------|--|
| <code>ok_packet</code> | The OK packet type. |
| <code>silent</code> | Whether mysqlnd may emit errors. |
| <code>ignore_upsert_status</code> | Whether to ignore <code>UPDATE/INSERT</code> status. |

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.365. `MysqlndUhConnection::simpleCommand` example

```
<?php
function server_cmd_2_string($command) {
    $mapping = array(
        MYSQLND_UH_MYSQLND_COM_SLEEP => "MYSQLND_UH_MYSQLND_COM_SLEEP",
        MYSQLND_UH_MYSQLND_COM_QUIT => "MYSQLND_UH_MYSQLND_COM_QUIT",
        MYSQLND_UH_MYSQLND_COM_INIT_DB => "MYSQLND_UH_MYSQLND_COM_INIT_DB",
        MYSQLND_UH_MYSQLND_COM_QUERY => "MYSQLND_UH_MYSQLND_COM_QUERY",
        MYSQLND_UH_MYSQLND_COM_FIELD_LIST => "MYSQLND_UH_MYSQLND_COM_FIELD_LIST",
        MYSQLND_UH_MYSQLND_COM_CREATE_DB => "MYSQLND_UH_MYSQLND_COM_CREATE_DB",
        MYSQLND_UH_MYSQLND_COM_DROP_DB => "MYSQLND_UH_MYSQLND_COM_DROP_DB",
        MYSQLND_UH_MYSQLND_COM_REFRESH => "MYSQLND_UH_MYSQLND_COM_REFRESH",
        MYSQLND_UH_MYSQLND_COM_SHUTDOWN => "MYSQLND_UH_MYSQLND_COM_SHUTDOWN",
        MYSQLND_UH_MYSQLND_COM_STATISTICS => "MYSQLND_UH_MYSQLND_COM_STATISTICS",
        MYSQLND_UH_MYSQLND_COM_PROCESS_INFO => "MYSQLND_UH_MYSQLND_COM_PROCESS_INFO",
        MYSQLND_UH_MYSQLND_COM_CONNECT => "MYSQLND_UH_MYSQLND_COM_CONNECT",
        MYSQLND_UH_MYSQLND_COM_PROCESS_KILL => "MYSQLND_UH_MYSQLND_COM_PROCESS_KILL",
        MYSQLND_UH_MYSQLND_COM_DEBUG => "MYSQLND_UH_MYSQLND_COM_DEBUG",
        MYSQLND_UH_MYSQLND_COM_PING => "MYSQLND_UH_MYSQLND_COM_PING",
        MYSQLND_UH_MYSQLND_COM_TIME => "MYSQLND_UH_MYSQLND_COM_TIME",
        MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT => "MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT",
        MYSQLND_UH_MYSQLND_COM_CHANGE_USER => "MYSQLND_UH_MYSQLND_COM_CHANGE_USER",
        MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP => "MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP",
        MYSQLND_UH_MYSQLND_COM_TABLE_DUMP => "MYSQLND_UH_MYSQLND_COM_TABLE_DUMP",
        MYSQLND_UH_MYSQLND_COM_CONNECT_OUT => "MYSQLND_UH_MYSQLND_COM_CONNECT_OUT",
        MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVED => "MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVED",
        MYSQLND_UH_MYSQLND_COM_STMT_PREPARE => "MYSQLND_UH_MYSQLND_COM_STMT_PREPARE",
        MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE => "MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE",
        MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA => "MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA",
        MYSQLND_UH_MYSQLND_COM_STMT_CLOSE => "MYSQLND_UH_MYSQLND_COM_STMT_CLOSE",
        MYSQLND_UH_MYSQLND_COM_STMT_RESET => "MYSQLND_UH_MYSQLND_COM_STMT_RESET",
        MYSQLND_UH_MYSQLND_COM_SET_OPTION => "MYSQLND_UH_MYSQLND_COM_SET_OPTION",
        MYSQLND_UH_MYSQLND_COM_STMT_FETCH => "MYSQLND_UH_MYSQLND_COM_STMT_FETCH",
        MYSQLND_UH_MYSQLND_COM_DAEMON => "MYSQLND_UH_MYSQLND_COM_DAEMON",
        MYSQLND_UH_MYSQLND_COM_END => "MYSQLND_UH_MYSQLND_COM_END",
    );
    return (isset($mapping[$command])) ? $mapping[$command] : 'unknown';
}

function ok_packet_2_string($ok_packet) {
    $mapping = array(
        MYSQLND_UH_MYSQLND_PROT_GREET_PACKET => "MYSQLND_UH_MYSQLND_PROT_GREET_PACKET",
        MYSQLND_UH_MYSQLND_PROT_AUTH_PACKET => "MYSQLND_UH_MYSQLND_PROT_AUTH_PACKET",
        MYSQLND_UH_MYSQLND_PROT_OK_PACKET => "MYSQLND_UH_MYSQLND_PROT_OK_PACKET",
        MYSQLND_UH_MYSQLND_PROT_EOF_PACKET => "MYSQLND_UH_MYSQLND_PROT_EOF_PACKET",
        MYSQLND_UH_MYSQLND_PROT_CMD_PACKET => "MYSQLND_UH_MYSQLND_PROT_CMD_PACKET",
        MYSQLND_UH_MYSQLND_PROT_RSET_HEADER_PACKET => "MYSQLND_UH_MYSQLND_PROT_RSET_HEADER_PACKET",
        MYSQLND_UH_MYSQLND_PROT_RSET_FLD_PACKET => "MYSQLND_UH_MYSQLND_PROT_RSET_FLD_PACKET",
        MYSQLND_UH_MYSQLND_PROT_ROW_PACKET => "MYSQLND_UH_MYSQLND_PROT_ROW_PACKET",
        MYSQLND_UH_MYSQLND_PROT_STATS_PACKET => "MYSQLND_UH_MYSQLND_PROT_STATS_PACKET",
        MYSQLND_UH_MYSQLND_PREPARE_RESP_PACKET => "MYSQLND_UH_MYSQLND_PREPARE_RESP_PACKET",
        MYSQLND_UH_MYSQLND_CHG_USER_RESP_PACKET => "MYSQLND_UH_MYSQLND_CHG_USER_RESP_PACKET",
    );
}
```

```

    MYSQLND_UH_MYSQLND_PROT_LAST => "MYSQLND_UH_MYSQLND_PROT_LAST",
    );
    return (isset($mapping[$ok_packet])) ? $mapping[$ok_packet] : 'unknown';
}

class proxy extends MySQLndUhConnection {
    public function simpleCommand($conn, $command, $arg, $ok_packet, $silent, $ignore_upsert_status) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("Command '%s'\n", server_cmd_2_string($command));
        printf("OK packet '%s'\n", ok_packet_2_string($ok_packet));
        $ret = parent::simpleCommand($conn, $command, $arg, $ok_packet, $silent, $ignore_upsert_status);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}

mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("SELECT 1");
?>

```

The above example will output:

```

proxy::simpleCommand(array (
    0 => NULL,
    1 => 3,
    2 => 'SELECT 1',
    3 => 13,
    4 => false,
    5 => false,
))
Command 'MYSQLND_UH_MYSQLND_COM_QUERY'
OK packet 'MYSQLND_UH_MYSQLND_PROT_LAST'
proxy::simpleCommand returns true
:)proxy::simpleCommand(array (
    0 => NULL,
    1 => 1,
    2 => '',
    3 => 13,
    4 => true,
    5 => true,
))
Command 'MYSQLND_UH_MYSQLND_COM_QUIT'
OK packet 'MYSQLND_UH_MYSQLND_PROT_LAST'
proxy::simpleCommand returns true

```

See Also

[mysqlnd_uh_set_connection_proxy](#)

22.9.8.7.45. [MySQLndUhConnection::simpleCommandHandleResponse](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::simpleCommandHandleResponse](#)

Process a response for a basic COM_* command send to the client

Description

```
public bool MySQLndUhConnection::simpleCommandHandleResponse(
```

```
mysqlnd_connection connection,
int ok_packet,
bool silent,
int command,
bool ignore_upsert_status);
```

Process a response for a basic COM_* command send to the client.

Parameters

| | |
|-----------------------------------|--|
| <code>connection</code> | MysqLnd connection handle. Do not modify! |
| <code>ok_packet</code> | The OK packet type. |
| <code>silent</code> | Whether mysqlnd may emit errors. |
| <code>command</code> | The COM command to process results from. |
| <code>ignore_upsert_status</code> | Whether to ignore <code>UPDATE/INSERT</code> status. |

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.366. `MysqlndUhConnection::simpleCommandHandleResponse` example

```
<?php
function server_cmd_2_string($command) {
    $mapping = array(
        MYSQLND_UH_MYSQLND_COM_SLEEP => "MYSQLND_UH_MYSQLND_COM_SLEEP",
        MYSQLND_UH_MYSQLND_COM_QUIT => "MYSQLND_UH_MYSQLND_COM_QUIT",
        MYSQLND_UH_MYSQLND_COM_INIT_DB => "MYSQLND_UH_MYSQLND_COM_INIT_DB",
        MYSQLND_UH_MYSQLND_COM_QUERY => "MYSQLND_UH_MYSQLND_COM_QUERY",
        MYSQLND_UH_MYSQLND_COM_FIELD_LIST => "MYSQLND_UH_MYSQLND_COM_FIELD_LIST",
        MYSQLND_UH_MYSQLND_COM_CREATE_DB => "MYSQLND_UH_MYSQLND_COM_CREATE_DB",
        MYSQLND_UH_MYSQLND_COM_DROP_DB => "MYSQLND_UH_MYSQLND_COM_DROP_DB",
        MYSQLND_UH_MYSQLND_COM_REFRESH => "MYSQLND_UH_MYSQLND_COM_REFRESH",
        MYSQLND_UH_MYSQLND_COM_SHUTDOWN => "MYSQLND_UH_MYSQLND_COM_SHUTDOWN",
        MYSQLND_UH_MYSQLND_COM_STATISTICS => "MYSQLND_UH_MYSQLND_COM_STATISTICS",
        MYSQLND_UH_MYSQLND_COM_PROCESS_INFO => "MYSQLND_UH_MYSQLND_COM_PROCESS_INFO",
        MYSQLND_UH_MYSQLND_COM_CONNECT => "MYSQLND_UH_MYSQLND_COM_CONNECT",
        MYSQLND_UH_MYSQLND_COM_PROCESS_KILL => "MYSQLND_UH_MYSQLND_COM_PROCESS_KILL",
        MYSQLND_UH_MYSQLND_COM_DEBUG => "MYSQLND_UH_MYSQLND_COM_DEBUG",
        MYSQLND_UH_MYSQLND_COM_PING => "MYSQLND_UH_MYSQLND_COM_PING",
        MYSQLND_UH_MYSQLND_COM_TIME => "MYSQLND_UH_MYSQLND_COM_TIME",
        MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT => "MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT",
        MYSQLND_UH_MYSQLND_COM_CHANGE_USER => "MYSQLND_UH_MYSQLND_COM_CHANGE_USER",
        MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP => "MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP",
        MYSQLND_UH_MYSQLND_COM_TABLE_DUMP => "MYSQLND_UH_MYSQLND_COM_TABLE_DUMP",
        MYSQLND_UH_MYSQLND_COM_CONNECT_OUT => "MYSQLND_UH_MYSQLND_COM_CONNECT_OUT",
        MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVED => "MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVED",
        MYSQLND_UH_MYSQLND_COM_STMT_PREPARE => "MYSQLND_UH_MYSQLND_COM_STMT_PREPARE",
        MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE => "MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE",
        MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA => "MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA",
        MYSQLND_UH_MYSQLND_COM_STMT_CLOSE => "MYSQLND_UH_MYSQLND_COM_STMT_CLOSE",
        MYSQLND_UH_MYSQLND_COM_STMT_RESET => "MYSQLND_UH_MYSQLND_COM_STMT_RESET",
        MYSQLND_UH_MYSQLND_COM_SET_OPTION => "MYSQLND_UH_MYSQLND_COM_SET_OPTION",
        MYSQLND_UH_MYSQLND_COM_STMT_FETCH => "MYSQLND_UH_MYSQLND_COM_STMT_FETCH",
        MYSQLND_UH_MYSQLND_COM_DAEMON => "MYSQLND_UH_MYSQLND_COM_DAEMON",
        MYSQLND_UH_MYSQLND_COM_END => "MYSQLND_UH_MYSQLND_COM_END",
    );
}
```

```

return (isset($mapping[$command])) ? $mapping[$command] : 'unknown';
}

function ok_packet_2_string($ok_packet) {
    $mapping = array(
        MYSQLND_UH_MYSQLND_PROT_GREET_PACKET => "MYSQLND_UH_MYSQLND_PROT_GREET_PACKET",
        MYSQLND_UH_MYSQLND_PROT_AUTH_PACKET => "MYSQLND_UH_MYSQLND_PROT_AUTH_PACKET",
        MYSQLND_UH_MYSQLND_PROT_OK_PACKET => "MYSQLND_UH_MYSQLND_PROT_OK_PACKET",
        MYSQLND_UH_MYSQLND_PROT_EOF_PACKET => "MYSQLND_UH_MYSQLND_PROT_EOF_PACKET",
        MYSQLND_UH_MYSQLND_PROT_CMD_PACKET => "MYSQLND_UH_MYSQLND_PROT_CMD_PACKET",
        MYSQLND_UH_MYSQLND_PROT_RSET_HEADER_PACKET => "MYSQLND_UH_MYSQLND_PROT_RSET_HEADER_PACKET",
        MYSQLND_UH_MYSQLND_PROT_RSET_FLD_PACKET => "MYSQLND_UH_MYSQLND_PROT_RSET_FLD_PACKET",
        MYSQLND_UH_MYSQLND_PROT_ROW_PACKET => "MYSQLND_UH_MYSQLND_PROT_ROW_PACKET",
        MYSQLND_UH_MYSQLND_PROT_STATS_PACKET => "MYSQLND_UH_MYSQLND_PROT_STATS_PACKET",
        MYSQLND_UH_MYSQLND_PREPARE_RESP_PACKET => "MYSQLND_UH_MYSQLND_PREPARE_RESP_PACKET",
        MYSQLND_UH_MYSQLND_CHG_USER_RESP_PACKET => "MYSQLND_UH_MYSQLND_CHG_USER_RESP_PACKET",
        MYSQLND_UH_MYSQLND_PROT_LAST => "MYSQLND_UH_MYSQLND_PROT_LAST",
    );
    return (isset($mapping[$ok_packet])) ? $mapping[$ok_packet] : 'unknown';
}

class proxy extends MysqlndUhConnection {
    public function simpleCommandHandleResponse($conn, $ok_packet, $silent, $command, $ignore_upsert_status) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("Command '%s'\n", server_cmd_2_string($command));
        printf("OK packet '%s'\n", ok_packet_2_string($ok_packet));
        $ret = parent::simpleCommandHandleResponse($conn, $ok_packet, $silent, $command, $ignore_upsert_status);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}

mysqlnd_uh_set_connection_proxy(new proxy());
$mysql = mysql_connect("localhost", "root", "");
mysql_query("SELECT 1 FROM DUAL", $mysql);
?>

```

The above example will output:

```

proxy::simpleCommandHandleResponse(array (
    0 => NULL,
    1 => 5,
    2 => false,
    3 => 27,
    4 => true,
))
Command 'MYSQLND_UH_MYSQLND_COM_SET_OPTION'
OK packet 'MYSQLND_UH_MYSQLND_PROT_EOF_PACKET'
proxy::simpleCommandHandleResponse returns true

```

See Also

[mysqlnd_uh_set_connection_proxy](#)

22.9.8.7.46. `MysqlndUhConnection::sslSet`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::sslSet](#)

Used for establishing secure connections using SSL

Description

```
public bool MysqlndUhConnection::sslSet(
    mysqlnd_connection connection,
    string key,
    string cert,
    string ca,
    string capath,
    string cipher);
```

Used for establishing secure connections using SSL.

Parameters

| | |
|-------------------|--|
| <i>connection</i> | Mysqlnd connection handle. Do not modify! |
| <i>key</i> | The path name to the key file. |
| <i>cert</i> | The path name to the certificate file. |
| <i>ca</i> | The path name to the certificate authority file. |
| <i>capath</i> | The pathname to a directory that contains trusted SSL CA certificates in PEM format. |
| <i>cipher</i> | A list of allowable ciphers to use for SSL encryption. |

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.367. `MysqlndUhConnection::sslSet` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function sslSet($conn, $key, $cert, $ca, $capath, $cipher) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::sslSet($conn, $key, $cert, $ca, $capath, $cipher);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->ssl_set("key", "cert", "ca", "capath", "cipher");
?>
```

The above example will output:

```
proxy::sslSet(array (
    0 => NULL,
    1 => 'key',
    2 => 'cert',
    3 => 'ca',
    4 => 'capath',
    5 => 'cipher',
))
proxy::sslSet returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_ssl_set](#)

22.9.8.7.47. [MysqlndUhConnection::stmtInit](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::stmtInit](#)

Initializes a statement and returns a resource for use with `mysqli_statement::prepare`

Description

```
public resource MysqlndUhConnection::stmtInit(  
    mysqlnd_connection connection);
```

Initializes a statement and returns a resource for use with `mysqli_statement::prepare`.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Resource of type [Mysqlnd Prepared Statement](#) (*internal only - you must not modify it!*). The documentation may also refer to such resources using the alias name [mysqlnd_prepared_statement](#).

Examples

Example 22.368. [MysqlndUhConnection::stmtInit](#) example

```
<?php  
class proxy extends MysqlndUhConnection {  
    public function stmtInit($res) {  
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));  
        var_dump($res);  
        $ret = parent::stmtInit($res);  
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));  
        var_dump($ret);  
        return $ret;  
    }  
}  
mysqlnd_uh_set_connection_proxy(new proxy());  
$mysqli = new mysqli("localhost", "root", "", "test");  
$stmt = $mysqli->prepare("SELECT 1 AS _one FROM DUAL");  
$stmt->execute();  
$one = NULL;  
$stmt->bind_result($one);  
$stmt->fetch();  
var_dump($one);  
?>
```

The above example will output:

```
proxy::stmtInit(array (
  0 => NULL,
))
resource(19) of type (MySQLnd Connection)
proxy::stmtInit returns NULL
resource(246) of type (MySQLnd Prepared Statement (internal only - you must not modify it!))
int(1)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_stmt_init](#)

22.9.8.7.48. MySQLndUhConnection::storeResult

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::storeResult](#)

Transfers a result set from the last query

Description

```
public resource MySQLndUhConnection::storeResult(
    mysqlnd_connection connection);
```

Transfers a result set from the last query.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

Resource of type [MySQLnd Resultset \(internal only - you must not modify it!\)](#). The documentation may also refer to such resources using the alias name [mysqlnd_resultset](#).

Examples

Example 22.369. MySQLndUhConnection::storeResult example

```
<?php
class proxy extends MySQLndUhConnection {
    public function storeResult($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::storeResult($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        var_dump($ret);
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$msgsql = new mysqli("localhost", "root", "", "test");
$res = $msgsql->query("SELECT 'Also called buffered result' AS _msg FROM DUAL");
var_dump($res->fetch_assoc());

$msgsql->real_query("SELECT 'Good morning!' AS _msg FROM DUAL");
$res = $msgsql->store_result();
```

```
var_dump($res->fetch_assoc());  
?>
```

The above example will output:

```
proxy::storeResult(array (
  0 => NULL,
))
proxy::storeResult returns NULL
resource(475) of type (Mysqlnd Resultset (internal only - you must not modify it!))
array(1) {
  ["_msg"]=>
    string(27) "Also called buffered result"
}
proxy::storeResult(array (
  0 => NULL,
))
proxy::storeResult returns NULL
resource(730) of type (Mysqlnd Resultset (internal only - you must not modify it!))
array(1) {
  ["_msg"]=>
    string(13) "Good morning!"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_store_result](#)
[mysqli_real_query](#)

22.9.8.7.49. [MysqlndUhConnection::txCommit](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhConnection::txCommit](#)

Commits the current transaction

Description

```
public bool MysqlndUhConnection::txCommit(  
    mysqlnd_connection connection);
```

Commits the current transaction.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.370. [MysqlndUhConnection::txCommit](#) example

```
<?php
class proxy extends MySQLndUhConnection {
    public function txCommit($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::txCommit($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->commit();
?>
```

The above example will output:

```
proxy::txCommit(array (
    0 => NULL,
))
proxy::txCommit returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_commit](#)

22.9.8.7.50. MySQLndUhConnection::txRollback

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MySQLndUhConnection::txRollback](#)

Rolls back current transaction

Description

```
public bool MySQLndUhConnection::txRollback(
    mysqlnd_connection connection);
```

Rolls back current transaction.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.371. MySQLndUhConnection::txRollback example

```
<?php
class proxy extends MySQLndUhConnection {
    public function txRollback($res) {
```

```

printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
$ret = parent::txRollback($res);
printf("%s returns %s\n", __METHOD__, var_export($ret, true));
return $ret;
}
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->rollback();
?>

```

The above example will output:

```

proxy::txRollback(array (
  0 => NULL,
))
proxy::txRollback returns true

```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_commit](#)

22.9.8.7.51. `MysqlndUhConnection::useResult`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `MysqlndUhConnection::useResult`

Initiate a result set retrieval

Description

```

public resource MysqlndUhConnection::useResult(
    mysqlnd_connection connection);

```

Initiate a result set retrieval.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Resource of type `Mysqlnd Resultset (internal only - you must not modify it!)`. The documentation may also refer to such resources using the alias name `mysqlnd_resultset`.

Examples

Example 22.372. `MysqlndUhConnection::useResult` example

```

<?php
class proxy extends MysqlndUhConnection {
    public function useResult($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::useResult($res);
    }
}

```

```
printf("%s returns %s\n", __METHOD__, var_export($ret, true));
var_dump($ret);
return $ret;
}
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->real_query("SELECT 'Good morning!' AS _msg FROM DUAL");
$res = $mysqli->use_result();
var_dump($res->fetch_assoc());
?>
```

The above example will output:

```
proxy::useResult(array (
  0 => NULL,
))
proxy::useResult returns NULL
resource(425) of type (Mysqlnd Resultset (internal only - you must not modify it!))
array(1) {
  ["_msg"]=>
    string(13) "Good morning!"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

[mysqli_use_result](#)

[mysqli_real_query](#)

22.9.8.8. The MysqlndUhPreparedStatement class ([MysqlndUhPreparedStatement](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

```
MysqlndUhPreparedStatement {
MysqlndUhPreparedStatement

    Methods

    public MysqlndUhPreparedStatement::__construct();

    public bool MysqlndUhPreparedStatement::execute(
        mysqlnd_prepared_statement statement);

    public bool MysqlndUhPreparedStatement::prepare(
        mysqlnd_prepared_statement statement,
        string query);
}
```

22.9.8.8.1. [MysqlndUhPreparedStatement::__construct](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhPreparedStatement::__construct](#)

The `__construct` purpose

Description

```
public MysqlndUhpStatement::__construct();
```



Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

22.9.8.8.2. [MysqlndUhpStatement::execute](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhpStatement::execute](#)

Executes a prepared Query

Description

```
public bool MysqlndUhpStatement::execute(
    mysqlnd_prepared_statement statement);
```

Executes a prepared Query.

Parameters

statement

Mysqlnd prepared statement handle. Do not modify! Resource of type [Mysqlnd Prepared Statement \(internal only - you must not modify it!\)](#).

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 22.373. [MysqlndUhpStatement::execute](#) example

```
<?php
class stmt_proxy extends MysqlndUhpStatement {
    public function execute($res) {
        printf("%s(", __METHOD__);
        var_dump($res);
        printf(")\n");
        $ret = parent::execute($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        var_dump($ret);
        return $ret;
    }
}
```

```
mysqlnd_uh_set_statement_proxy(new stmt_proxy());

$msgqli = new mysqli("localhost", "root", "", "test");
$stmt = $mysqli->prepare("SELECT 'Labskaus' AS _msg FROM DUAL");
$stmt->execute();
$msg = NULL;
$stmt->bind_result($msg);
$stmt->fetch();
var_dump($msg);
?>
```

The above example will output:

```
stmt_proxy::execute(resource(256) of type (Mysqlnd Prepared Statement (internal only - you must not modify
))
stmt_proxy::execute returns true
bool(true)
string(8) "Labskaus"
```

See Also

[mysqlnd_uh_set_statement_proxy](#)
[mysqli_stmt_execute](#)

22.9.8.8.3. `MysqlndUhpPreparedStatement::prepare`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [MysqlndUhpPreparedStatement::prepare](#)

Prepare an SQL statement for execution

Description

```
public bool MysqlndUhpPreparedStatement::prepare(
    mysqlnd_prepared_statement statement,
    string query);
```

Prepare an SQL statement for execution.

Parameters

| | |
|------------------|--|
| <i>statement</i> | Mysqlnd prepared statement handle. Do not modify! Resource of type <code>Mysqlnd Prepared Statement (internal only - you must not modify it!)</code> . |
| <i>query</i> | The query to be prepared. |

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.374. `MysqlndUhpPreparedStatement::prepare` example

```
<?php
class stmt_proxy extends MysqlndUhpStatement {
    public function prepare($res, $query) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $query = "SELECT 'No more you-know-what-I-mean for lunch, please' AS _msg FROM DUAL";
        $ret = parent::prepare($res, $query);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        var_dump($ret);
        return $ret;
    }
}
mysqlnd_uh_set_statement_proxy(new stmt_proxy());

$mysqli = new mysqli("localhost", "root", "", "test");
$stmt = $mysqli->prepare("SELECT 'Labskaus' AS _msg FROM DUAL");
$stmt->execute();
$msg = NULL;
$stmt->bind_result($msg);
$stmt->fetch();
var_dump($msg);
?>
```

The above example will output:

```
stmt_proxy::prepare(array (
  0 => NULL,
  1 => 'SELECT \'Labskaus\' AS _msg FROM DUAL',
))
stmt_proxy::prepare returns true
bool(true)
string(46) "No more you-know-what-I-mean for lunch, please"
```

See Also

[mysqlnd_uh_set_statement_proxy](#)
[mysqli_stmt_prepare](#)
[mysqli_prepare](#)

22.9.8.9. Mysqlnd_uh Functions

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.8.9.1. [mysqlnd_uh_convert_to_mysqlnd](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_uh_convert_to_mysqlnd](#)

Converts a MySQL connection handle into a mysqlnd connection handle

Description

```
resource mysqlnd_uh_convert_to_mysqlnd(
    mysqli mysql_connection);
```

Converts a MySQL connection handle into a mysqlnd connection handle. After conversion you can execute mysqlnd library calls on the connection handle. This can be used to access mysqlnd functionality not made available through user space API calls.

The function can be disabled with `mysqlnd_uh.enable`. If `mysqlnd_uh.enable` is set to `FALSE` the function will not install the proxy and always return `TRUE`. Additionally, an error of the type `E_WARNING` may be emitted. The error message may read like `PHP Warning: mysqlnd_uh_convert_to_mysqlnd(): (MySQLnd User Handler) The plugin has been disabled by setting the configuration parameter mysqlnd_uh.enable = false. You are not allowed to call this function [...]`.

Parameters

MySQL connection handle A MySQL connection handle of type `mysql`, `mysqli` or `PDO_MySQL`.

Return Values

A `mysqlnd` connection handle.

Changelog

| Version | Description |
|---------|--|
| 5.4.0 | The <code>mysql_connection</code> parameter can now be of type <code>mysql</code> , <code>PDO_MySQL</code> , or <code>mysqli</code> . Before, only the <code>mysqli</code> type was allowed. |

Examples

Example 22.375. `mysqlnd_uh_convert_to_mysqlnd` example

```
<?php
/* PDO user API gives no access to connection thread id */
$mysql_connection = new PDO("mysql:host=localhost;dbname=test", "root", "");

/* Convert PDO MySQL handle to mysqlnd handle */
$mysqlnd = mysqlnd_uh_convert_to_mysqlnd($mysql_connection);

/* Create Proxy to call mysqlnd connection class methods */
$obj = new MySQLndUHConnection();
/* Call mysqlnd_conn::get_thread_id */
var_dump($obj->getThreadId($mysqlnd));

/* Use SQL to fetch connection thread id */
var_dump($mysql_connection->query("SELECT CONNECTION_ID()")->fetchAll());
?>
```

The above example will output:

```
int(27054)
array(1) {
  [0]=>
  array(2) {
    ["CONNECTION_ID()"]=>
    string(5) "27054"
    [0]=>
    string(5) "27054"
  }
}
```

See Also

`mysqlnd_uh.enable`

22.9.8.9.2. `mysqlnd_uh_set_connection_proxy`

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_uh_set_connection_proxy`

Installs a proxy for mysqlnd connections

Description

```
bool mysqlnd_uh_set_connection_proxy(
    MysqlndUhConnection connection_proxy,
    mysqli mysqli_connection);
```

Installs a proxy object to hook mysqlnd's connection objects methods. Once installed, the proxy will be used for all MySQL connections opened with `mysqli`, `mysql` or `PDO_MYSQL`, assuming that the listed extensions are compiled to use the `mysqlnd` library.

The function can be disabled with `mysqlnd_uh.enable`. If `mysqlnd_uh.enable` is set to `FALSE` the function will not install the proxy and always return `TRUE`. Additionally, an error of the type `E_WARNING` may be emitted. The error message may read like `PHP Warning: mysqlnd_uh_set_connection_proxy(): (Mysqlnd User Handler) The plugin has been disabled by setting the configuration parameter mysqlnd_uh.enable = false. The proxy has not been installed [...]`.

Parameters

| | |
|--------------------------------|---|
| <code>connection_proxy</code> | A proxy object of type <code>MysqlndUhConnection</code> . |
| <code>mysqli_connection</code> | Object of type <code>mysqli</code> . If given, the proxy will be set for this particular connection only. |

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 22.376. `mysqlnd_uh_set_connection_proxy` example

```
<?php
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("SELECT 'No proxy installed, yet'");

class proxy extends MysqlndUhConnection {
    public function query($res, $query) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::query($res, $query);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());

$mysqli->query("SELECT 'mysqlnd rocks!'");

$mysql = mysql_connect("localhost", "root", "", "test");
mysql_query("SELECT 'Ahoy Andrey!'", $mysql);
```

```
$pdo = new PDO("mysql:host=localhost;dbname=test", "root", "");
$pdo->query("SELECT 'Moin Johannes!'");
?>
```

The above example will output:

```
proxy::query(array (
  0 => NULL,
  1 => 'SELECT \'mysqlnd rocks!\'',
))
proxy::query returns true
proxy::query(array (
  0 => NULL,
  1 => 'SELECT \'Ahoy Andrey!\'',
))
proxy::query returns true
proxy::query(array (
  0 => NULL,
  1 => 'SELECT \'Moin Johannes!\'',
))
proxy::query returns true
```

See Also

[mysqlnd_uh_set_statement_proxy](#)
[mysqlnd_uh.enable](#)

22.9.8.9.3. [mysqlnd_uh_set_statement_proxy](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_uh_set_statement_proxy](#)

Installs a proxy for mysqlnd statements

Description

```
bool mysqlnd_uh_set_statement_proxy(
    MySQLndUhStatement statement_proxy);
```

Installs a proxy for mysqlnd statements. The proxy object will be used for all mysqlnd prepared statement objects, regardless which PHP MySQL extension ([mysqli](#), [mysql](#), [PDO_MYSQL](#)) has created them as long as the extension is compiled to use the [mysqlnd](#) library.

The function can be disabled with [mysqlnd_uh.enable](#). If [mysqlnd_uh.enable](#) is set to [FALSE](#) the function will not install the proxy and always return [TRUE](#) . Additionally, an error of the type [E_WARNING](#) may be emitted. The error message may read like [PHP Warning: mysqlnd_uh_set_statement_proxy\(\): \(MySQLnd User Handler\) The plugin has been disabled by setting the configuration parameter mysqlnd_uh.enable = false. The proxy has not been installed \[...\]](#).

Parameters

[statement_proxy](#) The mysqlnd statement proxy object of type [MySQLndUhStatement](#)

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqlnd_uh.enable](#)

22.9.8.10. Change History

Copyright 1997-2012 the PHP Documentation Group. [3186]

The Change History lists major changes users need to be aware of if upgrading from one version to another. It is a high level summary of selected changes that may impact applications or might even break backwards compatibility. See also the [CHANGES](#) file contained in the source for additional changelog information. The commit history is also available.

22.9.8.10.1. PECL/mysqlnd_uh 1.0 series

Copyright 1997-2012 the PHP Documentation Group. [3186]

1.0.1-alpha

- Release date: TBD
- Motto/theme: bug fix release

Feature changes

- Support of PHP 5.4.0 or later.
- BC break: `MysqlndUhConnection::changeUser` requires additional `passwd_len` parameter.
- BC break: `MYSQLND_UH_VERSION_STR` renamed to `MYSQLND_UH_VERSION`. `MYSQLND_UH_VERSION` renamed to `MYSQLND_UH_VERSION_ID`.
- BC break: `mysqlnd_uh.enabled` configuration setting renamed to `mysqlnd_uh.enable`.

1.0.0-alpha

- Release date: 08/2010
- Motto/theme: Initial release

22.9.9. Mysqlnd connection multiplexing plugin ([mysqlnd_mux](#))

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqlnd` multiplexing plugin ([mysqlnd_mux](#)) multiplexes MySQL connections established by all PHP MySQL extensions that use the MySQL native driver ([mysqlnd](#)) for PHP.

The MySQL native driver for PHP features an internal C API for plugins, such as the connection multiplexing plugin, which can extend the functionality of `mysqlnd`. See the [mysqlnd](#) for additional details about its benefits over the MySQL Client Library `libmysql/libmysqlclient`.

`mysqlnd` plugins like [mysqlnd_mux](#) operate, for the most part, transparently from a user perspective. The connection multiplexing plugin supports all PHP applications, and all MySQL PHP extensions. It does not change existing APIs. Therefore, it can easily be used with existing PHP applications.



Note

This is a proof-of-concept. All features are at an early stage. Not all kinds of queries are handled by the plugin yet. Thus, it cannot be used in a drop-in fashion at the moment.

Please, do not use this version in production environments.

22.9.9.1. Key Features

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The key features of `mysqLnd_mux` are as follows:

- Transparent and therefore easy to use:
 - Supports all of the PHP MySQL extensions.
 - Little to no application changes are required, dependent on the required usage scenario.
- Reduces server load and connection establishment latency:
 - Opens less connections to the MySQL server.
 - Less connections to MySQL mean less work for the MySQL server. In a client-server environment scaling the server is often more difficult than scaling the client. Multiplexing helps with horizontal scale-out (scale-by-client).
 - Pooling saves connection time.
- Multiplexed connection: multiple user handles share the same network connection. Once opened, a network connection is cached and shared among multiple user handles. There is a 1:n relationship between internal network connection and user connection handles.
- Persistent connection: a network connection is kept open at the end of the web request, if the PHP deployment model allows. Thus, subsequently web requests can reuse a previously opened connection. Like other resources, network connections are bound to the scope of a process. Thus, they can be reused for all web requests served by a process.

22.9.9.2. Limitations

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The proof-of-concept does not support unbuffered queries, prepared statements, and asynchronous queries.

The connection pool is using a combination of the transport method and hostname as keys. As a consequence, two connections to the same host using the same transport method (TCP/IP, Unix socket, Windows named pipe) will be linked to the same pooled connection even if username and password differ. Be aware of the possible security implications.

The proof-of-concept is transaction agnostic. It does not about SQL transactions.



Note

Applications must be aware of the consequences of connection sharing connections.

22.9.9.3. About the name `mysqlnd_mux`

Copyright 1997-2012 the PHP Documentation Group. [3186]

The shortcut `mysqlnd_mux` stands for `mysqlnd connection multiplexing plugin`.

22.9.9.4. Concepts

Copyright 1997-2012 the PHP Documentation Group. [3186]

This explains the architecture and related concepts for this plugin. Reading and understanding these concepts is required to successfully use this plugin.

22.9.9.4.1. Architecture

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqlnd` connection multiplexing plugin is implemented as a PHP extension. It is written in C and operates under the hood of PHP. During the startup of the PHP interpreter, in the module initialization phase of the PHP engine, it gets registered as a `mysqlnd` plugin to replace specific `mysqlnd` C methods.

The `mysqlnd` library uses PHP streams to communicate with the MySQL server. PHP streams are accessed by the `mysqlnd` library through its net module. The `mysqlnd` connection multiplexing plugin proxies methods of the `mysqlnd` library net module to control opening and closing of network streams.

Upon opening a user connection to MySQL using the appropriate connection functions of either `mysqli`, `PDO_MYSQL` or `ext/mysqli`, the plugin will search its connection pool for an open network connection. If the pool contains a network connection to the host specified by the connect function using the transport method requested (TCP/IP, Unix domain socket, Windows named pipe), the pooled connection is linked to the user handle. Otherwise, a new network connection is opened, put into the pool and associated with the user connection handle. This way, multiple user handles can be linked to the same network connection.

22.9.9.4.2. Connection pool

Copyright 1997-2012 the PHP Documentation Group. [3186]

The plugin's connection pool is created when PHP initializes its modules (`MINIT`) and free'd when PHP shuts down the modules (`MSHUTDOWN`). This is the same as for persistent MySQL connections.

Depending on the deployment model, the pool is used for the duration of one or multiple web requests. Network connections are bound to the lifespan of an operating system level process. If the PHP process serves multiple web requests as it is the case for Fast-CGI or threaded web server deployments, then the pooled connections can be reused over multiple connections. Because multiplexing means sharing connections, it can even happen with a threaded deployment that two threads or two distinct web requests are linked to one pooled network connection.

A pooled connection is explicitly closed once the last reference to it is released. An implicit close happens when PHP shuts down its modules.

22.9.9.4.3. Sharing connections

Copyright 1997-2012 the PHP Documentation Group. [3186]

The PHP `mysqlnd` connection multiplexing plugin changes the relationship between a user's connection handle and the underlying MySQL connection. Without the plugin, every MySQL connection belongs to exactly one user connection at a time. The multiplexing plugin changes. A MySQL connection is shared among multiple user handles. There is no one-to-one relation if using the plugin.

Sharing pooled connections has an impact on the connection state. State changing operations from multiple user handles pointing to one MySQL connection are not isolated from each other. If, for example, a session variable is set through one user connection handle, the session variable becomes visible to all other user handles that reference the same underlying MySQL connection.

This is similar in concept to connection state related phenomena described for the PHP mysqlnd replication and load balancing plugin. Please, check the [PECL/mysqlnd_ms documentation](#) for more details on the state of a connection.

The proof-of-concept takes no measures to isolate multiplexed connections from each other.

22.9.9.5. Installing/Configuring

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

22.9.9.5.1. Requirements

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

PHP 5.5.0 or newer. Some advanced functionality requires PHP 5.5.0 or newer.

The `mysqlnd_mux` replication and load balancing plugin supports all PHP applications and all available PHP MySQL extensions (`mysqli`, `mysql`, `PDO_MYSQL`). The PHP MySQL extension must be configured to use `mysqlnd` in order to be able to use the `mysqlnd_mux` plugin for `mysqlnd`.

22.9.9.5.2. Installation

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: http://pecl.php.net/package/mysqlnd_mux

22.9.9.5.3. Runtime Configuration

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The behaviour of these functions is affected by settings in `php.ini`.

Table 22.75. Mysqlnd_mux Configure Options

| Name | Default | Changeable | Changelog |
|---------------------------------|---------|----------------|-----------|
| <code>mysqlnd_mux.enable</code> | 0 | PHP_INI_SYSTEM | |

Here's a short explanation of the configuration directives.

`mysqlnd_mux.enable` integer Enables or disables the plugin. If disabled, the extension will not plug into `mysqlnd` to proxy internal `mysqlnd` C API calls.

22.9.9.6. Predefined Constants

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Other

The plugins version number can be obtained using `MYSQLND_MUX_VERSION` or `MYSQLND_MUX_VERSION_ID`. `MYSQLND_MUX_VERSION` is the string representation of the numerical version number `MYSQLND_MUX_VERSION_ID`, which is an integer such as 10000. Developers can calculate the version number as follows.

| Version (part) | Example |
|------------------------|-----------------|
| Major*10000 | 1*10000 = 10000 |
| Minor*100 | 0*100 = 0 |
| Patch | 0 = 0 |
| MYSQLND_MUX_VERSION_ID | 10000 |

`MYSQLND_MUX_VERSION` (string) Plugin version string, for example, "1.0.0-prototype".

`MYSQLND_MUX_VERSION_ID` (integer) Plugin version number, for example, 10000.

22.9.9.7. Change History

Copyright 1997-2012 the PHP Documentation Group. [3186]

This change history is a high level summary of selected changes that may impact applications and/or break backwards compatibility.

See also the [CHANGES](#) file in the source distribution for a complete list of changes.

22.9.9.7.1. PECL/mysqlnd_mux 1.0 series

Copyright 1997-2012 the PHP Documentation Group. [3186]

1.0.0-pre-alpha

- Release date: no package released, initial check-in 09/2012
- Motto/theme: Proof of concept

Initial check-in. Essentially a demo of the `mysqlnd` plugin API.



Note

This is the current development series. All features are at an early stage. Changes may happen at any time without prior notice. Please, do not use this version in production environments.

The documentation may not reflect all changes yet.

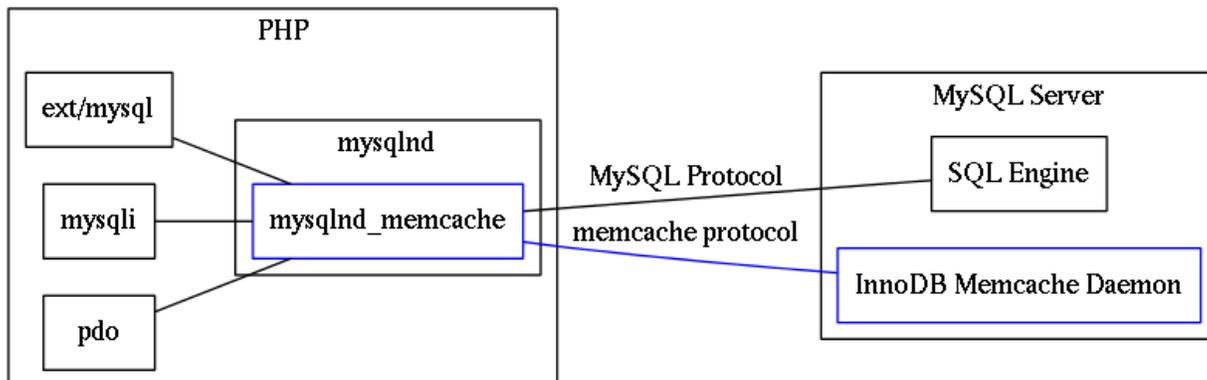
22.9.10. MySQLnd Memache plugin (`mysqlnd_memcache`)

Copyright 1997-2012 the PHP Documentation Group. [3186]

The `mysqlnd memcache` plugin (`mysqlnd_memcache`) is an PHP extension for transparently translating SQL into requests for the MySQL InnoDB Memcached Daemon Plugin (server plugin). It includes experimental support for the MySQL Cluster Memcached Daemon. The server plugin provides access to

data stored inside MySQL InnoDB (respectively MySQL Cluster NDB) tables using the Memcache protocol. This PHP extension, which supports all PHP MySQL extensions that use [mysqlnd](#), will identify tables exported in this way and will translate specific SELECT queries into Memcache requests.

Figure 22.92. mysqlnd_memcache data flow



Note

This plugin depends on the MySQL InnoDB Memcached Daemon Plugin. It is not provided to be used with a stand-alone Memcached. For a generic query cache using Memcached look at the [mysqlnd query cache plugin](#). For direct Memcache access look at the [memcache](#) and [memcached](#) extensions.

The MySQL native driver for PHP is a C library that ships together with PHP as of PHP 5.3.0. It serves as a drop-in replacement for the MySQL Client Library (libmysql/libmysqlclient). Using mysqlnd has several advantages: no extra downloads are required because it's bundled with PHP, it's under the PHP license, there is lower memory consumption in certain cases, and it contains new functionality such as asynchronous queries.

The [mysqlnd_mmemcache](#) operates, for the most part, transparently from a user perspective. The mysqlnd memcache plugin supports all PHP applications, and all MySQL PHP extensions. It does not change existing APIs. Therefore, it can easily be used with existing PHP applications.

The MySQL Memcache plugins add key-value style access method for data stored in InnoDB resp. NDB (MySQL Cluster) SQL tables through the Memcache protocol. This type of key-value access is often faster than using SQL.

22.9.10.1. Key Features

Copyright 1997-2012 the PHP Documentation Group. [3186]

The key features of PECL/mysqlnd_memcache are as follows.

- Possible performance benefits
 - Client-side: light-weight protocol.
 - Server-side: no SQL parsing, direct access to the storage.
- Please, run your own benchmarks! Actual performance results are highly dependent on setup and hardware used.

22.9.10.2. Limitations

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The initial version is not binary safe. Due to the way the MySQL Memcache plugin works there are restrictions related to separators.

Prepared statements and asynchronous queries are not supported. Result set meta data support is limited.

The mapping information for tables accessible via Memcache is not cached in the plugin between requests but fetched from the MySQL server each time a MySQL connection is associated with a Memcache connection. See [mysqlnd_memcache_set](#) for details.

22.9.10.3. On the name

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The shortcut `mysqlnd_memcache` stands for `mysqlnd memcache plugin`. Memcache refers to support of the MySQL Memcache plugins for InnoDB and NDB (MySQL Cluster). The plugin is not related to the Memcached cache server.

22.9.10.4. Quickstart and Examples

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The `mysqlnd memcache` plugin is easy to use. This quickstart will demo typical use-cases, and provide practical advice on getting started.

It is strongly recommended to read the reference sections in addition to the quickstart. The quickstart tries to avoid discussing theoretical concepts and limitations. Instead, it will link to the reference sections. It is safe to begin with the quickstart. However, before using the plugin in mission critical environments we urge you to read additionally the background information from the reference sections.

22.9.10.4.1. Setup

[Copyright 1997-2012 the PHP Documentation Group. \[3186\]](#)

The plugin is implemented as a PHP extension. See also the [installation instructions](#) to install this extension.

Compile or configure the PHP MySQL extension (API) ([mysqli](#), [PDO_MYSQL](#), [mysql](#)). That extension must use the `mysqlnd` library as because `mysqlnd_memcache` is a plugin for the `mysqlnd` library. For additional information, refer to the [mysqlnd_memcache installation instructions](#).

Then, load this extension into PHP and activate the plugin in the PHP configuration file using the PHP configuration directive named `mysqlnd_memcache.enable`.

Example 22.377. Enabling the plugin (php.ini)

```
; On Windows the filename is php_mysqlnd_memcache.dll
; Load the extension
extension=mysqlnd_memcache.so
; Enable it
mysqlnd_memcache.enable=1
```

Follow the instructions given in the [MySQL Reference Manual on installing the Memcache plugins](#) for the MySQL server. Activate the plugins and configure Memcache access for SQL tables.

The examples in this quickguide assume that the following table exists, and that Memcache is configured with access to it.

Example 22.378. SQL table used for the Quickstart

```
CREATE TABLE test(
  id CHAR(16),
  f1 VARCHAR(255),
  f2 VARCHAR(255),
  f3 VARCHAR(255),
  flags INT NOT NULL,
  cas_column INT,
  expire_time_column INT,
  PRIMARY KEY(id)
) ENGINE=InnoDB;

INSERT INTO test (id, f1, f2, f3) VALUES (1, 'Hello', 'World', '!');
INSERT INTO test (id, f1, f2, f3) VALUES (2, 'Lady', 'and', 'the tramp');

INSERT INTO innodb_memcache.containers(
  name, db_schema, db_table, key_columns, value_columns,
  flags, cas_column, expire_time_column, unique_idx_name_on_key)
VALUES (
  'plugin_test', 'test', 'test', 'id', 'f1,f2,f3',
  'flags', 'cas_column', 'expire_time_column', 'PRIMARY KEY');
```

22.9.10.4.2. Usage

Copyright 1997-2012 the PHP Documentation Group. [3186]

After associating a MySQL connection with a Memcache connection using `mysqlnd_memcache_set` the plugin attempts to transparently replace SQL `SELECT` statements by a memcache access. For that purpose the plugin monitors all SQL statements executed and tries to match the statement string against `MYSQLND_MEMCACHE_DEFAULT_REGEX`. In case of a match, the `mysqlnd` memcache plugin checks whether the `SELECT` is accessing only columns of a mapped table and the `WHERE` clause is limited to a single key lookup.

In case of the example SQL table, the plugin will use the Memcache interface of the MySQL server to fetch results for a SQL query like `SELECT f1, f2, f3 WHERE id = n`.

Example 22.379. Basic example.

```
<?php
$mysqli = new mysqli("host", "user", "passwd", "database");
$memc = new Memcached();
$memc->addServer("host", 11211);
mysqlnd_memcache_set($mysqli, $memc);

/*
   This is a query which queries table test using id as key in the WHERE part
   and is accessing fields f1, f2 and f3. Therefore, mysqlnd_memcache
   will intercept it and route it via memcache.
*/
$result = $mysqli->query("SELECT f1, f2, f3 FROM test WHERE id = 1");
while ($row = $result->fetch_row()) {
    print_r($row);
}
```

```
}

/*
  This is a query which queries table test but using f1 in the WHERE clause.
  Therefore, mysqlnd_memcache can't intercept it. This will be executed
  using the MySQL protocol
*/
$mysqli->query("SELECT id FROM test WHERE f1 = 'Lady'");
while ($row = $result->fetch_row()) {
    print_r($row);
}
?>
```

The above example will output:

```
array(
  [f1] => Hello
  [f2] => World
  [f3] => !
)
array(
  [id] => 2
)
```

22.9.10.5. Installing/Configuring

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.10.5.1. Requirements

Copyright 1997-2012 the PHP Documentation Group. [3186]

PHP: this extension requires PHP 5.4+, version PHP 5.4.4 or never. The required PHP extensions are [PCRE](#) (enabled by default), and the [memcached](#) extension version 2.0.x.

The `mysqlnd_memcache` Memcache plugin supports all PHP applications and all available PHP MySQL extensions ([mysqli](#), [mysql](#), [PDO_MYSQL](#)). The PHP MySQL extension must be configured with [mysqlnd](#) support.

For accessing [InnoDB](#) tables, this PHP extension requires [MySQL Server 5.6.6](#) or newer with the InnoDB Memcache Daemon Plugin enabled.

For accessing [MySQL Cluster NDB](#) tables, this PHP extension requires [MySQL Cluster 7.2](#) or newer with the NDB Memcache API nodes enabled.

22.9.10.5.2. Installation

Copyright 1997-2012 the PHP Documentation Group. [3186]

This [PECL](#) extension is not bundled with PHP.

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: http://pecl.php.net/package/mysqlnd_memcache

A DLL for this PECL extension is currently unavailable. See also the [building on Windows](#) section.

22.9.10.5.3. Runtime Configuration

Copyright 1997-2012 the PHP Documentation Group. [3186]

The behaviour of these functions is affected by settings in `php.ini`.

Table 22.76. MySQLnd_memcache Configure Options

| Name | Default | Changeable | Changelog |
|--------------------------------------|----------------|-----------------------------|-----------------------|
| <code>mysqlnd_memcache.enable</code> | <code>1</code> | <code>PHP_INI_SYSTEM</code> | Available since 1.0.0 |

Here's a short explanation of the configuration directives.

`mysqlnd_memcache.enable` integer Enables or disables the plugin. If disabled, the extension will not plug into `mysqlnd` to proxy internal `mysqlnd` C API calls.



Note

This option is mainly used by developers to build this extension statically into PHP. General users are encouraged to build this extension as a shared object, and to unload it completely when it is not needed.

22.9.10.6. Predefined Constants

Copyright 1997-2012 the PHP Documentation Group. [3186]

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

MySQL Memcache Plugin related

`MYSQLND_MEMCACHE_DEFAULT_FILTER` Default regular expression (PCRE style) used for matching `SELECT` statements that will be mapped into a MySQL Memcache Plugin access point, if possible.

It is also possible to use `mysqlnd_memcache_set`, but the default approach is using this regular expression for pattern matching.

Assorted

The version number of this plugin can be obtained by using `MYSQLND_MEMCACHE_VERSION` or `MYSQLND_MEMCACHE_VERSION_ID`. `MYSQLND_MEMCACHE_VERSION` is the string representation of the numerical version number `MYSQLND_MEMCACHE_VERSION_ID`, which is an integer such as 10000. Developers can calculate the version number as follows.

| Version (part) | Example |
|--|-----------------|
| Major*10000 | 1*10000 = 10000 |
| Minor*100 | 0*100 = 0 |
| Patch | 0 = 0 |
| <code>MYSQLND_MEMCACHE_VERSION_ID</code> | 10000 |

`MYSQLND_MEMCACHE_VERSION` Plugin version string, for example, "1.0.0-alpha".
(string)

`MYSQLND_MEMCACHE_VERSION_ID` Plugin version number, for example, 10000.
(integer)

22.9.10.7. MySQLnd_memcache Functions

Copyright 1997-2012 the PHP Documentation Group. [3186]

22.9.10.7.1. mysqlnd_memcache_get_config

Copyright 1997-2012 the PHP Documentation Group. [3186]

- `mysqlnd_memcache_get_config`

Returns information about the plugin configuration

Description

```
array mysqlnd_memcache_get_config(
    mixed connection);
```

This function returns an array of all `mysqlnd_memcache` related configuration information that is attached to the MySQL connection. This includes MySQL, the Memcache object provided via `mysqlnd_memcache_set`, and the table mapping configuration that was automatically collected from the MySQL Server.

Parameters

connection A handle to a MySQL Server using one of the MySQL API extensions for PHP, which are `PDO_MYSQL`, `mysqli` or `ext/mysql`.

Return Values

An array of `mysqlnd_memcache` configuration information on success, otherwise `FALSE`.

The returned array has these elements:

Table 22.77. `mysqlnd_memcache_get_config` array structure

| Array Key | Description |
|-----------|--|
| memcached | Instance of Memcached associated to this MySQL connection by <code>mysqlnd_memcache_set</code> . You can use this to change settings of the memcache connection, or directly by querying the server on this connection. |
| pattern | The PCRE regular expression used to match the SQL query sent to the server. Queries matching this pattern will be further analyzed to decide whether the query can be intercepted and sent via the memcache interface or whether the query is sent using the general MySQL protocol to the server. The pattern is either the default pattern (<code>MYSQLND_MEMCACHE_DEFAULT_REGEXP</code>) or it is set via <code>mysqlnd_memcache_set</code> . |
| mappings | An associative array with a list of all configured containers as they were discovered by this plugin. The key for these elements is the name of the |

| Array Key | Description |
|----------------------------|--|
| | container in the MySQL configuration. The value is described below. The contents of this field is created by querying the MySQL Server during association to MySQL and a memcache connection using <code>mysqlnd_memcache_set</code> . |
| <code>mapping_query</code> | An SQL query used during <code>mysqlnd_memcache_set</code> to identify the available containers and mappings. The result of that query is provided in the mappings element. |

Table 22.78. Mapping entry structure

| Array Key | Description |
|----------------------------|---|
| <code>prefix</code> | A prefix used while accessing data via memcache. With the MySQL InnoDB Memcache Daemon plugin, this usually begins with @@ and ends with a configurable separator. This prefix is placed in front of the key value while using the memcache protocol. |
| <code>schema_name</code> | Name of the schema (database) which contains the table being accessed. |
| <code>table_name</code> | Name of the table which contains the data accessible via memcache protocol. |
| <code>id_field_name</code> | Name of the database field (column) with the id used as key when accessing the table via memcache. Often this is the database field having a primary key. |
| <code>separator</code> | <p>The separator used to split the different field values. This is needed as memcache only provides access to a single value while MySQL can map multiple columns to this value.</p> <div style="display: flex; align-items: flex-start;"> <div style="margin-right: 10px;">  </div> <div> <p>Note</p> <p>The separator, which can be set in the MySQL Server configuration, should not be part of any value retrieved via memcache because proper mapping can't be guaranteed.</p> </div> </div> |
| <code>fields</code> | An array with the name of all fields available for this mapping. |

Examples

Example 22.380. `mysqlnd_memcache_get_config` example

```
<?php
$mysqli = new mysqli("host", "user", "passwd", "database");
$memc = new Memcached();
$memc->addServer("host", 11211);
```

```
mysqlnd_memcache_set($mysqli, $memc);

var_dump(mysqlnd_memcache_get_config($mysqli));
?>
```

The above example will output:

```
array(4) {
  ["memcached"]=>
  object(Memcached)#2 (0) {
  }
  ["pattern"]=>
  string(125) "/^\s*SELECT\s*(.+?)\s*FROM\s*`?([a-z0-9_]+)`?\s*WHERE\s*`?([a-z0-9_]+)`?\s*=\s*(?=[''])['']([
  ["mappings"]=>
  array(1) {
    ["mymem_test"]=>
    array(6) {
      ["prefix"]=>
      string(13) "@@mymem_test."
      ["schema_name"]=>
      string(4) "test"
      ["table_name"]=>
      string(10) "mymem_test"
      ["id_field_name"]=>
      string(2) "id"
      ["separator"]=>
      string(1) "|"
      ["fields"]=>
      array(3) {
        [0]=>
        string(2) "f1"
        [1]=>
        string(2) "f2"
        [2]=>
        string(2) "f3"
      }
    }
  }
  ["mapping_query"]=>
  string(209) "      SELECT c.name,
              CONCAT('@@', c.name, (SELECT value FROM innodb_memcache.config_options WHERE name =
              c.db_schema,
              c.db_table,
              c.key_columns,
              c.value_columns,
              (SELECT value FROM innodb_memcache.config_options WHERE name = 'separator') AS sep
              FROM innodb_memcache.containers c"
```

See Also

[mysqlnd_memcache_set](#)

22.9.10.7.2. [mysqlnd_memcache_set](#)

Copyright 1997-2012 the PHP Documentation Group. [3186]

- [mysqlnd_memcache_set](#)

Associate a MySQL connection with a Memcache connection

Description

```
bool mysqlnd_memcache_set(
    mixed mysql_connection,
    Memcached memcache_connection,
    string pattern,
    callback callback);
```

Associate `mysql_connection` with `memcache_connection` using `pattern` as a PCRE regular expression, and `callback` as a notification callback or to unset the association of `mysql_connection`.

While associating a MySQL connection with a Memcache connection, this function will query the MySQL Server for its configuration. It will automatically detect whether the server is configured to use the InnoDB Memcache Daemon Plugin or MySQL Cluster NDB Memcache support. It will also query the server to automatically identify exported tables and other configuration options. The results of this automatic configuration can be retrieved using `mysqlnd_memcache_get_config`.

Parameters

| | |
|----------------------------------|--|
| <code>mysql_connection</code> | A handle to a MySQL Server using one of the MySQL API extensions for PHP, which are <code>PDO_MYSQL</code> , <code>mysqli</code> or <code>ext/mysqli</code> . |
| <code>memcache_connection</code> | A <code>Memcached</code> instance with a connection to the MySQL Memcache Daemon plugin. If this parameter is omitted, then <code>mysql_connection</code> will be unassociated from any memcache connection. And if a previous association exists, then it will be replaced. |
| <code>pattern</code> | A regular expression in <code>Perl Compatible Regular Expression</code> syntax used to identify potential Memcache-queries. The query should have three sub patterns. The first subpattern contains the requested field list, the second the name of the ID column from the query and the third the requested value. If this parameter is omitted or set to <code>NULL</code> , then a default pattern will be used. |
| <code>callback</code> | A callback which will be used whenever a query is being sent to MySQL. The callback will receive a single boolean parameter telling if a query was sent via Memcache. |

Return Values

`TRUE` if the association or disassociation is successful, otherwise `FALSE` if there is an error.

Examples

Example 22.381. `mysqlnd_memcache_set` example with `var_dump` as a simple debugging callback.

```
<?php
$mysqli = new mysqli("host", "user", "passwd", "database");
$memc = new Memcached();
$memc->addServer("host", 11211);
mysqlnd_memcache_set($mysqli, $memc, NULL, 'var_dump');

/* This query will be intercepted and executed via Memcache protocol */
echo "Sending query for id via Memcache: ";
$mysqli->query("SELECT f1, f2, f3 FROM test WHERE id = 1");

/* f1 is not configured as valid key field, this won't be sent via Memcache */
echo "Sending query for f1 via Memcache: ";
$mysqli->query("SELECT id FROM test WHERE f1 = 1");
```

```
mysqlnd_memcache_set($mysqli);

/* Now the regular MySQL protocol will be used */
echo "var_dump won't be invoked: ";
$mysqli->query("SELECT f1, f2, f3 WHERE id = 1");

?>
```

The above example will output:

```
Sending query for id via Memcache: bool(true)
Sending query for f1 via Memcache: bool(false)
var_dump won't be invoked:
```

See Also

[mysqlnd_memcache_get_config](#)

22.9.10.8. Change History

Copyright 1997-2012 the PHP Documentation Group. [3186]

This change history is a high level summary of selected changes that may impact applications and/or break backwards compatibility.

See also the [CHANGES](#) file in the source distribution for a complete list of changes.

22.9.10.8.1. PECL/mysqlnd_memcache 1.0 series

Copyright 1997-2012 the PHP Documentation Group. [3186]

1.0.0-alpha

- Release date: TBD
- Motto/theme: Basic mapping of SQL SELECT to a MySQL Memcache plugin access.

The initial release does map basic SQL SELECT statements to a MySQL Memcache plugin access. This bares potential performance benefits as the direct key-value access to MySQL storage using the Memcache protocol is usually faster than using SQL access.

22.9.11. Connector/PHP

This documentation, and other publications, sometimes uses the term [Connector/PHP](#). This term refers to the full set of MySQL related functionality in PHP, which includes the three APIs that are described above, along with the [mysqlnd](#) core library and all of its plugins.

22.9.12. Common Problems with MySQL and PHP

- [Error: Maximum Execution Time Exceeded](#): This is a PHP limit; go into the `php.ini` file and set the maximum execution time up from 30 seconds to something higher, as needed. It is also not a bad idea to double the RAM allowed per script to 16MB instead of 8MB.
- [Fatal error: Call to unsupported or undefined function mysql_connect\(\) in ...](#): This means that your PHP version isn't compiled with MySQL support. You can either compile

a dynamic MySQL module and load it into PHP or recompile PHP with built-in MySQL support. This process is described in detail in the PHP manual.

- **Error: Undefined reference to 'uncompress'**: This means that the client library is compiled with support for a compressed client/server protocol. The fix is to add `-lz` last when linking with `-lmysqlclient`.
- **Error: Client does not support authentication protocol**: This is most often encountered when trying to use the older `mysql` extension with MySQL 4.1.1 and later. Possible solutions are: downgrade to MySQL 4.0; switch to PHP 5 and the newer `mysqli` extension; or configure the MySQL server with the `old_passwords` [534] system variable set to 1. (See [Section C.5.2.4](#), “Client does not support authentication protocol”, for more information.)

22.9.13. Enabling Both `mysql` and `mysqli` in PHP

If you're experiencing problems with enabling both the `mysql` and the `mysqli` extension when building PHP on Linux yourself, you should try the following procedure.

1. Configure PHP like this:

```
./configure --with-mysqli=/usr/bin/mysql_config --with-mysql=/usr
```

2. Edit the `Makefile` and search for a line that starts with `EXTRA_LIBS`. It might look like this (all on one line):

```
EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl
-lxml2 -lz -lm -lxml2 -lz -lm -lmysqlclient -lz -lcrypt -lnsl -lm
-lxml2 -lz -lm -lcrypt -lxml2 -lz -lm -lcrypt
```

Remove all duplicates, so that the line looks like this (all on one line):

```
EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl
-lxml2
```

3. Build and install PHP:

```
make
make install
```

22.10. MySQL Perl API

The Perl `DBI` module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI with MySQL, install the following:

1. The `DBI` module.
2. The `DBD::mysql` module. This is the DataBase Driver (DBD) module for Perl.
3. Optionally, the DBD module for any other type of database server you want to access.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

These sections contain information about using Perl with MySQL and writing MySQL applications in Perl:

- For installation instructions for Perl DBI support, see [Section 2.14](#), “Perl Installation Notes”.
- For an example of reading options from option files, see [Section 5.3.4](#), “Using Client Programs in a Multiple-Server Environment”.

- For secure coding tips, see [Section 6.1.1, “Security Guidelines”](#).
- For debugging tips, see [Section 23.4.1.4, “Debugging `mysqld` under `gdb`”](#).
- For some Perl-specific environment variables, see [Section 2.13, “Environment Variables”](#).
- For considerations for running on Mac OS X, see [Section 2.4.5, “Using the Bundled MySQL on Mac OS X Server”](#).
- For ways to quote string literals, see [Section 9.1.1, “String Literals”](#).

DBI information is available at the command line, online, or in printed form:

- Once you have the `DBI` and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

```
shell> perldoc DBI
shell> perldoc DBI::FAQ
shell> perldoc DBD::mysql
```

You can also use `pod2man`, `pod2html`, and so on to translate this information into other formats.

- For online information about Perl DBI, visit the DBI Web site, <http://dbi.perl.org/>. That site hosts a general DBI mailing list. Oracle Corporation hosts a list specifically about `DBD::mysql`; see [Section 1.6.1, “MySQL Mailing Lists”](#).
- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI Web site, <http://dbi.perl.org/>.

For information that focuses specifically on using DBI with MySQL, see *MySQL and Perl for the Web* (Paul DuBois, New Riders, 2001). This book's Web site is <http://www.kitebird.com/mysql-perl/>.

22.11. MySQL Python API

`MySQLdb` is a third-party driver that provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at <http://sourceforge.net/projects/mysql-python/>.

The new MySQL Connector/Python component provides an interface to the same Python API, and is built into the MySQL Server and supported by Oracle. See [Section 22.6, “MySQL Connector/Python”](#) for details on the Connector, as well as coding guidelines for Python applications and sample Python code.

22.12. MySQL Ruby APIs

Two APIs are available for Ruby programmers developing MySQL applications:

- The MySQL/Ruby API is based on the `libmysql` API library. For information on installing and using the MySQL/Ruby API, see [Section 22.12.1, “The MySQL/Ruby API”](#).
- The Ruby/MySQL API is written to use the native MySQL network protocol (a native driver). For information on installing and using the Ruby/MySQL API, see [Section 22.12.2, “The Ruby/MySQL API”](#).

For background and syntax information about the Ruby language, see [Ruby Programming Language](#).

22.12.1. The MySQL/Ruby API

The MySQL/Ruby module provides access to MySQL databases using Ruby through `libmysql`.

For information on installing the module, and the functions exposed, see [MySQL/Ruby](#).

22.12.2. The Ruby/MySQL API

The Ruby/MySQL module provides access to MySQL databases using Ruby through a native driver interface using the MySQL network protocol.

For information on installing the module, and the functions exposed, see [Ruby/MySQL](#).

22.13. MySQL Tcl API

[MySQLtcl](#) is a simple API for accessing a MySQL database server from the [Tcl programming language](#). It can be found at <http://www.xdobry.de/mysqltcl/>.

22.14. MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the [Eiffel programming language](#), written by Michael Ravits. It can be found at <http://efsa.sourceforge.net/archive/ravits/mysql.htm>.

Chapter 23. Extending MySQL

Table of Contents

| | |
|---|------|
| 23.1. MySQL Internals | 3793 |
| 23.1.1. MySQL Threads | 3793 |
| 23.1.2. The MySQL Test Suite | 3794 |
| 23.2. The MySQL Plugin API | 3795 |
| 23.2.1. Plugin API Characteristics | 3795 |
| 23.2.2. Plugin API Components | 3796 |
| 23.2.3. Types of Plugins | 3797 |
| 23.2.4. Writing Plugins | 3800 |
| 23.2.5. MySQL Services for Plugins | 3840 |
| 23.3. Adding New Functions to MySQL | 3841 |
| 23.3.1. Features of the User-Defined Function Interface | 3842 |
| 23.3.2. Adding a New User-Defined Function | 3842 |
| 23.3.3. Adding a New Native Function | 3852 |
| 23.4. Debugging and Porting MySQL | 3854 |
| 23.4.1. Debugging a MySQL Server | 3854 |
| 23.4.2. Debugging a MySQL Client | 3861 |
| 23.4.3. The DEBUG Package | 3861 |

23.1. MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. To track or contribute to MySQL development, follow the instructions in [Section 2.10.3, “Installing MySQL from a Development Source Tree”](#). If you are interested in MySQL internals, you should also subscribe to our `internals` mailing list. This list has relatively low traffic. For details on how to subscribe, please see [Section 1.6.1, “MySQL Mailing Lists”](#). Many MySQL developers at Oracle Corporation are on the `internals` list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

23.1.1. MySQL Threads

The MySQL server creates the following threads:

- Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests. The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.
- Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

For information about tuning the parameters that control thread resources, see [Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”](#).

- On a master replication server, connections from slave servers are handled like client connections: There is one thread per connected slave.
- On a slave replication server, an I/O thread is started to connect to the master server and read updates from it. An SQL thread is started to apply updates read from the master. These two threads run independently and can be started and stopped independently.
- A signal thread handles all signals. This thread also normally handles alarms and calls `process_alarm()` to force timeouts on connections that have been idle too long.
- If `InnoDB` is used, there will be additional read and write threads by default. The number of these are controlled by the `innodb_read_io_threads` [1568] and `innodb_write_io_threads` [1575] parameters. See [Section 14.3.4, “InnoDB Startup Options and System Variables”](#).
- If `mysqld` is compiled with `-DUSE_ALARM_THREAD`, a dedicated thread that handles alarms is created. This is only used on some systems where there are problems with `sigwait()` or if you want to use the `thr_alarm()` code in your application without a dedicated signal handling thread.
- If the server is started with the `--flush_time=val` [500] option, a dedicated thread is created to flush all tables every `val` seconds.
- Each table for which `INSERT DELAYED` statements are issued gets its own thread. See [Section 13.2.5.2, “INSERT DELAYED Syntax”](#).
- If the event scheduler is active, there is one thread for the scheduler, and a thread for each event currently running. See [Section 19.4.1, “Event Scheduler Overview”](#).

`mysqladmin processlist` only shows the connection, `INSERT DELAYED`, replication, and event threads.

23.1.2. The MySQL Test Suite

The test system that is included in Unix source and binary distributions makes it possible for users and developers to perform regression tests on the MySQL code. These tests can be run on Unix.

You can also write your own test cases. For information about the MySQL Test Framework, including system requirements, see the manual available at <http://dev.mysql.com/doc/mysqltest/2.0/en/>.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, operating system or library issues, and is quite thorough in testing replication. Our goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system because this ensures that all future MySQL releases work well with your applications.

The test system consists of a test language interpreter (`mysqltest`), a Perl script to run all tests (`mysql-test-run.pl`), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type `make test` from the source root directory, or change location to the `mysql-test` directory and type `./mysql-test-run.pl`. If you have installed a binary distribution, change location to the `mysql-test` directory under the installation root directory (for example, `/usr/local/mysql/mysql-test`), and run `./mysql-test-run.pl`. All tests should succeed. If any do not, feel free to try to find out why and report the problem if it indicates a bug in MySQL. See [Section 1.7, “How to Report Bugs or Problems”](#).

If one test fails, you should run `mysql-test-run.pl` with the `--force` option to check whether any other tests fail.

If you have a copy of `mysqld` running on the machine where you want to run the test suite, you do not have to stop it, as long as it is not using ports `9306` or `9307`. If either of those ports is taken, you should

set the `MTR_BUILD_THREAD` environment variable to an appropriate value, and the test suite will use a different set of ports for master, slave, and NDB). For example:

```
shell> export MTR_BUILD_THREAD=31
shell> ./mysql-test-run.pl [options] [test_name]
```

In the `mysql-test` directory, you can run an individual test case with `./mysql-test-run.pl test_name`.

If you have a question about the test suite, or have a test case to contribute, send an email message to the MySQL `internals` mailing list. See [Section 1.6.1, “MySQL Mailing Lists”](#).

23.2. The MySQL Plugin API

MySQL supports a plugin API that enables creation of server components. Plugins can be loaded at server startup, or loaded and unloaded at runtime without restarting the server. The API is generic and does not specify what plugins can do. The components supported by this interface include, but are not limited to, storage engines, full-text parser plugins, and server extensions.

For example, full-text parser plugins can be used to replace or augment the built-in full-text parser. A plugin can parse text into words using rules that differ from those used by the built-in parser. This can be useful if you need to parse text with characteristics different from those expected by the built-in parser.

The plugin interface is more general than the older user-defined function (UDF) interface.

The plugin interface uses the `plugin` table in the `mysql` database to record information about plugins that have been installed permanently with the `INSTALL PLUGIN` statement. This table is created as part of the MySQL installation process. Plugins can also be installed for a single server invocation with the `--plugin-load` option. Plugins installed this way are not recorded in the `plugin` table. See [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

MySQL 5.5.7 and up supports an API for client plugins in addition to that for server plugins. This is used, for example, by authentication plugins where a server-side plugin and a client-side plugin cooperate to enable clients to connect to the server through a variety of authentication methods.

Additional Resources

The book *MySQL 5.1 Plugin Development* by Sergei Golubchik and Andrew Hutchings provides a wealth of detail about the plugin API. Despite the fact that the book’s title refers to MySQL Server 5.1, most of the information in it applies to later versions as well.

23.2.1. Plugin API Characteristics

The server plugin API has these characteristics:

- All plugins have several things in common.

Each plugin has a name that it can be referred to in SQL statements, as well as other metadata such as an author and a description that provide other information. This information can be examined in the `INFORMATION_SCHEMA.PLUGINS` table or using the `SHOW PLUGINS` statement.

- The plugin framework is extendable to accommodate different kinds of plugins.

Although some aspects of the plugin API are common to all types of plugins, the API also permits type-specific interface elements so that different types of plugins can be created. A plugin with one purpose can have an interface most appropriate to its own requirements and not the requirements of some other plugin type.

Interfaces for several types of plugins exist, such as storage engines, full-text parser, and `INFORMATION_SCHEMA` tables. Others can be added.

- Plugins can expose information to users.

A plugin can implement system and status variables that are available through the `SHOW VARIABLES` and `SHOW STATUS` statements.

- The plugin API includes versioning information.

The version information included in the plugin API enables a plugin library and each plugin that it contains to be self-identifying with respect to the API version that was used to build the library. If the API changes over time, the version numbers will change, but a server can examine a given plugin library's version information to determine whether it supports the plugins in the library.

There are two types of version numbers. The first is the version for the general plugin framework itself. Each plugin library includes this kind of version number. The second type of version applies to individual plugins. Each specific type of plugin has a version for its interface, so each plugin in a library has a type-specific version number. For example, a library containing a full-text parser plugin has a general plugin API version number, and the plugin has a version number specific to the full-text plugin interface.

- The plugin API implements security restrictions.

A plugin library must be installed in a specific dedicated directory for which the location is controlled by the server and cannot be changed at runtime. Also, the library must contain specific symbols that identify it as a plugin library. The server will not load something as a plugin if it was not built as a plugin.

- Plugins have access to server services.

The services interface exposes server functionality that plugins can access using ordinary function calls. For details, see [Section 23.2.5, “MySQL Services for Plugins”](#).

In some respects, the server plugin API is similar to the older user-defined function (UDF) API that it supersedes, but the plugin API has several advantages over the older interface. For example, UDFs had no versioning information. Also, the newer plugin interface eliminates the security issues of the older UDF interface. The older interface for writing nonplugin UDFs permitted libraries to be loaded from any directory searched by the system's dynamic linker, and the symbols that identified the UDF library were relatively nonspecific.

The client plugin API has similar architectural characteristics, but client plugins have no direct access to the server the way server plugins do.

23.2.2. Plugin API Components

The server plugin implementation comprises several components.

SQL statements:

- `INSTALL PLUGIN` registers a plugin in the `mysql.plugin` table and loads the plugin code.
- `UNINSTALL PLUGIN` unregisters a plugin from the `mysql.plugin` table and unloads the plugin code.
- The `WITH PARSER` clause for full-text index creation associates a full-text parser plugin with a given `FULLTEXT` index.
- `SHOW PLUGINS` displays information about server plugins.

Command-line options and system variables:

- The `--plugin-load` [457] option enables plugins to be loaded at server startup time.
- The `plugin_dir` [537] system variable indicates the location of the directory where all plugins must be installed. The value of this variable can be specified at server startup with a `--plugin_dir=path` [537] option. `mysql_config --plugindir` displays the default plugin directory path name.

For additional information about plugin loading, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

Plugin-related tables:

- The `INFORMATION_SCHEMA.PLUGINS` table contains plugin information.
- The `mysql.plugin` table lists each plugin that was installed with `INSTALL PLUGIN` and is required for plugin use. For new MySQL installations, this table is created during the installation process.

The client plugin implementation is simpler:

- For the `mysql_options()` C API function, the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options enable client programs to load authentication plugins.
- There are C API functions that enable management of client plugins.

To examine how MySQL implements plugins, consult the following source files in a MySQL source distribution:

- In the `include/mysql` directory, `plugin.h` exposes the public plugin API. This file should be examined by anyone who wants to write a plugin library. `plugin_XXX.h` files provide additional information that pertains to specific types of plugins. `client_plugin.h` contains information specific to client plugins.
- In the `sql` directory, `sql_plugin.h` and `sql_plugin.cc` comprise the internal plugin implementation. `sql_acl.cc` is where the server uses authentication plugins. These files need not be consulted by plugin developers. They may be of interest for those who want to know more about how the server handles plugins.
- In the `sql-common` directory, `client_plugin.h` implements the C API client plugin functions, and `client.c` implements client authentication support. These files need not be consulted by plugin developers. They may be of interest for those who want to know more about how the server handles plugins.

23.2.3. Types of Plugins

The plugin API enables creation of plugins that implement several capabilities:

- Storage engines
- Full-text parsers
- Daemons
- `INFORMATION_SCHEMA` tables
- Semisynchronous replication
- Auditing

- Authentication

The following sections provide an overview of these plugin types.

23.2.3.1. Storage Engine Plugins

The pluggable storage engine architecture used by MySQL Server enables storage engines to be written as plugins and loaded into and unloaded from a running server. For a description of this architecture, see [Section 14.2, “Overview of MySQL Storage Engine Architecture”](#).

For information on how to use the plugin API to write storage engines, see [MySQL Internals: Writing a Custom Storage Engine](#).

23.2.3.2. Full-Text Parser Plugins

MySQL has a built-in parser that it uses by default for full-text operations (parsing text to be indexed, or parsing a query string to determine the terms to be used for a search). For full-text processing, “parsing” means extracting words from text or a query string based on rules that define which character sequences make up a word and where word boundaries lie.

When parsing for indexing purposes, the parser passes each word to the server, which adds it to a full-text index. When parsing a query string, the parser passes each word to the server, which accumulates the words for use in a search.

The parsing properties of the built-in full-text parser are described in [Section 12.9, “Full-Text Search Functions”](#). These properties include rules for determining how to extract words from text. The parser is influenced by certain system variables such as `ft_min_word_len` [501] and `ft_max_word_len` [501] that cause words shorter or longer to be excluded, and by the stopword list that identifies common words to be ignored.

The plugin API enables you to provide a full-text parser of your own so that you have control over the basic duties of a parser. A parser plugin can operate in either of two roles:

- The plugin can replace the built-in parser. In this role, the plugin reads the input to be parsed, splits it up into words, and passes the words to the server (either for indexing or for word accumulation).

One reason to use a parser this way is that you need to use different rules from those of the built-in parser for determining how to split up input into words. For example, the built-in parser considers the text “case-sensitive” to consist of two words “case” and “sensitive,” whereas an application might need to treat the text as a single word.

- The plugin can act in conjunction with the built-in parser by serving as a front end for it. In this role, the plugin extracts text from the input and passes the text to the parser, which splits up the text into words using its normal parsing rules. In particular, this parsing will be affected by the `ft_xxx` system variables and the stopword list.

One reason to use a parser this way is that you need to index content such as PDF documents, XML documents, or `.doc` files. The built-in parser is not intended for those types of input but a plugin can pull out the text from these input sources and pass it to the built-in parser.

It is also possible for a parser plugin to operate in both roles. That is, it could extract text from nonplaintext input (the front end role), and also parse the text into words (thus replacing the built-in parser).

A full-text plugin is associated with full-text indexes on a per-index basis. That is, when you install a parser plugin initially, that does not cause it to be used for any full-text operations. It simply becomes available. For example, a full-text parser plugin becomes available to be named in a `WITH PARSER` clause when creating individual `FULLTEXT` indexes. To create such an index at table-creation time, do this:

```
CREATE TABLE t
(
  doc CHAR(255),
  FULLTEXT INDEX (doc) WITH PARSER my_parser
) ENGINE=MyISAM;
```

Or you can add the index after the table has been created:

```
ALTER TABLE t ADD FULLTEXT INDEX (doc) WITH PARSER my_parser;
```

The only SQL change for associating the parser with the index is the `WITH PARSER` clause. Searches are specified as before, with no changes needed for queries.

When you associate a parser plugin with a `FULLTEXT` index, the plugin is required for using the index. If the parser plugin is dropped, any index associated with it becomes unusable. Any attempt to use it a table for which a plugin is not available results in an error, although `DROP TABLE` is still possible.

For more information about full-text plugins, see [Section 23.2.4.4, “Writing Full-Text Parser Plugins”](#).

23.2.3.3. Daemon Plugins

A daemon plugin is a simple type of plugin used for code that should be run by the server but that does not communicate with it. MySQL distributions include an example daemon plugin that writes periodic heartbeat messages to a file.

For more information about daemon plugins, see [Section 23.2.4.5, “Writing Daemon Plugins”](#).

23.2.3.4. `INFORMATION_SCHEMA` Plugins

`INFORMATION_SCHEMA` plugins enable the creation of tables containing server metadata that are exposed to users through the `INFORMATION_SCHEMA` database. For example, `InnoDB` uses `INFORMATION_SCHEMA` plugins to provide tables that contain information about current transactions and locks.

For more information about `INFORMATION_SCHEMA` plugins, see [Section 23.2.4.6, “Writing `INFORMATION_SCHEMA` Plugins”](#).

23.2.3.5. Semisynchronous Replication Plugins

MySQL replication is asynchronous by default. With semisynchronous replication, a commit performed on the master side blocks before returning to the session that performed the transaction until at least one slave acknowledges that it has received and logged the events for the transaction. Semisynchronous replication is implemented through complementary master and client plugins. See [Section 16.3.8, “Semisynchronous Replication”](#).

For more information about semisynchronous replication plugins, see [Section 23.2.4.7, “Writing Semisynchronous Replication Plugins”](#).

23.2.3.6. Audit Plugins

As of MySQL 5.5.3, the server provides a pluggable audit interface that enables information about server operations to be reported to interested parties. Currently, audit notification occurs for these operations (although the interface is general and the server could be modified to report others):

- Write a message to the general query log (if the log is enabled)
- Write a message to the error log

- Send a query result to a client

Audit plugins may register with the audit interface to receive notification about server operations. When an auditable event occurs within the server, the server determines whether notification is needed. For each registered audit plugin, the server checks the event against those event classes in which the plugin is interested and passes the event to the plugin if there is a match.

This interface enables audit plugins to receive notifications only about operations in event classes they consider significant and to ignore others. The interface provides for categorization of operations into event classes and further division into event subclasses within each class.

When an audit plugin is notified of an auditable event, it receives a pointer to the current THD structure and a pointer to a structure that contains information about the event. The plugin can examine the event and perform whatever auditing actions are appropriate. For example, the plugin can see what statement produced a result set or was logged, the number of rows in a result, who the current user was for an operation, or the error code for failed operations.

For more information about audit plugins, see [Section 23.2.4.8, “Writing Audit Plugins”](#).

23.2.3.7. Authentication Plugins

MySQL 5.5.7 and up supports pluggable authentication. Authentication plugins exist on both the server and client sides. Plugins on the server side implement authentication methods for use by clients when they connect to the server. A plugin on the client side communicates with a server-side plugin to provide the authentication information that it requires. A client-side plugin may interact with the user, performing tasks such as soliciting a password or other authentication credentials to be sent to the server. See [Section 6.3.6, “Pluggable Authentication”](#).

Pluggable authentication also enables proxy user capability, in which one user takes the identity of another user. A server-side authentication plugin can return to the server the name of the user whose identity the connecting user should have. See [Section 6.3.7, “Proxy Users”](#).

For more information about authentication plugins, see [Section 23.2.4.9, “Writing Authentication Plugins”](#).

23.2.4. Writing Plugins

To create a plugin library, you must provide the required descriptor information that indicates what plugins the library file contains, and write the interface functions for each plugin.

Every server plugin must have a general descriptor that provides information to the plugin API, and a type-specific descriptor that provides information about the plugin interface for a given type of plugin. The structure of the general descriptor is the same for all plugin types. The structure of the type-specific descriptor varies among plugin types and is determined by the requirements of what the plugin needs to do. The server plugin interface also enables plugins to expose status and system variables. These variables become visible through the `SHOW STATUS` and `SHOW VARIABLES` statements and the corresponding `INFORMATION_SCHEMA` tables.

For client-side plugins, the architecture is a bit different. Each plugin must have a descriptor, but there is no division into separate general and type-specific descriptors. Instead, the descriptor begins with a fixed set of members common to all client plugin types, and the common members are followed by any additional members required to implement the specific plugin type.

You can write plugins in C or C++ (or another language that can use C calling conventions). Plugins are loaded and unloaded dynamically, so your operating system must support dynamic loading and you must have compiled the calling application dynamically (not statically). For server plugins, this means that `mysqld` must be compiled dynamically.

A server plugin contains code that becomes part of the running server, so when you write the plugin, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to plugins originally written for older servers. For information about these constraints, see [Section 2.10.4, “MySQL Source-Configuration Options”](#), and [Section 2.10.5, “Dealing with Problems Compiling MySQL”](#).

Client plugin writers should avoid dependencies on what symbols the calling application has because you cannot be sure what applications will use the plugin.

23.2.4.1. Overview of Plugin Writing

The following procedure provides an overview of the steps needed to create a plugin library. The next sections provide additional details on setting plugin data structures and writing specific types of plugins.

1. In the plugin source file, include the header files that the plugin library needs. The `plugin.h` file is required, and the library might require other files as well. For example:

```
#include <stdlib.h>
#include <ctype.h>
#include <mysql/plugin.h>
```

2. Set up the descriptor information for the plugin library file. For server plugins, write the library descriptor, which must contain the general plugin descriptor for each server plugin in the file. For more information, see [Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”](#). In addition, set up the type-specific descriptor for each server plugin in the library. Each plugin's general descriptor points to its type-specific descriptor.

For client plugins, write the client descriptor. For more information, see [Section 23.2.4.2.3, “Client Plugin Descriptors”](#).

3. Write the plugin interface functions for each plugin. For example, each plugin's general plugin descriptor points to the initialization and deinitialization functions that the server should invoke when it loads and unloads the plugin. The plugin's type-specific description may also point to interface functions.
4. For server plugins, set up the status and system variables, if there are any.
5. Compile the plugin library as a shared library and install it in the plugin directory. For more information, see [Section 23.2.4.3, “Compiling and Installing Plugin Libraries”](#).
6. For server plugins, register the plugin with the server. For more information, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).
7. Test the plugin to verify that it works properly.

23.2.4.2. Plugin Data Structures

A plugin library file includes descriptor information to indicate what plugins it contains.

If the plugin library contains any server plugins, it must include the following descriptor information:

- A library descriptor indicates the general server plugin API version number used by the library and contains a general plugin descriptor for each server plugin in the library. To provide the framework for this descriptor, invoke two macros from the `plugin.h` header file:

```
mysql_declare_plugin(name)
```

```
... one or more server plugin descriptors here ...
mysql_declare_plugin_end;
```

The macros expand to provide a declaration for the API version automatically. You must provide the plugin descriptors.

- Within the library descriptor, each general server plugin is described by a `st_mysql_plugin` structure. This plugin descriptor structure contains information that is common to every type of server plugin: A value that indicates the plugin type; the plugin name, author, description, and license type; pointers to the initialization and deinitialization functions that the server invokes when it loads and unloads the plugin, and pointers to any status or system variables the plugin implements.
- Each general server plugin descriptor within the library descriptor also contains a pointer to a type-specific plugin descriptor. The structure of the type-specific descriptors varies from one plugin type to another because each type of plugin can have its own API. A type-specific plugin descriptor contains a type-specific API version number and pointers to the functions that are needed to implement that plugin type. For example, a full-text parser plugin has initialization and deinitialization functions, and a main parsing function. The server invokes these functions when it uses the plugin to parse text.

The plugin library also contains the interface functions that are referenced by the general and type-specific descriptors for each plugin in the library.

If the plugin library contains a client plugin, it must include a descriptor for the plugin. The descriptor begins with a fixed set of members common to all client plugins, followed by any members specific to the plugin type. To provide the descriptor framework, invoke two macros from the `client_plugin.h` header file:

```
mysql_declare_client_plugin(plugin_type)
... members common to all client plugins ...
... type-specific extra members ...
mysql_end_client_plugin;
```

The plugin library also contains any interface functions referenced by the client descriptor.

The `mysql_declare_plugin()` and `mysql_declare_client_plugin()` macros differ somewhat in how they can be invoked, which has implications for the contents of plugin libraries. The following guidelines summarize the rules:

- `mysql_declare_plugin()` and `mysql_declare_client_plugin()` can both be used in the same source file, which means that a plugin library can contain both server and client plugins. However, each of `mysql_declare_plugin()` and `mysql_declare_client_plugin()` can be used at most once.
- `mysql_declare_plugin()` permits multiple server plugin declarations, so a plugin library can contain multiple server plugins.
- `mysql_declare_client_plugin()` permits only a single client plugin declaration. To create multiple client plugins, separate plugin libraries must be used.

When a client program looks for a client plugin that is in a plugin library and not built into `libmysql`, it looks for a file with a basename that is the same as the plugin name. For example, if a program needs to use a client authentication plugin named `auth_xxx` on a system that uses `.so` as the library extension, it looks in the file named `auth_xxx.so`. (On Mac OS X, the program looks first for `auth_xxx.dylib`, then for `auth_xxx.so`.) For this reason, if a plugin library contains a client plugin, the library must have the same basename as that plugin.

The same is not true for a library that contains server plugins. The `--plugin-load [457]` option and the `INSTALL PLUGIN` statement provide the library file name explicitly, so there need be no explicit relationship between the library name and the name of any server plugins it contains.

23.2.4.2.1. Server Plugin Library and Plugin Descriptors

Every plugin library that contains server plugins must include a library descriptor that contains the general plugin descriptor for each server plugin in the file. This section discusses how to write the library and general descriptors for server plugins.

The library descriptor must define two symbols:

- `_mysql_plugin_interface_version` specifies the version number of the general plugin framework. This is given by the `MYSQL_PLUGIN_INTERFACE_VERSION` symbol, which is defined in the `plugin.h` file.
- `_mysql_plugin_declarations` defines an array of plugin declarations, terminated by a declaration with all members set to 0. Each declaration is an instance of the `st_mysql_plugin` structure (also defined in `plugin.h`). There must be one of these for each server plugin in the library.

If the server does not find those two symbols in a library, it does not accept it as a legal plugin library and rejects it with an error. This prevents use of a library for plugin purposes unless it was built specifically as a plugin library.

The conventional way to define the two required symbols is by using the `mysql_declare_plugin()` and `mysql_declare_plugin_end` macros from the `plugin.h` file:

```
mysql_declare_plugin(name)
... one or more server plugin descriptors here ...
mysql_declare_plugin_end;
```

Each server plugin must have a general descriptor that provides information to the server plugin API. The general descriptor has the same structure for all plugin types. The `st_mysql_plugin` structure in the `plugin.h` file defines this descriptor:

```
struct st_mysql_plugin
{
    int type; /* the plugin type (a MYSQL_XXX_PLUGIN value) */
    void *info; /* pointer to type-specific plugin descriptor */
    const char *name; /* plugin name */
    const char *author; /* plugin author (for I_S.PLUGINS) */
    const char *descr; /* general descriptive text (for I_S.PLUGINS) */
    int license; /* the plugin license (PLUGIN_LICENSE_XXX) */
    int (*init)(void *); /* the function to invoke when plugin is loaded */
    int (*deinit)(void *); /* the function to invoke when plugin is unloaded */
    unsigned int version; /* plugin version (for I_S.PLUGINS) */
    struct st_mysql_show_var *status_vars;
    struct st_mysql_sys_var **system_vars;
    void * __reserved1; /* reserved for dependency checking */
    unsigned long flags; /* flags for plugin */
};
```

The `st_mysql_plugin` descriptor structure members are used as follows. `char *` members should be specified as null-terminated strings.

- `type`: The plugin type. This must be one of the plugin-type values from `plugin.h`:

```
/*
 * The allowable types of plugins
 */
#define MYSQL_UDF_PLUGIN 0 /* User-defined function */
#define MYSQL_STORAGE_ENGINE_PLUGIN 1 /* Storage Engine */
#define MYSQL_FTPARSER_PLUGIN 2 /* Full-text parser plugin */
#define MYSQL_DAEMON_PLUGIN 3 /* The daemon/raw plugin type */
#define MYSQL_INFORMATION_SCHEMA_PLUGIN 4 /* The I_S plugin type */
```

```
#define MYSQL_AUDIT_PLUGIN          5 /* The Audit plugin type */
#define MYSQL_REPLICATION_PLUGIN    6 /* The replication plugin type */
#define MYSQL_AUTHENTICATION_PLUGIN 7 /* The authentication plugin type */
...
```

For example, for a full-text parser plugin, the `type` value is `MYSQL_FTPARSER_PLUGIN`.

- **info:** A pointer to the type-specific descriptor for the plugin. This descriptor's structure depends on the particular type of plugin, unlike that of the general plugin descriptor structure. For version-control purposes, the first member of the type-specific descriptor for every plugin type is expected to be the interface version for the type. This enables the server to check the type-specific version for every plugin no matter its type. Following the version number, the descriptor includes any other members needed, such as callback functions and other information needed by the server to invoke the plugin properly. Later sections on writing particular types of server plugins describe the structure of their type-specific descriptors.
- **name:** A string that gives the plugin name. This is the name that will be listed in the `mysql.plugin` table and by which you refer to the plugin in SQL statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`, or with the `--plugin-load` [457] option. The name is also visible in the `INFORMATION_SCHEMA.PLUGINS` table or the output from `SHOW PLUGINS`.

The plugin name should not begin with the name of any server option. If it does, the server will fail to initialize it. For example, the server has a `--socket` [464] option, so you should not use a plugin name such as `socket`, `socket_plugin`, and so forth.

- **author:** A string naming the plugin author. This can be whatever you like.
- **desc:** A string that provides a general description of the plugin. This can be whatever you like.
- **license:** The plugin license type. The value can be one of `PLUGIN_LICENSE_PROPRIETARY`, `PLUGIN_LICENSE_GPL`, or `PLUGIN_LICENSE_BSD`.
- **init:** A once-only initialization function, or `NULL` if there is no such function. The server executes this function when it loads the plugin, which happens for `INSTALL PLUGIN` or, for plugins listed in the `mysql.plugin` table, at server startup. The function takes one argument that points to the internal structure used to identify the plugin. It returns zero for success and nonzero for failure.
- **deinit:** A once-only deinitialization function, or `NULL` if there is no such function. The server executes this function when it unloads the plugin, which happens for `UNINSTALL PLUGIN` or, for plugins listed in the `mysql.plugin` table, at server shutdown. The function takes one argument that points to the internal structure used to identify the plugin. It returns zero for success and nonzero for failure.
- **version:** The plugin version number. When the plugin is installed, this value can be retrieved from the `INFORMATION_SCHEMA.PLUGINS` table. The value includes major and minor numbers. If you write the value as a hex constant, the format is `0xMMNN`, where `MM` and `NN` are the major and minor numbers, respectively. For example, `0x0302` represents version 3.2.
- **status_vars:** A pointer to a structure for status variables associated with the plugin, or `NULL` if there are no such variables. When the plugin is installed, these variables are displayed in the output of the `SHOW STATUS` statement.

The `status_vars` member, if not `NULL`, points to an array of `st_mysql_show_var` structures that describe status variables. See Section 23.2.4.2.2, “Server Plugin Status and System Variables”.

- **system_vars:** A pointer to a structure for system variables associated with the plugin, or `NULL` if there are no such variables. These options and system variables can be used to help initialize variables within the plugin.

The `system_vars` member, if not `NULL`, points to an array of `st_mysql_sys_var` structures that describe system variables. See [Section 23.2.4.2.2, “Server Plugin Status and System Variables”](#).

- `__reserved1`: A placeholder for the future. Currently, it should be set to `NULL`.
- `flags`: Plugin flags. Individual bits correspond to different flags. The value should be set to the OR of the applicable flags. These flags are available:

```
#define PLUGIN_OPT_NO_INSTALL 1UL /* Not dynamically loadable */
#define PLUGIN_OPT_NO_UNINSTALL 2UL /* Not dynamically unloadable */
```

`PLUGIN_OPT_NO_INSTALL` indicates that the plugin cannot be loaded at runtime with the `INSTALL PLUGIN` statement. This is appropriate for plugins that must be loaded at server startup with the `--plugin-load [457]` option. `PLUGIN_OPT_NO_UNINSTALL` indicates that the plugin cannot be unloaded at runtime with the `UNINSTALL PLUGIN` statement.

This member was added in MySQL 5.5.16.

The server invokes the `init` and `deinit` functions in the general plugin descriptor only when loading and unloading the plugin. They have nothing to do with use of the plugin such as happens when an SQL statement causes the plugin to be invoked.

For example, the descriptor information for a library that contains a single full-text parser plugin named `simple_parser` looks like this:

```
mysql_declare_plugin(ftexample)
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    "simple_parser", /* name */
    "Oracle Corporation", /* author */
    "Simple Full-Text Parser", /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status, /* status variables */
    simple_system_variables, /* system variables */
    NULL,
    0
}
mysql_declare_plugin_end;
```

For a full-text parser plugin, the type must be `MYSQL_FTPARSER_PLUGIN`. This is the value that identifies the plugin as being legal for use in a `WITH PARSER` clause when creating a `FULLTEXT` index. (No other plugin type is legal for this clause.)

`plugin.h` defines the `mysql_declare_plugin()` and `mysql_declare_plugin_end` macros like this:

```
#ifndef MYSQL_DYNAMIC_PLUGIN
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
MYSQL_PLUGIN_EXPORT int VERSION= MYSQL_PLUGIN_INTERFACE_VERSION; \
MYSQL_PLUGIN_EXPORT int PSIZE= sizeof(struct st_mysql_plugin); \
MYSQL_PLUGIN_EXPORT struct st_mysql_plugin DECLS[]= {
#else
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
MYSQL_PLUGIN_EXPORT int _mysql_plugin_interface_version= MYSQL_PLUGIN_INTERFACE_VERSION; \
MYSQL_PLUGIN_EXPORT int _mysql_sizeof_struct_st_plugin= sizeof(struct st_mysql_plugin); \
MYSQL_PLUGIN_EXPORT struct st_mysql_plugin _mysql_plugin_declarations[]= {
#endif
```

```
#define mysql_declare_plugin(NAME) \
_MYSQL_DECLARE_PLUGIN(NAME, \
    builtin_ ## NAME ## _plugin_interface_version, \
    builtin_ ## NAME ## _sizeof_struct_st_plugin, \
    builtin_ ## NAME ## _plugin)

#define mysql_declare_plugin_end ,{0,0,0,0,0,0,0,0,0,0,0,0}
```



Note

Those declarations define the `_mysql_plugin_interface_version` symbol only if the `MYSQL_DYNAMIC_PLUGIN` symbol is defined. This means that `-DMYSQL_DYNAMIC_PLUGIN` must be provided as part of the compilation command to build the plugin as a shared library.

When the macros are used as just shown, they expand to the following code, which defines both of the required symbols (`_mysql_plugin_interface_version` and `_mysql_plugin_declarations`):

```
int _mysql_plugin_interface_version= MYSQL_PLUGIN_INTERFACE_VERSION;
int _mysql_sizeof_struct_st_plugin= sizeof(struct st_mysql_plugin);
struct st_mysql_plugin _mysql_plugin_declarations[]= {
{
    MYSQL_FTPARSER_PLUGIN,          /* type */
    &simple_parser_descriptor,      /* descriptor */
    "simple_parser",                /* name */
    "Oracle Corporation",         /* author */
    "Simple Full-Text Parser",     /* description */
    PLUGIN_LICENSE_GPL,           /* plugin license */
    simple_parser_plugin_init,     /* init function (when loaded) */
    simple_parser_plugin_deinit,  /* deinit function (when unloaded) */
    0x0001,                        /* version */
    simple_status,                 /* status variables */
    simple_system_variables,       /* system variables */
    NULL,
    0
}
, {0,0,0,0,0,0,0,0,0,0,0,0}
};
```

The preceding example declares a single plugin in the general descriptor, but it is possible to declare multiple plugins. List the declarations one after the other between `mysql_declare_plugin()` and `mysql_declare_plugin_end`, separated by commas.

MySQL server plugins can be written in C or C++ (or another language that can use C calling conventions). If you write a C++ plugin, one C++ feature that you should not use is nonconstant variables to initialize global structures. Members of structures such as the `st_mysql_plugin` structure should be initialized only with constant variables. The `simple_parser` descriptor shown earlier is permissible in a C++ plugin because it satisfies that requirement:

```
mysql_declare_plugin(ftexample)
{
    MYSQL_FTPARSER_PLUGIN,          /* type */
    &simple_parser_descriptor,      /* descriptor */
    "simple_parser",                /* name */
    "Oracle Corporation",         /* author */
    "Simple Full-Text Parser",     /* description */
    PLUGIN_LICENSE_GPL,           /* plugin license */
    simple_parser_plugin_init,     /* init function (when loaded) */
    simple_parser_plugin_deinit,  /* deinit function (when unloaded) */
    0x0001,                        /* version */
    simple_status,                 /* status variables */
    simple_system_variables,       /* system variables */
    NULL,
    0
}
```

```

}
mysql_declare_plugin_end;

```

Here is another valid way to write the general descriptor. It uses constant variables to indicate the plugin name, author, and description:

```

const char *simple_parser_name = "simple_parser";
const char *simple_parser_author = "Oracle Corporation";
const char *simple_parser_description = "Simple Full-Text Parser";

mysql_declare_plugin(ftexample)
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    simple_parser_name, /* name */
    simple_parser_author, /* author */
    simple_parser_description, /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status, /* status variables */
    simple_system_variables, /* system variables */
    NULL,
    0
}
mysql_declare_plugin_end;

```

However, the following general descriptor is invalid. It uses structure members to indicate the plugin name, author, and description, but structures are not considered constant initializers in C++:

```

typedef struct
{
    const char *name;
    const char *author;
    const char *description;
} plugin_info;

plugin_info parser_info = {
    "simple_parser",
    "Oracle Corporation",
    "Simple Full-Text Parser"
};

mysql_declare_plugin(ftexample)
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    parser_info.name, /* name */
    parser_info.author, /* author */
    parser_info.description, /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status, /* status variables */
    simple_system_variables, /* system variables */
    NULL,
    0
}
mysql_declare_plugin_end;

```

23.2.4.2.2. Server Plugin Status and System Variables

The server plugin interface enables plugins to expose status and system variables using the `status_vars` and `system_vars` members of the general plugin descriptor.

The `status_vars` member of the general plugin descriptor, if not 0, points to an array of `st_mysql_show_var` structures, each of which describes one status variable, followed by a structure with all members set to 0. The `st_mysql_show_var` structure has this definition:

```
struct st_mysql_show_var {
    const char *name;
    char *value;
    enum enum_mysql_show_type type;
};
```

When the plugin is installed, the plugin name and the `name` value are joined with an underscore to form the name displayed by `SHOW STATUS`.

The following table shows the permissible status variable `type` values and what the corresponding variable should be.

Table 23.1. Server Plugin Status Variable Types

| Variable Type | Meaning |
|----------------------------|---|
| <code>SHOW_BOOL</code> | Pointer to a boolean variable |
| <code>SHOW_INT</code> | Pointer to an integer variable |
| <code>SHOW_LONG</code> | Pointer to a long integer variable |
| <code>SHOW_LONGLONG</code> | Pointer to a longlong integer variable |
| <code>SHOW_CHAR</code> | A string |
| <code>SHOW_CHAR_PTR</code> | Pointer to a string |
| <code>SHOW_ARRAY</code> | Pointer to another <code>st_mysql_show_var</code> array |
| <code>SHOW_FUNC</code> | Pointer to a function |
| <code>SHOW_DOUBLE</code> | Pointer to a double |

For the `SHOW_FUNC` type, the function is called and fills in its `out` parameter, which then provides information about the variable to be displayed. The function has this signature:

```
#define SHOW_VAR_FUNC_BUFF_SIZE 1024

typedef int (*mysql_show_var_func) (void *thd,
                                   struct st_mysql_show_var *out,
                                   char *buf);
```

The `system_vars` member, if not 0, points to an array of `st_mysql_sys_var` structures, each of which describes one system variable (which can also be set from the command-line or configuration file), followed by a structure with all members set to 0. The `st_mysql_sys_var` structure is defined as follows:

```
struct st_mysql_sys_var {
    int flags;
    const char *name, *comment;
    int (*check)(THD*, struct st_mysql_sys_var *, void*, st_mysql_value*);
    void (*update)(THD*, struct st_mysql_sys_var *, void*, const void*);
};
```

Additional fields are append as required depending upon the flags.

For convenience, a number of macros are defined that make creating new system variables within a plugin much simpler.

Throughout the macros, the following fields are available:

- `name`: An unquoted identifier for the system variable.

- `varname`: The identifier for the static variable. Where not available, it is the same as the `name` field.
- `opt`: Additional use flags for the system variable. The following table shows the permissible flags.

Table 23.2. Server Plugin System Variable Flags

| Flag Value | Description |
|----------------------------------|---|
| <code>PLUGIN_VAR_READONLY</code> | The system variable is read only |
| <code>PLUGIN_VAR_NOSYSVAR</code> | The system variable is not user visible at runtime |
| <code>PLUGIN_VAR_NOCMDOPT</code> | The system variable is not configurable from the command line |
| <code>PLUGIN_VAR_NOCMDARG</code> | No argument is required at the command line (typically used for boolean variables) |
| <code>PLUGIN_VAR_RQCMDARG</code> | An argument is required at the command line (this is the default) |
| <code>PLUGIN_VAR_OPCMDARG</code> | An argument is optional at the command line |
| <code>PLUGIN_VAR_MEMALLOC</code> | Used for string variables; indicates that memory is to be allocated for storage of the string |

- `comment`: A descriptive comment to be displayed in the server help message. `NULL` if this variable is to be hidden.
- `check`: The check function, `NULL` for default.
- `update`: The update function, `NULL` for default.
- `default`: The variable default value.
- `minimum`: The variable minimum value.
- `maximum`: The variable maximum value.
- `blocksize`: The variable block size. When the value is set, it is rounded to the nearest multiple of `blocksize`.

A system variable may be accessed either by using the static variable directly or by using the `SYSVAR()` accessor macro. The `SYSVAR()` macro is provided for completeness. Usually it should be used only when the code cannot directly access the underlying variable.

For example:

```
static int my_foo;
static MYSQL_SYSVAR_INT(foo_var, my_foo,
                        PLUGIN_VAR_RQCMDARG, "foo comment",
                        NULL, NULL, 0, 0, INT_MAX, 0);
...
SYSVAR(foo_var)= value;
value= SYSVAR(foo_var);
my_foo= value;
value= my_foo;
```

Session variables may be accessed only through the `THDVAR()` accessor macro. For example:

```
static MYSQL_THDVAR_BOOL(some_flag,
                        PLUGIN_VAR_NOCMDARG, "flag comment",
                        NULL, NULL, FALSE);
...
if (THDVAR(thd, some_flag))
{
    do_something();
    THDVAR(thd, some_flag)= FALSE;
```

```
}

```

All global and session system variables must be published to `mysqld` before use. This is done by constructing a `NULL`-terminated array of the variables and linking to it in the plugin public interface. For example:

```
static struct st_mysql_sys_var *my_plugin_vars[] = {
    MYSQL_SYSVAR(foo_var),
    MYSQL_SYSVAR(some_flag),
    NULL
};
mysql_declare_plugin(fooplug)
{
    MYSQL_..._PLUGIN,
    &plugin_data,
    "fooplug",
    "foo author",
    "This does foo!",
    PLUGIN_LICENSE_GPL,
    foo_init,
    foo_fini,
    0x0001,
    NULL,
    my_plugin_vars,
    NULL,
    0
}
mysql_declare_plugin_end;
```

The following convenience macros enable you to declare different types of system variables:

- Boolean system variables of type `my_bool`, which is a 1-byte boolean. (0 = FALSE, 1 = TRUE)

```
MYSQL_THDVAR_BOOL(name, opt, comment, check, update, default)
MYSQL_SYSVAR_BOOL(name, varname, opt, comment, check, update, default)
```

- String system variables of type `char*`, which is a pointer to a null-terminated string.

```
MYSQL_THDVAR_STR(name, opt, comment, check, update, default)
MYSQL_SYSVAR_STR(name, varname, opt, comment, check, update, default)
```

- Integer system variables, of which there are several varieties.

- An `int` system variable, which is typically a 4-byte signed word.

```
MYSQL_THDVAR_INT(name, opt, comment, check, update, default, min, max, blk)
MYSQL_SYSVAR_INT(name, varname, opt, comment, check, update, default,
                 minimum, maximum, blocksize)
```

- An `unsigned int` system variable, which is typically a 4-byte unsigned word.

```
MYSQL_THDVAR_UINT(name, opt, comment, check, update, default, min, max, blk)
MYSQL_SYSVAR_UINT(name, varname, opt, comment, check, update, default,
                  minimum, maximum, blocksize)
```

- A `long` system variable, which is typically either a 4- or 8-byte signed word.

```
MYSQL_THDVAR_LONG(name, opt, comment, check, update, default, min, max, blk)
MYSQL_SYSVAR_LONG(name, varname, opt, comment, check, update, default,
                  minimum, maximum, blocksize)
```

- An `unsigned long` system variable, which is typically either a 4- or 8-byte unsigned word.

```
MYSQL_THDVAR_ULONG(name, opt, comment, check, update, default, min, max, blk)
MYSQL_SYSVAR_ULONG(name, varname, opt, comment, check, update, default,
```

```
minimum, maximum, blocksize)
```

- A `long long` system variable, which is typically an 8-byte signed word.

```
MYSQL_THDVAR_LONGLONG(name, opt, comment, check, update,
                      default, minimum, maximum, blocksize)
MYSQL_SYSVAR_LONGLONG(name, varname, opt, comment, check, update,
                      default, minimum, maximum, blocksize)
```

- An `unsigned long long` system variable, which is typically an 8-byte unsigned word.

```
MYSQL_THDVAR_ULONGLONG(name, opt, comment, check, update,
                       default, minimum, maximum, blocksize)
MYSQL_SYSVAR_ULONGLONG(name, varname, opt, comment, check, update,
                       default, minimum, maximum, blocksize)
```

- An `unsigned long` system variable, which is typically either a 4- or 8-byte unsigned word. The range of possible values is an ordinal of the number of elements in the `typelib`, starting from 0.

```
MYSQL_THDVAR_ENUM(name, opt, comment, check, update, default, typelib)
MYSQL_SYSVAR_ENUM(name, varname, opt, comment, check, update,
                  default, typelib)
```

- An `unsigned long long` system variable, which is typically an 8-byte unsigned word. Each bit represents an element in the `typelib`.

```
MYSQL_THDVAR_SET(name, opt, comment, check, update, default, typelib)
MYSQL_SYSVAR_SET(name, varname, opt, comment, check, update,
                 default, typelib)
```

Internally, all mutable and plugin system variables are stored in a `HASH` structure.

Display of the server command-line help text is handled by compiling a `DYNAMIC_ARRAY` of all variables relevant to command-line options, sorting them, and then iterating through them to display each option.

When a command-line option has been handled, it is then removed from the `argv` by the `handle_option()` function (`my_getopt.c`); in effect, it is consumed.

The server processes command-line options during the plugin installation process, immediately after the plugin has been successfully loaded but before the plugin initialization function has been called

Plugins loaded at runtime do not benefit from any configuration options and must have usable defaults. Once they are installed, they are loaded at `mysqld` initialization time and configuration options can be set at the command line or within `my.cnf`.

Plugins should consider the `thd` parameter to be read only.

23.2.4.2.3. Client Plugin Descriptors

Each client plugin must have a descriptor that provides information to the client plugin API. The descriptor structure begins with a fixed set of members common to all client plugins, followed by any members specific to the plugin type.

The `st_mysql_client_plugin` structure in the `client_plugin.h` file defines a “generic” descriptor that contains the common members:

```
struct st_mysql_client_plugin
{
    int type;
    unsigned int interface_version;
    const char *name;
    const char *author;
```

```

const char *desc;
unsigned int version[3];
const char *license;
void *mysql_api;
int (*init)(char *, size_t, int, va_list);
int (*deinit)();
int (*options)(const char *option, const void *);
};

```

The common `st_mysql_client_plugin` descriptor structure members are used as follows. `char *` members should be specified as null-terminated strings.

- `type`: The plugin type. This must be one of the plugin-type values from `client_plugin.h`, such as `MYSQL_CLIENT_AUTHENTICATION_PLUGIN`.
- `interface_version`: The plugin interface version. For example, this is `MYSQL_CLIENT_AUTHENTICATION_PLUGIN_INTERFACE_VERSION` for an authentication plugin.
- `name`: A string that gives the plugin name. This is the name by which you refer to the plugin when you call `mysql_options()` with the `MYSQL_DEFAULT_AUTH` option or specify the `--default-auth` option to a MySQL client program.
- `author`: A string naming the plugin author. This can be whatever you like.
- `desc`: A string that provides a general description of the plugin. This can be whatever you like.
- `version`: The plugin version as an array of three integers indicating the major, minor, and teeny versions. For example, `{1, 2, 3}` indicates version 1.2.3.
- `license`: A string that specifies the license type.
- `mysql_api`: For internal use. Specify it as `NULL` in the plugin descriptor.
- `init`: A once-only initialization function, or `NULL` if there is no such function. The client library executes this function when it loads the plugin. The function returns zero for success and nonzero for failure.

The `init` function uses its first two arguments to return an error message if an error occurs. The first argument is a pointer to a `char` buffer, and the second argument indicates the buffer length. Any message returned by the `init` function must be null-terminated, so the maximum message length is the buffer length minus one. The next arguments are passed to `mysql_load_plugin()`. The first indicates how many more arguments there are (0 if none), followed by any remaining arguments.

- `deinit`: A once-only deinitialization function, or `NULL` if there is no such function. The client library executes this function when it unloads the plugin. The function takes no arguments. It returns zero for success and nonzero for failure.
- `options`: A function for handling options passed to the plugin, or `NULL` if there is no such function. The function takes two arguments representing the option name and a pointer to its value.

For a given client plugin type, the common descriptor members may be followed by additional members necessary to implement plugins of that type. For example, the `st_mysql_client_plugin_AUTHENTICATION` structure for authentication plugins has a function at the end that the client library calls to perform authentication.

To declare a plugin, use the `mysql_declare_client_plugin()` and `mysql_end_client_plugin` macros:

```

mysql_declare_client_plugin(plugin_type)
... members common to all client plugins ...
... type-specific extra members ...
mysql_end_client_plugin;

```

Do not specify the `type` or `interface_version` member explicitly. The `mysql_declare_client_plugin()` macro uses the `plugin_type` argument to generate their values automatically. For example, declare an authentication client plugin like this:

```
mysql_declare_client_plugin(AUTHENTICATION)
    "my_auth_plugin",
    "Author Name",
    "My Client Authentication Plugin",
    {1,0,0},
    "GPL",
    NULL,
    my_auth_init,
    my_auth_deinit,
    my_auth_options,
    my_auth_main
mysql_end_client_plugin;
```

This declaration uses the `AUTHENTICATION` argument to set the `type` and `interface_version` members to `MYSQL_CLIENT_AUTHENTICATION_PLUGIN` and `MYSQL_CLIENT_AUTHENTICATION_PLUGIN_INTERFACE_VERSION`.

Depending on the plugin type, the descriptor may have other members following the common members. For example, for an authentication plugin, there is a function (`my_auth_main()` in the descriptor just shown) that handles communication with the server. See [Section 23.2.4.9, “Writing Authentication Plugins”](#).

Normally, a client program that supports the use of authentication plugins causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

Should a client program require lower-level plugin management, the client library contains functions that take an `st_mysql_client_plugin` argument. See [Section 22.8.10, “C API Client Plugin Functions”](#).

23.2.4.3. Compiling and Installing Plugin Libraries

After your plugin is written, you must compile it and install it. The procedure for compiling shared objects varies from system to system. If you build your library using `CMake`, it should be able to generate the correct compilation commands for your system. If the library is named `somepluglib`, you should end up with a shared object file that has a name something like `somepluglib.so`. (The file name might have a different extension on your system.)

To use `CMake`, you'll need to set up the configuration files to enable the plugin to be compiled and installed. Use the plugin examples under the `plugin` directory of a MySQL source distribution as a guide.

Create `CMakeLists.txt`, which should look something like this:

```
MYSQL_ADD_PLUGIN(somepluglib somepluglib.c
    MODULE_ONLY MODULE_OUTPUT_NAME "somepluglib")
```

When `CMake` generates the `Makefile`, it should take care of passing to the compilation command the `-DMYSQL_DYNAMIC_PLUGIN` flag, and passing to the linker the `-lmysqlservices` flag, which is needed to link in any functions from services provided through the plugin services interface. See [Section 23.2.5, “MySQL Services for Plugins”](#).

Run `CMake`, then run `make`:

```
shell> cmake .
shell> make
```

If you need to specify configuration options to `CMake`, see [Section 2.10.4, “MySQL Source-Configuration Options”](#), for a list. For example, you might want to specify `CMAKE_INSTALL_PREFIX` [\[151\]](#) to indicate the MySQL base directory under which the plugin should be installed. You can see what value to use for this option with `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'basedir';
+-----+-----+
| Variable_name | Value           |
+-----+-----+
| base          | /usr/local/mysql |
+-----+-----+
```

The location of the plugin directory where you should install the library is given by the `plugin_dir` [\[537\]](#) system variable. For example:

```
mysql> SHOW VARIABLES LIKE 'plugin_dir';
+-----+-----+
| Variable_name | Value           |
+-----+-----+
| plugin_dir    | /usr/local/mysql/lib/mysql/plugin |
+-----+-----+
```

To install the plugin library, use `make`:

```
shell> make install
```

Verify that `make install` installed the plugin library in the proper directory. After installing it, make sure that the library permissions permit it to be executed by the server.

23.2.4.4. Writing Full-Text Parser Plugins

A full-text parser server plugin can be used to replace or modify the built-in full-text parser. This section describes how to write a full-text parser plugin named `simple_parser`. This plugin performs parsing based on simpler rules than those used by the MySQL built-in full-text parser: Words are nonempty runs of whitespace characters.

The instructions use the source code in the `plugin/fulltext` directory of MySQL source distributions, so change location into that directory. The following procedure describes how the plugin library is created:

1. To write a full-text parser plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed.

```
#include <mysql/plugin.h>
```

`plugin.h` defines the `MYSQL_FTPARSER_PLUGIN` server plugin type and the data structures needed to declare the plugin.

2. Set up the library descriptor for the plugin library file.

This descriptor contains the general plugin descriptor for the server plugin. For a full-text parser plugin, the type must be `MYSQL_FTPARSER_PLUGIN`. This is the value that identifies the plugin as being legal for use in a `WITH PARSER` clause when creating a `FULLTEXT` index. (No other plugin type is legal for this clause.)

For example, the library descriptor for a library that contains a single full-text parser plugin named `simple_parser` looks like this:

```
mysql_declare_plugin(ftexample)
{
    MYSQL_FTPARSER_PLUGIN,          /* type                */
    &simple_parser_descriptor,       /* descriptor          */
    "simple_parser",                 /* name                */
    "Oracle Corporation",          /* author              */
    "Simple Full-Text Parser",     /* description         */
    PLUGIN_LICENSE_GPL,           /* plugin license     */
    simple_parser_plugin_init,     /* init function (when loaded) */
    simple_parser_plugin_deinit,   /* deinit function (when unloaded) */
    0x0001,                         /* version            */
    simple_status,                 /* status variables   */
    simple_system_variables,       /* system variables   */
    NULL,                          /*                     */
    0
}
mysql_declare_plugin_end;
```

The `name` member (`simple_parser`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

For more information, see [Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”](#).

3. Set up the type-specific plugin descriptor.

Each general plugin descriptor in the library descriptor points to a type-specific descriptor. For a full-text parser plugin, the type-specific descriptor is an instance of the `st_mysql_ftparser` structure in the `plugin.h` file:

```
struct st_mysql_ftparser
{
    int interface_version;
    int (*parse)(MYSQL_FTPARSER_PARAM *param);
    int (*init)(MYSQL_FTPARSER_PARAM *param);
    int (*deinit)(MYSQL_FTPARSER_PARAM *param);
};
```

As shown by the structure definition, the descriptor has an interface version number and contains pointers to three functions. The version is specified using a symbol of the form `MYSQL_xxx_INTERFACE_VERSION` (such as `MYSQL_FTPARSER_INTERFACE_VERSION` for full-text parser plugins). The `init` and `deinit` members should point to a function or be set to 0 if the function is not needed. The `parse` member must point to the function that performs the parsing.

In the `simple_parser` declaration, that descriptor is indicated by `&simple_parser_descriptor`. The descriptor specifies the version number for the full-text plugin interface (as given by `MYSQL_FTPARSER_INTERFACE_VERSION`), and the plugin's parsing, initialization, and deinitialization functions:

```
static struct st_mysql_ftparser simple_parser_descriptor=
{
    MYSQL_FTPARSER_INTERFACE_VERSION, /* interface version */
    simple_parser_parse,             /* parsing function */
    simple_parser_init,              /* parser init function */
    simple_parser_deinit             /* parser deinit function */
};
```

A full-text parser plugin is used in two different contexts, indexing and searching. In both contexts, the server calls the initialization and deinitialization functions at the beginning and end of processing each SQL statement that causes the plugin to be invoked. However, during statement processing, the server calls the main parsing function in context-specific fashion:

- For indexing, the server calls the parser for each column value to be indexed.
- For searching, the server calls the parser to parse the search string. The parser might also be called for rows processed by the statement. In natural language mode, there is no need for the server to call the parser. For boolean mode phrase searches or natural language searches with query expansion, the parser is used to parse column values for information that is not in the index. Also, if a boolean mode search is done for a column that has no `FULLTEXT` index, the built-in parser will be called. (Plugins are associated with specific indexes. If there is no index, no plugin is used.)

The plugin declaration in the general plugin descriptor has `init` and `deinit` members that point initialization and deinitialization functions, and so does the type-specific plugin descriptor to which it points. However, these pairs of functions have different purposes and are invoked for different reasons:

- For the plugin declaration in the general plugin descriptor, the initialization and deinitialization functions are invoked when the plugin is loaded and unloaded.
- For the type-specific plugin descriptor, the initialization and deinitialization functions are invoked per SQL statement for which the plugin is used.

Each interface function named in the plugin descriptor should return zero for success or nonzero for failure, and each of them receives an argument that points to a `MYSQL_FTPARSER_PARAM` structure containing the parsing context. The structure has this definition:

```
typedef struct st_mysql_ftparser_param
{
    int (*mysql_parse)(struct st_mysql_ftparser_param *,
                     char *doc, int doc_len);
    int (*mysql_add_word)(struct st_mysql_ftparser_param *,
                        char *word, int word_len,
                        MYSQL_FTPARSER_BOOLEAN_INFO *boolean_info);
    void *ftparser_state;
    void *mysql_ftparam;
    struct charset_info_st *cs;
    char *doc;
    int length;
    int flags;
    enum enum_ftparser_mode mode;
} MYSQL_FTPARSER_PARAM;
```

The structure members are used as follows:

- `mysql_parse`: A pointer to a callback function that invokes the server's built-in parser. Use this callback when the plugin acts as a front end to the built-in parser. That is, when the plugin parsing function is called, it should process the input to extract the text and pass the text to the `mysql_parse` callback.

The first parameter for this callback function should be the `param` value itself:

```
param->mysql_parse(param, ...);
```

A front end plugin can extract text and pass it all at once to the built-in parser, or it can extract and pass text to the built-in parser a piece at a time. However, in this case, the built-in parser treats the pieces of text as though there are implicit word breaks between them.

- `mysql_add_word`: A pointer to a callback function that adds a word to a full-text index or to the list of search terms. Use this callback when the parser plugin replaces the built-in parser. That is, when the plugin parsing function is called, it should parse the input into words and invoke the `mysql_add_word` callback for each word.

The first parameter for this callback function should be the `param` value itself:

```
param->mysql_add_word(param, ...);
```

- `ftparser_state`: This is a generic pointer. The plugin can set it to point to information to be used internally for its own purposes.
- `mysql_ftparam`: This is set by the server. It is passed as the first argument to the `mysql_parse` or `mysql_add_word` callback.
- `cs`: A pointer to information about the character set of the text, or 0 if no information is available.
- `doc`: A pointer to the text to be parsed.
- `length`: The length of the text to be parsed, in bytes.
- `flags`: Parser flags. This is zero if there are no special flags. Currently, the only nonzero flag is `MYSQL_FTFLAGS_NEED_COPY`, which means that `mysql_add_word()` must save a copy of the word (that is, it cannot use a pointer to the word because the word is in a buffer that will be overwritten.) This member was added in MySQL 5.1.12.

This flag might be set or reset by MySQL before calling the parser plugin, by the parser plugin itself, or by the `mysql_parse()` function.

- `mode`: The parsing mode. This value will be one of the following constants:
 - `MYSQL_FTPARSER_SIMPLE_MODE`: Parse in fast and simple mode, which is used for indexing and for natural language queries. The parser should pass to the server only those words that should be indexed. If the parser uses length limits or a stopword list to determine which words to ignore, it should not pass such words to the server.
 - `MYSQL_FTPARSER_WITH_STOPWORDS`: Parse in stopword mode. This is used in boolean searches for phrase matching. The parser should pass all words to the server, even stopwords or words that are outside any normal length limits.
 - `MYSQL_FTPARSER_FULL_BOOLEAN_INFO`: Parse in boolean mode. This is used for parsing boolean query strings. The parser should recognize not only words but also boolean-mode operators and pass them to the server as tokens using the `mysql_add_word` callback. To tell the server what kind of token is being passed, the plugin needs to fill in a `MYSQL_FTPARSER_BOOLEAN_INFO` structure and pass a pointer to it.

If the parser is called in boolean mode, the `param->mode` value will be `MYSQL_FTPARSER_FULL_BOOLEAN_INFO`. The `MYSQL_FTPARSER_BOOLEAN_INFO` structure that the parser uses for passing token information to the server looks like this:

```
typedef struct st_mysql_ftparser_boolean_info
{
    enum enum_ft_token_type type;
    int yesno;
    int weight_adjust;
    bool wasign;
    bool trunc;
    /* These are parser state and must be removed. */
    byte prev;
    byte *quot;
} MYSQL_FTPARSER_BOOLEAN_INFO;
```

The parser should fill in the structure members as follows:

- `type`: The token type. The following table shows the permissible types.

Table 23.3. Full-Text Parser Token Types

| Token Value | Meaning |
|-----------------------------------|---|
| <code>FT_TOKEN_EOF</code> | End of data |
| <code>FT_TOKEN_WORD</code> | A regular word |
| <code>FT_TOKEN_LEFT_PAREN</code> | The beginning of a group or subexpression |
| <code>FT_TOKEN_RIGHT_PAREN</code> | The end of a group or subexpression |
| <code>FT_TOKEN_STOPWORD</code> | A stopword |

- `yesno`: Whether the word must be present for a match to occur. 0 means that the word is optional but increases the match relevance if it is present. Values larger than 0 mean that the word must be present. Values smaller than 0 mean that the word must not be present.
- `weight_adjust`: A weighting factor that determines how much a match for the word counts. It can be used to increase or decrease the word's importance in relevance calculations. A value of zero indicates no weight adjustment. Values greater than or less than zero mean higher or lower weight, respectively. The examples at [Section 12.9.2, “Boolean Full-Text Searches”](#), that use the `<` and `>` operators illustrate how weighting works.
- `wasign`: The sign of the weighting factor. A negative value acts like the `~` boolean-search operator, which causes the word's contribution to the relevance to be negative.
- `trunc`: Whether matching should be done as if the boolean-mode `*` truncation operator had been given.

Plugins should not use the `prev` and `quot` members of the `MYSQL_FTPARSER_BOOLEAN_INFO` structure.

4. Set up the plugin interface functions.

The general plugin descriptor in the library descriptor names the initialization and deinitialization functions that the server should invoke when it loads and unloads the plugin. For `simple_parser`, these functions do nothing but return zero to indicate that they succeeded:

```
static int simple_parser_plugin_init(void *arg __attribute__((unused)))
{
    return(0);
}

static int simple_parser_plugin_deinit(void *arg __attribute__((unused)))
{
    return(0);
}
```

Because those functions do not actually do anything, you could omit them and specify 0 for each of them in the plugin declaration.

The type-specific plugin descriptor for `simple_parser` names the initialization, deinitialization, and parsing functions that the server invokes when the plugin is used. For `simple_parser`, the initialization and deinitialization functions do nothing:

```
static int simple_parser_init(MYSQL_FTPARSER_PARAM *param
                             __attribute__((unused)))
{
```

```

    return(0);
}

static int simple_parser_deinit(MYSQL_FTPARSER_PARAM *param
                               __attribute__((unused)))
{
    return(0);
}

```

Here too, because those functions do nothing, you could omit them and specify 0 for each of them in the plugin descriptor.

The main parsing function, `simple_parser_parse()`, acts as a replacement for the built-in full-text parser, so it needs to split text into words and pass each word to the server. The parsing function's first argument is a pointer to a structure that contains the parsing context. This structure has a `doc` member that points to the text to be parsed, and a `length` member that indicates how long the text is. The simple parsing done by the plugin considers nonempty runs of whitespace characters to be words, so it identifies words like this:

```

static int simple_parser_parse(MYSQL_FTPARSER_PARAM *param)
{
    char *end, *start, *docend= param->doc + param->length;

    for (end= start= param->doc;; end++)
    {
        if (end == docend)
        {
            if (end > start)
                add_word(param, start, end - start);
            break;
        }
        else if (isspace(*end))
        {
            if (end > start)
                add_word(param, start, end - start);
            start= end + 1;
        }
    }
    return(0);
}

```

As the parser finds each word, it invokes a function `add_word()` to pass the word to the server. `add_word()` is a helper function only; it is not part of the plugin interface. The parser passes the parsing context pointer to `add_word()`, as well as a pointer to the word and a length value:

```

static void add_word(MYSQL_FTPARSER_PARAM *param, char *word, size_t len)
{
    MYSQL_FTPARSER_BOOLEAN_INFO bool_info=
        { FT_TOKEN_WORD, 0, 0, 0, 0, ' ', 0 };

    param->mysql_add_word(param, word, len, &bool_info);
}

```

For boolean-mode parsing, `add_word()` fills in the members of the `bool_info` structure as described earlier in the discussion of the `st_mysql_ftparser_boolean_info` structure.

5. Set up the status variables. For the `simple_parser` plugin, the following status variable array sets up one status variable with a value that is static text, and another with a value that is stored in a long integer variable:

```

long number_of_calls= 0;

struct st_mysql_show_var simple_status[]=

```

```
{
  {"static", (char *)"just a static text", SHOW_CHAR},
  {"called", (char *)&number_of_calls, SHOW_LONG},
  {0,0,0}
};
```

When the plugin is installed, the plugin name and the `name` value are joined with an underscore to form the name displayed by `SHOW STATUS`. For the array just shown, the resulting status variable names are `simple_parser_static` and `simple_parser_called`. This convention means that you can easily display the variables for a plugin using its name:

```
mysql> SHOW STATUS LIKE 'simple_parser%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| simple_parser_static | just a static text |
| simple_parser_called | 0 |
+-----+-----+
```

- To compile and install a plugin library object file, use the instructions in [Section 23.2.4.3, “Compiling and Installing Plugin Libraries”](#). To use the library file, it must be installed in the plugin directory (the directory named by the `plugin_dir` [537] system variable). For the `simple_parser` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `mypluglib.so` (the extension might be different depending on your platform).
- To use the plugin, register it with the server. For example, to register the plugin at runtime, use this statement (changing the extension as necessary):

```
mysql> INSTALL PLUGIN simple_parser SONAME 'mypluglib.so';
```

For additional information about plugin loading, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

- To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement.
- Test the plugin to verify that it works properly.

Create a table that contains a string column and associate the parser plugin with a `FULLTEXT` index on the column:

```
mysql> CREATE TABLE t (c VARCHAR(255),
-> FULLTEXT (c) WITH PARSER simple_parser
-> ) ENGINE=MyISAM;
Query OK, 0 rows affected (0.01 sec)
```

Insert some text into the table and try some searches. These should verify that the parser plugin treats all nonwhitespace characters as word characters:

```
mysql> INSERT INTO t VALUES
-> ('latin1_general_cs is a case-sensitive collation'),
-> ('I\'d like a case of oranges'),
-> ('this is sensitive information'),
-> ('another row'),
-> ('yet another row');
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT c FROM t;
+-----+
| c |
+-----+
```

```

+-----+
| latin1_general_cs is a case-sensitive collation |
| I'd like a case of oranges                     |
| this is sensitive information                  |
| another row                                   |
| yet another row                               |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('case') FROM t;
+-----+
| MATCH(c) AGAINST('case') |
+-----+
|                          0 |
| 1.2968142032623          |
|                          0 |
|                          0 |
|                          0 |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('sensitive') FROM t;
+-----+
| MATCH(c) AGAINST('sensitive') |
+-----+
|                          0 |
|                          0 |
| 1.3253291845322            |
|                          0 |
|                          0 |
+-----+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('case-sensitive') FROM t;
+-----+
| MATCH(c) AGAINST('case-sensitive') |
+-----+
| 1.3109166622162           |
|                          0 |
|                          0 |
|                          0 |
|                          0 |
+-----+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('I\'d') FROM t;
+-----+
| MATCH(c) AGAINST('I\'d') |
+-----+
|                          0 |
| 1.2968142032623          |
|                          0 |
|                          0 |
|                          0 |
+-----+
5 rows in set (0.01 sec)

```

Note how neither “case” nor “insensitive” match “case-insensitive” the way that they would for the built-in parser.

23.2.4.5. Writing Daemon Plugins

A daemon plugin is a simple type of plugin used for code that should be run by the server but that does not communicate with it. This section describes how to write a daemon server plugin, using the example plugin found in the [plugin/daemon_example](#) directory of MySQL source distributions. That directory

contains the `daemon_example.cc` source file for a daemon plugin named `daemon_example` that writes a heartbeat string at regular intervals to a file named `mysql-heartbeat.log` in the data directory.

To write a daemon plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed.

```
#include <mysql/plugin.h>
```

`plugin.h` defines the `MYSQL_DAEMON_PLUGIN` server plugin type and the data structures needed to declare the plugin.

The `daemon_example.cc` file sets up the library descriptor as follows. The library descriptor includes a single general server plugin descriptor.

```
mysql_declare_plugin(daemon_example)
{
    MYSQL_DAEMON_PLUGIN,
    &daemon_example_plugin,
    "daemon_example",
    "Brian Aker",
    "Daemon example, creates a heartbeat beat file in mysql-heartbeat.log",
    PLUGIN_LICENSE_GPL,
    daemon_example_plugin_init, /* Plugin Init */
    daemon_example_plugin_deinit, /* Plugin Deinit */
    0x0100 /* 1.0 */,
    NULL, /* status variables */
    NULL, /* system variables */
    NULL, /* config options */
    0, /* flags */
}
mysql_declare_plugin_end;
```

The `name` member (`daemon_example`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

The second member of the plugin descriptor, `daemon_example_plugin`, points to the type-specific daemon plugin descriptor. This structure consists only of the type-specific API version number:

```
struct st_mysql_daemon daemon_example_plugin=
{ MYSQL_DAEMON_INTERFACE_VERSION };
```

The type-specific structure has no interface functions. There is no communication between the server and the plugin, except that the server calls the initialization and deinitialization functions from the general plugin descriptor to start and stop the plugin:

- `daemon_example_plugin_init()` opens the heartbeat file and spawns a thread that wakes up periodically and writes the next message to the file.
- `daemon_example_plugin_deinit()` closes the file and performs other cleanup.

To compile and install a plugin library object file, use the instructions in [Section 23.2.4.3, “Compiling and Installing Plugin Libraries”](#). To use the library file, it must be installed in the plugin directory (the directory named by the `plugin_dir` [537] system variable). For the `daemon_example` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `libdaemon_example.so` (the extension might be different depending on your platform).

To use the plugin, register it with the server. For example, to register the plugin at runtime, use this statement (change the extension as necessary):

```
mysql> INSTALL PLUGIN daemon_example SONAME 'libdaemon_example.so';
```

For additional information about plugin loading, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement.

While the plugin is loaded, it writes a heartbeat string at regular intervals to a file named `mysql-heartbeat.log` in the data directory. This file grows without limit, so after you have satisfied yourself that the plugin operates correctly, unload it:

```
mysql> UNINSTALL PLUGIN daemon_example;
```

23.2.4.6. Writing `INFORMATION_SCHEMA` Plugins

This section describes how to write an `INFORMATION_SCHEMA` table server plugin. For example code that implements such plugins, see the `sql/sql_show.cc` file of a MySQL source distribution. You can also look at the example plugins found in the InnoDB source. See the `handler/i_s.cc` and `handler/ha_innodb.cc` files within the InnoDB source tree (in the `storage/innobase` directory).

To write an `INFORMATION_SCHEMA` table plugin, include the following header files in the plugin source file. Other MySQL or general header files might also be needed.

```
#include <sql_class.h>
#include <table.h>
```

These header files are located in the `sql` directory of MySQL source distributions. They contain C++ structures, so the source file for an `INFORMATION_SCHEMA` plugin must be compiled as C++ (not C) code.

The source file for the example plugin developed here is named `simple_i_s_table.cc`. It creates a simple `INFORMATION_SCHEMA` table named `SIMPLE_I_S_TABLE` that has two columns named `NAME` and `VALUE`. The general descriptor for a plugin library that implements the table looks like this:

```
mysql_declare_plugin(simple_i_s_library)
{
    MYSQL_INFORMATION_SCHEMA_PLUGIN,
    &simple_table_info,           /* type-specific descriptor */
    "SIMPLE_I_S_TABLE",        /* table name */
    "Author Name",            /* author */
    "Simple INFORMATION_SCHEMA table", /* description */
    PLUGIN_LICENSE_GPL,      /* license type */
    simple_table_init,        /* init function */
    NULL,
    0x0100,                   /* version = 1.0 */
    NULL,                     /* no status variables */
    NULL,                     /* no system variables */
    NULL,                     /* no reserved information */
    0                          /* no flags */
}
mysql_declare_plugin_end;
```

The `name` member (`SIMPLE_I_S_TABLE`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

The `simple_table_info` member of the general descriptor points to the type-specific descriptor, which consists only of the type-specific API version number:

```
static struct st_mysql_information_schema simple_table_info =
{ MYSQL_INFORMATION_SCHEMA_INTERFACE_VERSION };
```

The general descriptor points to the initialization and deinitialization functions:

- The initialization function provides information about the table structure and a function that populates the table.
- The deinitialization function performs any required cleanup. If no cleanup is needed, this descriptor member can be `NULL` (as in the example shown).

The initialization function should return 0 for success, 1 if an error occurs. The function receives a generic pointer, which it should interpret as a pointer to the table structure:

```
static int table_init(void *ptr)
{
    ST_SCHEMA_TABLE *schema_table= (ST_SCHEMA_TABLE*)ptr;

    schema_table->fields_info= simple_table_fields;
    schema_table->fill_table= simple_fill_table;
    return 0;
}
```

The function should set these two members of the table structure:

- `fields_info`: An array of `ST_FIELD_INFO` structures that contain information about each column.
- `fill_table`: A function that populates the table.

The array pointed to by `fields_info` should contain one element per column of the `INFORMATION_SCHEMA` plus a terminating element. The following `simple_table_fields` array for the example plugin indicates that `SIMPLE_I_S_TABLE` has two columns. `NAME` is string-valued with a length of 10 and `VALUE` is integer-valued with a display width of 20. The last structure marks the end of the array.

```
static ST_FIELD_INFO simple_table_fields[]=
{
    {"NAME", 10, MYSQL_TYPE_STRING, 0, 0 0, 0},
    {"VALUE", 6, MYSQL_TYPE_LONG, 0, MY_I_S_UNSIGNED, 0, 0},
    {0, 0, MYSQL_TYPE_NULL, 0, 0, 0, 0}
};
```

For more information about the column information structure, see the definition of `ST_FIELD_INFO` in the `table.h` header file. The permissible `MYSQL_TYPE_XXX` type values are those used in the C API; see [Section 22.8.1, "C API Data Structures"](#).

The `fill_table` member should be set to a function that populates the table and returns 0 for success, 1 if an error occurs. For the example plugin, the `simple_fill_table()` function looks like this:

```
static int simple_fill_table(THD *thd, TABLE_LIST *tables, COND *cond)
{
    TABLE *table= tables->table;

    table->field[0]->store("Name 1", 6, system_charset_info);
    table->field[1]->store(1);
    if (schema_table_store_record(thd, table))
        return 1;
    table->field[0]->store("Name 2", 6, system_charset_info);
    table->field[1]->store(2);
    if (schema_table_store_record(thd, table))
        return 1;
    return 0;
}
```

For each row of the `INFORMATION_SCHEMA` table, this function initializes each column, then calls `schema_table_store_record()` to install the row. The `store()` method arguments depend on the

type of value to be stored. For column 0 (`NAME`, a string), `store()` takes a pointer to a string, its length, and information about the character set of the string:

```
store(const char *to, uint length, CHARSET_INFO *cs);
```

For column 1 (`VALUE`, an integer), `store()` takes the value and a flag indicating whether it is unsigned:

```
store(longlong nr, bool unsigned_value);
```

For other examples of how to populate `INFORMATION_SCHEMA` tables, search for instances of `schema_table_store_record()` in `sql_show.cc`.

To compile and install a plugin library object file, see the instructions in [Section 23.2.4.3, “Compiling and Installing Plugin Libraries”](#). To use the library file, it must be installed in the plugin directory (the directory named by the `plugin_dir` [537] system variable).

To test the plugin, install it:

```
mysql> INSTALL PLUGIN SIMPLE_I_S_TABLE SONAME 'simple_i_s_table.so';
```

Verify that the table is present:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
      -> WHERE TABLE_NAME = 'SIMPLE_I_S_TABLE';
+-----+
| TABLE_NAME |
+-----+
| SIMPLE_I_S_TABLE |
+-----+
```

Try to select from it:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SIMPLE_I_S_TABLE;
+-----+-----+
| NAME | VALUE |
+-----+-----+
| Name 1 | 1 |
| Name 2 | 2 |
+-----+-----+
```

Uninstall it:

```
mysql> UNINSTALL PLUGIN SIMPLE_I_S_TABLE;
```

23.2.4.7. Writing Semisynchronous Replication Plugins

This section describes how to write semisynchronous replication server plugins, using the example plugins found in the `plugin/semisync` directory of MySQL source distributions. That directory contains the source files for master and slave plugins named `rpl_semi_sync_master` and `rpl_semi_sync_slave`. The information here covers only how to set up the plugin framework. For details about how the plugins implement replication functions, see the source.

To write a semisynchronous replication plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed.

```
#include <mysql/plugin.h>
```

`plugin.h` defines the `MYSQL_REPLICATION_PLUGIN` server plugin type and the data structures needed to declare the plugin.

For the master side, `semisync_master_plugin.cc` contains this general descriptor for a plugin named `rpl_semi_sync_master`:

```
mysql_declare_plugin(semi_sync_master)
{
    MYSQL_REPLICATION_PLUGIN,
    &semi_sync_master_plugin,
    "rpl_semi_sync_master",
    "He Zhenxing",
    "Semi-synchronous replication master",
    PLUGIN_LICENSE_GPL,
    semi_sync_master_plugin_init, /* Plugin Init */
    semi_sync_master_plugin_deinit, /* Plugin Deinit */
    0x0100 /* 1.0 */,
    semi_sync_master_status_vars, /* status variables */
    semi_sync_master_system_vars, /* system variables */
    NULL, /* config options */
    0, /* flags */
}
mysql_declare_plugin_end;
```

For the slave side, `semisync_slave_plugin.cc` contains this general descriptor for a plugin named `rpl_semi_sync_slave`:

```
mysql_declare_plugin(semi_sync_slave)
{
    MYSQL_REPLICATION_PLUGIN,
    &semi_sync_slave_plugin,
    "rpl_semi_sync_slave",
    "He Zhenxing",
    "Semi-synchronous replication slave",
    PLUGIN_LICENSE_GPL,
    semi_sync_slave_plugin_init, /* Plugin Init */
    semi_sync_slave_plugin_deinit, /* Plugin Deinit */
    0x0100 /* 1.0 */,
    semi_sync_slave_status_vars, /* status variables */
    semi_sync_slave_system_vars, /* system variables */
    NULL, /* config options */
    0, /* flags */
}
mysql_declare_plugin_end;
```

For both the master and slave plugins, the general descriptor has pointers to the type-specific descriptor, the initialization and deinitialization functions, and to the status and system variables implemented by the plugin. For information about variable setup, see [Section 23.2.4.2.2, “Server Plugin Status and System Variables”](#). The following remarks discuss the type-specific descriptor and the initialization and deinitialization functions for the master plugin but apply similarly to the slave plugin.

The `semi_sync_master_plugin` member of the master general descriptor points to the type-specific descriptor, which consists only of the type-specific API version number:

```
struct Mysql_replication semi_sync_master_plugin= {
    MYSQL_REPLICATION_INTERFACE_VERSION
};
```

The initialization and deinitialization function declarations look like this:

```
static int semi_sync_master_plugin_init(void *p);
static int semi_sync_master_plugin_deinit(void *p);
```

The initialization function uses the pointer to register transaction and binary logging “observers” with the server. After successful initialization, the server takes care of invoking the observers at the appropriate times. (For details on the observers, see the source files.) The deinitialization function cleans up by deregistering the observers. Each function returns 0 for success or 1 if an error occurs.

To compile and install a plugin library object file, use the instructions in [Section 23.2.4.3, “Compiling and Installing Plugin Libraries”](#). To use the library files, they must be installed in the plugin directory (the directory named by the `plugin_dir` [537] system variable). For the `rpl_semi_sync_master` and `rpl_semi_sync_slave` plugins, they are compiled and installed when you build MySQL from source. They are also included in binary distributions. The build process produces shared object libraries with names of `semisync_master.so` and `semisync_slave.so` (the extension might be different depending on your platform).

23.2.4.8. Writing Audit Plugins

This section describes how to write an audit server plugin, using the example plugin found in the `plugin/audit_null` directory of MySQL source distributions. The `audit_null.c` source file in that directory implements a simple example audit plugin named `NULL_AUDIT`.

This description is current as of MySQL 5.5.14. For differences in the interface in earlier versions, see [Audit Plugin Interface Changes](#).

Within the server, the pluggable audit interface is implemented in the `sql_audit.h` and `sql_audit.cc` files in the `sql` directory of MySQL source distributions. Additionally, several places in the server are modified to call the audit interface when an auditable event occurs, so that registered audit plugins can be notified about the event if necessary. To see where such calls occur, look for invocations of functions with names of the form `mysql_audit_xxx()`. Audit notification occurs for server operations such as these:

- Writing a message to the general query log (if the log is enabled)
- Writing a message to the error log
- Sending a query result to a client
- Client connect and disconnect events

To write an audit plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed.

```
#include <mysql/plugin_audit.h>
```

`plugin_audit.h` includes `plugin.h`, so you need not include the latter file explicitly. `plugin.h` defines the `MYSQL_AUDIT_PLUGIN` server plugin type and the data structures needed to declare the plugin. `plugin_audit.h` defines data structures specific to audit plugins.

An audit plugin, like any MySQL server plugin, has a general plugin descriptor (see [Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”](#)). In `audit_null.c`, the general descriptor looks like this:

```
mysql_declare_plugin(audit_null)
{
    MYSQL_AUDIT_PLUGIN,          /* type                */
    &audit_null_descriptor,      /* descriptor          */
    "NULL_AUDIT",                /* name                */
    "Oracle Corp",              /* author              */
    "Simple NULL Audit",        /* description         */
    PLUGIN_LICENSE_GPL,         /* license              */
    audit_null_plugin_init,     /* init function (when loaded) */
    audit_null_plugin_deinit,   /* deinit function (when unloaded) */
    0x0002,                      /* version             */
    simple_status,              /* status variables    */
    NULL,                        /* system variables    */
    NULL,
    0,
}
mysql_declare_plugin_end;
```

The `name` member (`NULL_AUDIT`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`.

The general descriptor also refers to `simple_status`, a structure that exposes several status variables to the `SHOW STATUS` statement:

```
static struct st_mysql_show_var simple_status[]=
{
  { "Audit_null_called", (char *) &number_of_calls, SHOW_INT },
  { "Audit_null_general_log", (char *) &number_of_calls_general_log,
    SHOW_INT },
  { "Audit_null_general_error", (char *) &number_of_calls_general_error,
    SHOW_INT },
  { "Audit_null_general_result", (char *) &number_of_calls_general_result,
    SHOW_INT },
  { 0, 0, 0 }
};
```

The `audit_null_plugin_init` initialization function sets the status variables to zero when the plugin is loaded. The `audit_null_plugin_deinit` function performs cleanup with the plugin is unloaded. During operation, the plugin increments the first status variable for each notification it receives. It also increments the others according to the event class and subclass. In effect, the first variable is the aggregate of the counts for the event subclasses.

The `audit_null_descriptor` value in the general descriptor points to the type-specific descriptor. For audit plugins, this descriptor has the following structure:

```
struct st_mysql_audit
{
  int interface_version;
  void (*release_thd)(MYSQL_THD);
  void (*event_notify)(MYSQL_THD, unsigned int, const void *);
  unsigned long class_mask[MYSQL_AUDIT_CLASS_MASK_SIZE];
};
```

The type-specific descriptor has these members:

- `interface_version`: By convention, type-specific plugin descriptors begin with the interface version for the given plugin type. The server checks `interface_version` when it loads the plugin to see whether the plugin is compatible with it. For audit plugins, the value of the `interface_version` member is `MYSQL_AUDIT_INTERFACE_VERSION` (defined in `plugin_audit.h`).
- `release_thd`: A function that the server calls to inform the plugin that it is being dissociated from its thread context. This should be `NULL` if there is no such function.
- `event_notify`: A function that the server calls to notify the plugin that an auditable event has occurred. This function should not be `NULL`; that would not make sense because no auditing would occur.
- `class_mask`: A bit mask that indicates the event classes for which the plugin wants to receive notification. If this value is 0, the server passes no events to the plugin.

The server uses the `event_notify` and `release_thd` functions together. They are called within the context of a specific thread, and a thread might perform an activity that produces several event notifications. The first time the server calls `event_notify` for a thread, it creates a binding of the plugin to the thread. The plugin cannot be uninstalled while this binding exists. When no more events for the thread will occur, the server informs the plugin of this by calling the `release_thd` function, and then destroys the binding. For example, when a client issues a statement, the thread processing the statement might notify audit plugins about the result set produced by the statement and about the statement being logged. After

these notifications occur, the server releases the plugin before putting the thread to sleep until the client issues another statement.

This design enables the plugin to allocate resources needed for a given thread in the first call to the `event_notify` function and release them in the `release_thd` function:

```
event_notify function:
  if memory is needed to service the thread
    allocate memory
  ... rest of notification processing ...

release_thd function:
  if memory was allocated
    release memory
  ... rest of release processing ...
```

That is more efficient than allocating and releasing memory repeatedly in the notification function.

For the `NULL_AUDIT` example audit plugin, the type-specific descriptor looks like this:

```
static struct st_mysql_audit audit_null_descriptor=
{
  MYSQL_AUDIT_INTERFACE_VERSION,          /* interface version */
  NULL,                                    /* release_thd function */
  audit_null_notify,                       /* notify function */
  { (unsigned long) MYSQL_AUDIT_GENERAL_CLASSMASK } /* class mask */
};
```

The server calls `audit_null_notify` to pass audit event information to the plugin. There is no `release_thd` function.

The event class mask indicates an interest in all events of the “general” class. `plugin_audit.h` defines its symbol, `MYSQL_AUDIT_GENERAL_CLASS`, and a mask with a bit for this class:

```
#define MYSQL_AUDIT_GENERAL_CLASS 0
#define MYSQL_AUDIT_GENERAL_CLASSMASK (1 << MYSQL_AUDIT_GENERAL_CLASS)
```

`plugin_audit.h` also has defines for a “connection” event class, although the `NULL_AUDIT` plugin does nothing with such events:

```
#define MYSQL_AUDIT_CONNECTION_CLASS 1
#define MYSQL_AUDIT_CONNECTION_CLASSMASK (1 << MYSQL_AUDIT_CONNECTION_CLASS)
```

A plugin could be written to receive both general and connection events by setting its type-specific descriptor class mask like this:

```
{ (unsigned long) MYSQL_AUDIT_GENERAL_CLASSMASK |
  MYSQL_AUDIT_CONNECTION_CLASSMASK } /* class mask */
```

In the type-specific descriptor, the second and third parameters of the `event_notify` function prototype represent the event class and a generic pointer to an event structure:

```
void (*event_notify)(MYSQL_THD, unsigned int, const void *);
```

Events in different classes may have different structures, so the notification function should use the event class value to determine how to interpret the pointer to the event structure.

If the server calls the notification function with an event class of `MYSQL_AUDIT_GENERAL_CLASS`, it passes the event structure as a pointer to a `mysql_event_general` structure:

```
struct mysql_event_general
{
```

```

unsigned int event_subclass;
int general_error_code;
unsigned long general_thread_id;
const char *general_user;
unsigned int general_user_length;
const char *general_command;
unsigned int general_command_length;
const char *general_query;
unsigned int general_query_length;
struct charset_info_st *general_charset;
unsigned long long general_time;
unsigned long long general_rows;
};

```

Audit plugins can interpret `mysql_event_general` members as follows:

- `event_subclass`: The event subclass, one of the following values:

```

#define MYSQL_AUDIT_GENERAL_LOG 0
#define MYSQL_AUDIT_GENERAL_ERROR 1
#define MYSQL_AUDIT_GENERAL_RESULT 2
#define MYSQL_AUDIT_GENERAL_STATUS 3

```

- `general_error_code`: The error code. This is a value like that returned by the `mysql_errno()` C API function; 0 means “no error.”
- `general_thread_id`: The ID of the thread for which the event occurred.
- `general_user`: The current user for the event.
- `general_user_length`: The length of `general_user`, in bytes.
- `general_command`: For general query log events, the type of operation. Examples: `Connect`, `Query`, `Shutdown`. For error log events, the error message. This is a value like that returned by the `mysql_error()` C API function; an empty string means “no error.” For result events, this is empty.
- `general_command_length`: The length of `general_command`, in bytes.
- `general_query`: The SQL statement that was logged or produced a result.
- `general_query_length`: The length of `general_query`, in bytes.
- `general_charset`: Character set information for the event.
- `general_time`: A `TIMESTAMP` value indicating the time just before the notification function was called.
- `general_rows`: For general query log events, zero. For error log events, the row number at which an error occurred. For result events, the number of rows in the result plus one. For statements that produce no result set, the value is 0. This encoding enables statements that produce no result set to be distinguished from those that produce an empty result set. For example, for a `DELETE` statement, this value is 0. For a `SELECT`, the result is always 1 or more, where 1 represents an empty result set.

The `NULL_AUDIT` plugin notification function is quite simple. It increments a global event counter, verifies that the event is of the “general” class, then looks at the event subclass to determine which subclass counter to increment:

```

static void audit_null_notify(MYSQL_THD thd __attribute__((unused)),
                             unsigned int event_class,
                             const void *event)
{
    number_of_calls++;
    if (event_class == MYSQL_AUDIT_GENERAL_CLASS)

```

```

{
  const struct mysql_event_general *event_general=
    (const struct mysql_event_general *) event;
  switch (event_general->event_subclass)
  {
  case MYSQL_AUDIT_GENERAL_LOG:
    number_of_calls_general_log++;
    break;
  case MYSQL_AUDIT_GENERAL_ERROR:
    number_of_calls_general_error++;
    break;
  case MYSQL_AUDIT_GENERAL_RESULT:
    number_of_calls_general_result++;
    break;
  default:
    break;
  }
}
}

```

To compile and install a plugin library object file, use the instructions in [Section 23.2.4.3, “Compiling and Installing Plugin Libraries”](#). To use the library file, it must be installed in the plugin directory (the directory named by the `plugin_dir` [537] system variable). For the `AUDIT_NULL` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `adt_null.so` (the extension might be different depending on your platform).

To register the plugin at runtime, use this statement (change the extension as necessary):

```
mysql> INSTALL PLUGIN NULL_AUDIT SONAME 'adt_null.so';
```

For additional information about plugin loading, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#).

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement.

While the audit plugin is installed, it exposes status variables that indicate the events for which the plugin has been called:

```
mysql> SHOW STATUS LIKE 'Audit_null%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
Audit_null_called	1385
Audit_null_general_error	1
Audit_null_general_log	741
Audit_null_general_result	643
+-----+-----+
```

`Audit_null_called` counts all events, and the other variables count instances of event subclasses. For example, the preceding `SHOW STATUS` statement causes the server to send a result to the client and to write a message to the general query log if that log is enabled. Thus, a client that issues the statement repeatedly causes `Audit_null_called` and `Audit_null_general_result` to be incremented each time, and `Audit_null_general_log` to be incremented if the log is enabled.

To disable the plugin after testing it, use this statement to unload it:

```
mysql> UNINSTALL PLUGIN NULL_AUDIT;
```

Audit Plugin Interface Changes

The audit plugin interface has undergone the following changes:

- In MySQL 5.5.5, the `event_subclass` member was added to the `mysql_event_general` structure. Corresponding to this change, the type-specific interface version, `MYSQL_AUDIT_INTERFACE_VERSION`, was increased from `0x0100` to `0x0200` and
- In MySQL 5.5.9, the `MYSQL_AUDIT_CONNECTION_CLASS` event class was added, and the `MYSQL_AUDIT_GENERAL_STATUS` subclass was added to the `MYSQL_AUDIT_GENERAL_CLASS` event class.
- In MySQL 5.5.14, the `event_class` member was removed from the `mysql_event_general` structure and the calling sequence for the notification function was changed. Originally, the second argument was a pointer to the event structure. The function now receives this information as two arguments: an event class number and a pointer to the event. Corresponding to these changes, `MYSQL_AUDIT_INTERFACE_VERSION` was increased to `0x0300`.

Each audit plugin must be compiled against MySQL source in which the value of `MYSQL_AUDIT_INTERFACE_VERSION` is appropriate for the interface version the plugin requires.

23.2.4.9. Writing Authentication Plugins

MySQL supports pluggable authentication, in which plugins are invoked to authenticate client connections. Authentication plugins enable the use of authentication methods other than the built-in method of passwords stored in the `mysql.user` table. For example, plugins can be written to access external authentication methods. Also, authentication plugins can support the proxy user capability, such that the connecting user is a proxy for another user and is treated, for purposes of access control, as having the privileges of a different user. For more information, see [Section 6.3.6, “Pluggable Authentication”](#), and [Section 6.3.7, “Proxy Users”](#).

An authentication plugin can be written for the server side or the client side. Server-side plugins use the same plugin API that is used for the other server plugin types such as full-text parser or audit plugins (although with a different type-specific descriptor). Client-side plugins use the client plugin API.

Several header files contain information relevant to authentication plugins:

- `plugin.h`: Defines the `MYSQL_AUTHENTICATION_PLUGIN` server plugin type.
- `client_plugin.h`: Defines the API for client plugins. This includes the client plugin descriptor and function prototypes for client plugin C API calls (see [Section 22.8.10, “C API Client Plugin Functions”](#)).
- `plugin_auth.h`: Defines the part of the server plugin API specific to authentication plugins. This includes the type-specific descriptor for server-side authentication plugins and the `MYSQL_SERVER_AUTH_INFO` structure.
- `plugin_auth_common.h`: Contains common elements of client and server authentication plugins. This includes return value definitions and the `MYSQL_PLUGIN_VIO` structure.

To write an authentication plugin, include the following header files in the plugin source file. Other MySQL or general header files might also be needed.

- For a source file that implements a server authentication plugin, include this file:

```
#include <mysql/plugin_auth.h>
```

- For a source file that implements a client authentication plugin, or both client and server plugins, include these files:

```
#include <mysql/plugin_auth.h>
#include <mysql/client_plugin.h>
```

```
#include <mysql>
```

`plugin_auth.h` includes `plugin.h` and `plugin_auth_common.h`, so you need not include the latter files explicitly.

This section describes how to write a pair of simple server and client authentication plugins that work together.



Warning

These plugins accept any non-empty password and the password is sent in clear text. This is insecure, so the plugins *should not be used in production environments*.

The server-side and client-side plugins developed here both are named `auth_simple`. As described in [Section 23.2.4.2, “Plugin Data Structures”](#), the plugin library file must have the same basename as the client plugin, so the source file name is `auth_simple.c` and produces a library named `auth_simple.so` (assuming that your system uses `.so` as the extension for library files).

In MySQL source distributions, authentication plugin source is located in the `plugin/auth` directory and can be examined as a guide to writing other authentication plugins. Also, to see how the built-in authentication plugins are implemented, see `sql/sql_acl.cc` for plugins that are built in to the MySQL server and `sql-common/client.c` for plugins that are built in to the `libmysql` client library. (For the built-in client plugins, note that the `auth_plugin_t` structures used there differ from the structures used with the usual client plugin declaration macros. In particular, the first two members are provided explicitly, not by declaration macros.)

23.2.4.9.1. Writing the Server-Side Authentication Plugin

Declare the server-side plugin with the usual general descriptor format that is used for all server plugin types (see [Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”](#)). For the `auth_simple` plugin, the descriptor looks like this:

```
mysql_declare_plugin(auth_simple)
{
    MYSQL_AUTHENTICATION_PLUGIN,
    &auth_simple_handler,           /* type-specific descriptor */
    "auth_simple",                 /* plugin name */
    "Author Name",                 /* author */
    "Any-password authentication plugin", /* description */
    PLUGIN_LICENSE_GPL,           /* license type */
    NULL,                          /* no init function */
    NULL,                          /* no deinit function */
    0x0100,                        /* version = 1.0 */
    NULL,                          /* no status variables */
    NULL,                          /* no system variables */
    NULL,                          /* no reserved information */
    0                               /* no flags */
}
mysql_declare_plugin_end;
```

The `name` member (`auth_simple`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

The `auth_simple_handler` member of the general descriptor points to the type-specific descriptor. For an authentication plugin, the type-specific descriptor is an instance of the `st_mysql_auth` structure (defined in `plugin_auth.h`):

```
struct st_mysql_auth
{
    int interface_version;
```

```
const char *client_auth_plugin;
int (*authenticate_user)(MYSQL_PLUGIN_VIO *vio, MYSQL_SERVER_AUTH_INFO *info);
};
```

The `st_mysql_auth` structure has three members: The type-specific API version number, the client plugin name, and a pointer to the main plugin function that communicates with the client. The `client_auth_plugin` member should indicate the name of the client plugin if a specific plugin is required. A value of `NULL` means “any plugin.” In the latter case, whatever plugin the client uses will do. This is useful if the server plugin does not care about the client plugin or what user name or password it sends. For example, this might be true if the server plugin authenticates only local clients and uses some property of the operating system rather than the information sent by the client plugin.

For `auth_simple`, the type-specific descriptor looks like this:

```
static struct st_mysql_auth auth_simple_handler =
{
    MYSQL_AUTHENTICATION_INTERFACE_VERSION,
    "auth_simple",          /* required client-side plugin name */
    auth_simple_server     /* server-side plugin main function */
};
```

The main function takes two arguments representing an I/O structure and a `MYSQL_SERVER_AUTH_INFO` structure. The structure definition is found in `plugin_auth.h`. It looks like this:

```
typedef struct st_mysql_server_auth_info
{
    char *user_name;
    unsigned int user_name_length;
    const char *auth_string;
    unsigned long auth_string_length;
    char authenticated_as[MYSQL_USERNAME_LENGTH+1];
    char external_user[512];
    int password_used;
    const char *host_or_ip;
    unsigned int host_or_ip_length;
} MYSQL_SERVER_AUTH_INFO;
```

The character set for string members is UTF-8. If there is a `_length` member associated with a string, it indicates the string length in bytes. Strings are also null-terminated.

When an authentication plugin is invoked by the server, it should interpret the `MYSQL_SERVER_AUTH_INFO` structure members as follows. Some of these are used to set the value of SQL functions or system variables within the client session, as indicated.

- `user_name`: The user name sent by the client. The value becomes the `USER()` [1197] function value.
- `user_name_length`: The length of `user_name` in bytes.
- `auth_string`: The value of the `authentication_string` column of the `mysql.user` table row for the matching account name (that is, the row that matches the client user name and host name and that the server uses to determine how to authenticate the client).

Suppose that you create an account using the following statement:

```
CREATE USER 'my_user'@'localhost'
    IDENTIFIED WITH my_plugin AS 'my_auth_string';
```

When `my_user` connects from the local host, the server invokes `my_plugin` and passes `'my_auth_string'` to it as the `auth_string` value.

- `auth_string_length`: The length of `auth_string` in bytes.

- `authenticated_as`: The server sets this to the user name (the value of `user_name`). The plugin can alter it to indicate that the client should have the privileges of a different user. For example, if the plugin supports proxy users, the initial value is the name of the connecting (proxy) user, and the plugin can change this member to the proxied user name. The server then treats the proxy user as having the privileges of the proxied user (assuming that the other conditions for proxy user support are satisfied; see [Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”](#)). The value is represented as a string at most `MYSQL_USER_NAME_LENGTH` bytes long, plus a terminating null. The value becomes the `CURRENT_USER() [1192]` function value.
- `external_user`: The server sets this to the empty string (null terminated). Its value becomes the `external_user [499]` system variable value. If the plugin wants that system variable to have a different value, it should set this member accordingly; for example, to the connecting user name. The value is represented as a string at most 511 bytes long, plus a terminating null.
- `password_used`: This member applies when authentication fails. The plugin can set it or ignore it. The value is used to construct the failure error message of `Authentication fails. Password used: %s`. The value of `password_used` determines how `%s` is handled, as shown in the following table.

| <code>password_used</code> | <code>%s</code> Handling |
|----------------------------|----------------------------------|
| 0 | NO |
| 1 | YES |
| 2 | There will be no <code>%s</code> |

- `host_or_ip`: The name of the client host if it can be resolved, or the IP address otherwise.
- `host_or_ip_length`: The length of `host_or_ip` in bytes.

The `auth_simple` main function, `auth_simple_server()`, reads the password (a null-terminated string) from the client and succeeds if the password is nonempty (first byte not null):

```
static int auth_simple_server (MYSQL_PLUGIN_VIO *vio,
                              MYSQL_SERVER_AUTH_INFO *info)
{
    unsigned char *pkt;
    int pkt_len;

    /* read the password as null-terminated string, fail on error */
    if ((pkt_len= vio->read_packet(vio, &pkt)) < 0)
        return CR_ERROR;

    /* fail on empty password */
    if (!pkt_len || *pkt == '\0')
    {
        info->password_used= PASSWORD_USED_NO;
        return CR_ERROR;
    }

    /* accept any nonempty password */
    info->password_used= PASSWORD_USED_YES;

    return CR_OK;
}
```

The main function should return one of the error codes shown in the following table.

| Error Code | Meaning |
|--------------------|---------|
| <code>CR_OK</code> | Success |

| Error Code | Meaning |
|--|--|
| CR_OK_HANDSHAKE_COMPLETE | Do not send a status packet back to client |
| CR_ERROR | Error |

For an example of how the handshake works, see the [plugin/auth/dialog.c](#) source file.

[auth_simple_server_main\(\)](#) is so basic that it does not use the authentication information structure except to set the member that indicates whether a password was received.

A plugin that supports proxy users must return to the server the name of the proxied user (the MySQL user whose privileges the client user should get). To do this, the plugin must set the [info->authenticated_as](#) member to the proxied user name. For information about proxying, see [Section 6.3.7, “Proxy Users”](#), and [Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”](#).

23.2.4.9.2. Writing the Client-Side Authentication Plugin

Declare the client-side plugin descriptor with the [mysql_declare_client_plugin\(\)](#) and [mysql_end_client_plugin](#) macros (see [Section 23.2.4.2.3, “Client Plugin Descriptors”](#)). For the [auth_simple](#) plugin, the descriptor looks like this:

```
mysql_declare_client_plugin(AUTHENTICATION)
  "auth_simple",          /* plugin name */
  "Author Name",         /* author */
  "Any-password authentication plugin", /* description */
  {1,0,0},               /* version = 1.0.0 */
  "GPL",                 /* license type */
  NULL,                  /* for internal use */
  NULL,                  /* no init function */
  NULL,                  /* no deinit function */
  NULL,                  /* no option-handling function */
  auth_simple_client     /* main function */
mysql_end_client_plugin;
```

The descriptor members common to all client plugins are the plugin name through the option-handling function. Following the common members, an authentication plugin also has one extra member. This is the “main” function, which handles communication with the server. The function takes two arguments representing an I/O structure and a connection handler. For our simple any-password plugin, the main function does nothing but write to the server the password provided by the user:

```
static int auth_simple_client (MYSQL_PLUGIN_VIO *vio, MYSQL *mysql)
{
  int res;

  /* send password as null-terminated string in clear text */
  res= vio->write_packet(vio, (const unsigned char *) mysql->passwd,
                          strlen(mysql->passwd) + 1);

  return res ? CR_ERROR : CR_OK;
}
```

The main function should return one of the error codes shown in the following table.

| Error Code | Meaning |
|--|----------------------|
| CR_OK | Success |
| CR_OK_HANDSHAKE_COMPLETE | Success, client done |

| Error Code | Meaning |
|------------|---------|
| CR_ERROR | Error |

`CR_OK_HANDSHAKE_COMPLETE` indicates that the client has done its part successfully and has read the last packet. A client plugin may return `CR_OK_HANDSHAKE_COMPLETE` if the number of round trips in the authentication protocol is not known in advance and the plugin must read another packet to determine whether authentication is finished.

23.2.4.9.3. Using the Authentication Plugins

To compile and install a plugin library object file, see the instructions in [Section 23.2.4.3, “Compiling and Installing Plugin Libraries”](#). To use the library file, it must be installed in the plugin directory (the directory named by the `plugin_dir` [537] system variable).

Register the server-side plugin with the server. For example, to load the plugin at server startup, use a `--plugin-load=auth_simple.so` [457] option (change the library extension as necessary for your system).

Create a user for whom the server will use the `auth_simple` plugin for authentication:

```
mysql> CREATE USER 'x'@'localhost'
-> IDENTIFIED WITH auth_simple;
```

Use a client program to connect to the server as user `x`. The server-side `auth_simple` plugin communicates with the client program that it should use the client-side `auth_simple` plugin, and the latter sends the password to the server. The server plugin should reject connections that send an empty password and accept connections that send a nonempty password. Invoke the client program each way to verify this:

```
shell> mysql --user=x --skip-password
ERROR 1045 (28000): Access denied for user 'x'@'localhost' (using password: NO)

shell> mysql --user=x --password=abc
mysql>
```

Because the server plugin accepts any nonempty password, it should be considered insecure. After testing the plugin to verify that it works, restart the server without the `--plugin-load` [457] option so as not to inadvertently leave the server running with an insecure authentication plugin loaded. Also, drop the user with `DROP USER 'x'@'localhost'`.

For additional information about loading and using authentication plugins, see [Section 5.1.8.1, “Installing and Uninstalling Plugins”](#), and [Section 6.3.6, “Pluggable Authentication”](#).

If you are writing a client program that supports the use of authentication plugins, normally such a program causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

Should a client program require lower-level plugin management, the client library contains functions that take an `st_mysql_client_plugin` argument. See [Section 22.8.10, “C API Client Plugin Functions”](#).

23.2.4.9.4. Implementing Proxy User Support in Authentication Plugins

One of the capabilities that pluggable authentication makes possible is proxy users (see [Section 6.3.7, “Proxy Users”](#)). For a server-side authentication plugin to participate in proxy user support, these conditions must be satisfied:

- When a connecting client should be treated as a proxy user, the plugin must return a different name in the `authenticated_as` member of the `MYSQL_SERVER_AUTH_INFO` structure, to indicate the proxied user name. It may also optionally set the `external_user` member, to set the value of the `external_user` [499] system variable.
- Proxy user accounts must be set up to be authenticated by the plugin. Use the `CREATE USER` or `GRANT` statement to associate accounts with plugins.
- Proxy user accounts must have the `PROXY` [696] privilege for the proxied accounts. Use the `GRANT` statement to grant this privilege.

In other words, the only aspect of proxy user support required of the plugin is that it set `authenticated_as` to the proxied user name. The rest is optional (setting `external_user`) or done by the DBA using SQL statements.

How does an authentication plugin determine which proxied user to return when the proxy user connects? That depends on the plugin. Typically, the plugin maps clients to proxied users based on the authentication string passed to it by the server. This string comes from the `AS` part of the `IDENTIFIED WITH` clause of the `CREATE USER` statement that specifies use of the plugin for authentication.

The plugin developer determines the syntax rules for the authentication string and implements the plugin according to those rules. Suppose that a plugin takes a comma-separated list of pairs that map external users to MySQL users. For example:

```
CREATE USER '@%.example.com'
  IDENTIFIED WITH my_plugin AS 'extuser1=mysqlusera, extuser2=mysqluserb'
CREATE USER '@%.example.org'
  IDENTIFIED WITH my_plugin AS 'extuser1=mysqluserc, extuser2=mysqluserd'
```

When the server invokes a plugin to authenticate a client, it passes the appropriate authentication string to the plugin. The plugin is responsible to:

1. Parse the string into its components to determine the mapping to use
2. Compare the client user name to the mapping
3. Return the proper MySQL user name

For example, if `extuser2` connects from an `example.com` host, the server passes `'extuser1=mysqlusera, extuser2=mysqluserb'` to the plugin, and the plugin should copy `mysqluserb` into `authenticated_as`, with a terminating null byte. If `extuser2` connects from an `example.org` host, the server passes `'extuser1=mysqluserc, extuser2=mysqluserd'`, and the plugin should copy `mysqluserd` instead.

If there is no match in the mapping, the action depends on the plugin. If a match is required, the plugin likely will return an error. Or the plugin might simply return the client name; in this case, it should not change `authenticated_as`, and the server will not treat the client as a proxy.

The following example demonstrates how to handle proxy users using a plugin named `auth_simple_proxy`. Like the `auth_simple` plugin described earlier, `auth_simple_proxy` accepts any nonempty password as valid (and thus should not be used in production environments). In addition, it examines the `auth_string` authentication string member and uses these very simple rules for interpreting it:

- If the string is empty, the plugin returns the user name as given and no proxying occurs. That is, the plugin leaves the value of `authenticated_as` unchanged.
- If the string is nonempty, the plugin treats it as the name of the proxied user and copies it to `authenticated_as` so that proxying occurs.

For testing, set up one account that is not proxied according to the preceding rules, and one that is. This means that one account has no `AS` clause, and one includes an `AS` clause that names the proxied user:

```
CREATE USER 'plugin_user1'@'localhost'
  IDENTIFIED WITH auth_simple_proxy;
CREATE USER 'plugin_user2'@'localhost'
  IDENTIFIED WITH auth_simple_proxy AS 'proxied_user';
```

In addition, create an account for the proxied user and grant `plugin_user2` the `PROXY [696]` privilege for it:

```
CREATE USER 'proxied_user'@'localhost'
  IDENTIFIED BY 'proxied_user_pass';
GRANT PROXY
  ON 'proxied_user'@'localhost'
  TO 'plugin_user2'@'localhost';
```

Before the server invokes an authentication plugin, it sets `authenticated_as` to the client user name. To indicate that the user is a proxy, the plugin should set `authenticated_as` to the proxied user name. For `auth_simple_proxy`, this means that it must examine the `auth_string` value, and, if the value is nonempty, copy it to the `authenticated_as` member to return it as the name of the proxied user. In addition, when proxying occurs, the plugin sets the `external_user` member to the client user name; this becomes the value of the `external_user [499]` system variable.

```
static int auth_simple_proxy_server (MYSQL_PLUGIN_VIO *vio,
                                     MYSQL_SERVER_AUTH_INFO *info)
{
  unsigned char *pkt;
  int pkt_len;

  /* read the password as null-terminated string, fail on error */
  if ((pkt_len= vio->read_packet(vio, &pkt)) < 0)
    return CR_ERROR;

  /* fail on empty password */
  if (!pkt_len || *pkt == '\0')
  {
    info->password_used= PASSWORD_USED_NO;
    return CR_ERROR;
  }

  /* accept any nonempty password */
  info->password_used= PASSWORD_USED_YES;

  /* if authentication string is nonempty, use as proxied user name */
  /* and use client name as external_user value */
  if (info->auth_string_length > 0)
  {
    strcpy (info->authenticated_as, info->auth_string);
    strcpy (info->external_user, info->user_name);
  }
}
```

```

}
return CR_OK;
}

```

After a successful connection, the `USER()` [1197] function should indicate the connecting client user and host name, and `CURRENT_USER()` [1192] should indicate the account whose privileges apply during the session. The latter value should be the connecting user account if no proxying occurs or the proxied account if proxying does occur.

Compile and install the plugin, then test it. First, connect as `plugin_user1`:

```
shell> mysql --user=plugin_user1 --password=x
```

In this case, there should be no proxying:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user, @@external_user\G
***** 1. row *****
      USER(): plugin_user1@localhost
    CURRENT_USER(): plugin_user1@localhost
      @@proxy_user: NULL
    @@external_user: NULL

```

Then connect as `plugin_user2`:

```
shell> mysql --user=plugin_user2 --password=x
```

In this case, `plugin_user2` should be proxied to `proxied_user`:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user, @@external_user\G
***** 1. row *****
      USER(): plugin_user2@localhost
    CURRENT_USER(): proxied_user@localhost
      @@proxy_user: 'plugin_user2'@'localhost'
    @@external_user: 'plugin_user2'@'localhost'

```

23.2.5. MySQL Services for Plugins

MySQL server plugins have access to server “services.” The services interface exposes server functionality that plugins can call. It complements the plugin API and has these characteristics:

- Services enable plugins to access code inside the server using ordinary function calls.
- Services are portable and work on multiple platforms.
- The interface includes a versioning mechanism so that service versions supported by the server can be checked at load time against plugin versions. Versioning protects against incompatibilities between the version of a service that the server provides and the version of the service expected or required by a plugin.

Current services include the following, and others can be implemented:

- `my_snprintf`: A string-formatting service that produces consistent results across platforms.
- `my_thd_scheduler`: A service for plugins to select a thread scheduler.
- `thd_alloc`: A memory-allocation service.
- `thd_wait`: A service for plugins to report when they are going to sleep or stall.

The plugin services interface differs from the plugin API as follows:

- The plugin API enables plugins to be used by the server. The calling initiative lies with the server to invoke plugins. This enables plugins to extend server functionality or register to receive notifications about server processing.
- The plugin services interface enables plugins to call code inside the server. The calling initiative lies with plugins to invoke service functions. This enables functionality already implemented in the server to be used by many plugins; they need not individually implement it themselves.

For developers who wish to modify the server to add a new service, see [MySQL Services for Plugins](#).

The remainder of this section describes how a plugin uses server functionality that is available as a service. See also the source for the “daemon” example plugin, which uses the `my_snprintf` service. Within a MySQL source distribution, that plugin is located in the `plugin/daemon_example` directory.

To determine what services exist and what functions they provide, look in the `include/mysql` directory of a MySQL source distribution. The relevant files are:

- `plugin.h` includes `services.h`.
- `services.h` is the “umbrella” header that includes all available service-specific header files.
- Service-specific headers have names like `service_my_snprintf.h` or `service_thd_alloc.h`.

Each service-specific header should contain comments that provide full usage documentation for a given service, including what service functions are available, their calling sequences, and return values.

To use a service or services from within a plugin, the plugin source file must include the `plugin.h` header file to access service-related information:

```
#include <mysql/plugin.h>
```

This does not represent any additional setup cost. A plugin must include that file anyway because it contains definitions and structures that every plugin needs.

To access a service, a plugin calls service functions like any other function. For example, to format a string into a buffer for printing, call the `my_snprintf()` function provided by the service of the same name:

```
char buffer[BUFFER_SIZE];

my_snprintf(buffer, sizeof(buffer), format_string, argument_to_format, ...);
```

When you build your plugin, you must link in the `libmysqservices` library. Use the `-lmysqservices` flag at link time. For example, for `CMake`, put this in the top-level `CMakeLists.txt` file:

```
FIND_LIBRARY(MYSQLSERVICES_LIB mysqlservices
  PATHS "${MYSQL_SRCDIR}/libservices" NO_DEFAULT_PATH)
```

Put this in the `CMakeLists.txt` file in the directory containing the plugin source:

```
# the plugin needs the mysql services library for error logging
TARGET_LINK_LIBRARIES (your_plugin_library_name ${MYSQLSERVICES_LIB})
```

23.3. Adding New Functions to MySQL

There are three ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as object files and then added to and removed from the server dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See [Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”](#).
- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the `mysqld` server and become available on a permanent basis.
- Another way to add functions is by creating stored functions. These are written using SQL statements rather than by compiling object code. The syntax for writing stored functions is not covered here. See [Section 19.2, “Using Stored Routines \(Procedures and Functions\)”](#).

Each method of creating compiled functions has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you don't need to do that.
- Native functions require you to modify a source distribution. UDFs do not. You can add UDFs to a binary MySQL distribution. No access to MySQL source is necessary.
- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as `ABS()` [1123] or `SOUNDEX()` [1108].

See [Section 9.2.4, “Function Name Parsing and Resolution”](#), for the rules describing how the server interprets references to different kinds of functions.

The following sections describe features of the UDF interface, provide instructions for writing UDFs, discuss security precautions that MySQL takes to prevent UDF misuse, and describe how to add native MySQL functions.

For example source code that illustrates how to write UDFs, take a look at the `sql/udf_example.c` file that is provided in MySQL source distributions.

23.3.1. Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilities:

- Functions can return string, integer, or real values and can accept arguments of those same types.
- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number, types, and names of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.
- You can indicate that a function returns `NULL` or that an error occurred.

23.3.2. Adding a New User-Defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. MySQL source distributions include a file `sql/udf_example.c` that defines 5

new functions. Consult this file to see how UDF calling conventions work. The `include/mysql_com.h` header file defines UDF-related symbols and data structures. (You need not include this header file directly because it is included by `mysql.h`.)

A UDF contains code that becomes part of the running server, so when you write a UDF, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to UDFs that were originally written for older servers. For information about these constraints, see [Section 2.10.4, “MySQL Source-Configuration Options”](#), and [Section 2.10.5, “Dealing with Problems Compiling MySQL”](#).

To be able to use UDFs, you need to link `mysqld` dynamically. If you want to use a UDF that needs to access symbols from `mysqld` (for example, the `metaphone` function in `sql/udf_example.c` that uses `default_charset_info`), you must link the program with `-rdynamic` (see `man dlopen`).

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the following discussion, the name “xxx” is used for an example function name. To distinguish between SQL and C/C++ usage, `XXX()` (uppercase) indicates an SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.



Note

When using C++ you can encapsulate your C functions within:

```
extern "C" { ... }
```

This will ensure that your C++ function names remain readable in the completed UDF.

The C/C++ functions that you write to implement the interface for `XXX()` are:

- `xxx()` (required)

The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here.

| SQL Type | C/C++ Type |
|----------|------------|
| STRING | char * |
| INTEGER | long long |
| REAL | double |

It is also possible to declare a `DECIMAL` function, but currently the value is returned as a string, so you should write the UDF as though it were a `STRING` function. `ROW` functions are not implemented.

- `xxx_init()` (optional)

The initialization function for `xxx()`. It can be used for the following purposes:

- To check the number of arguments to `xxx()`.
- To check that the arguments are of a required type or, alternatively, to tell MySQL to coerce arguments to the types you want when the main function is called.
- To allocate any memory required by the main function.
- To specify the maximum length of the result.

- To specify (for `REAL` functions) the maximum number of decimal places in the result.
- To specify whether the result can be `NULL`.
- `xxx_deinit()` (optional)

The deinitialization function for `xxx()`. It should deallocate any memory allocated by the initialization function.

When an SQL statement invokes `xxx()`, MySQL calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, MySQL aborts the SQL statement with an error message and does not call the main or deinitialization functions. Otherwise, MySQL calls the main function `xxx()` once for each row. After all rows have been processed, MySQL calls the deinitialization function `xxx_deinit()` so that it can perform any required cleanup.

For aggregate functions that work like `SUM()` [1206], you must also provide the following functions:

- `xxx_clear()`

Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- `xxx_add()`

Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call `xxx_init()` to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the `GROUP BY` expression.
3. Call `xxx_clear()` for the first row in each new group.
4. Call `xxx_add()` for each row that belongs in the same group.
5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.
6. Repeat steps 3 to 5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A consequence of this requirement is that you are not permitted to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

23.3.2.1. UDF Calling Sequences for Simple Functions

This section describes the different functions that you need to define when you create a simple UDF. [Section 23.3.2, “Adding a New User-Defined Function”](#), describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `xxx()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For **STRING** functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
         char *result, unsigned long *length,
         char *is_null, char *error);
```

For **INTEGER** functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
             char *is_null, char *error);
```

For **REAL** functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *is_null, char *error);
```

DECIMAL functions return string values and should be declared the same way as **STRING** functions. **ROW** functions are not implemented.

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- `my_bool maybe_null`

`xxx_init()` should set `maybe_null` to 1 if `xxx()` can return `NULL`. The default value is 1 if any of the arguments are declared `maybe_null`.

- `unsigned int decimals`

The number of decimal digits to the right of the decimal point. The default value is the maximum number of decimal digits in the arguments passed to the main function. For example, if the function is passed 1.34, 1.345, and 1.3, the default would be 3, because 1.345 has 3 decimal digits.

For arguments that have no fixed number of decimals, the `decimals` value is set to 31, which is 1 more than the maximum number of decimals permitted for the `DECIMAL`, `FLOAT`, and `DOUBLE` data types. As of MySQL 5.5.3, this value is available as the constant `NOT_FIXED_DEC` in the `mysql_com.h` header file.

A `decimals` value of 31 is used for arguments in cases such as a `FLOAT` or `DOUBLE` column declared without an explicit number of decimals (for example, `FLOAT` rather than `FLOAT(10,3)`) and for floating-point constants such as `1345E-3`. It is also used for string and other nonnumber arguments that might be converted within the function to numeric form.

The value to which the `decimals` member is initialized is only a default. It can be changed within the function to reflect the actual calculation performed. The default is determined such that the largest number of decimals of the arguments is used. If the number of decimals is `NOT_FIXED_DEC` for even one of the arguments, that is the value used for `decimals`.

- `unsigned int max_length`

The maximum length of the result. The default `max_length` value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions,

the default is 21 digits. For real functions, the default is 13 plus the number of decimal digits indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

If you want to return a blob value, you can set `max_length` to 65KB or 16MB. This memory is not allocated, but the value is used to decide which data type to use if there is a need to temporarily store the data.

- `char *ptr`

A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory among themselves. `xxx_init()` should allocate the memory and assign it to this pointer:

```
initid->ptr = allocated_memory;
```

In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

- `my_bool const_item`

`xxx_init()` should set `const_item` to 1 if `xxx()` always returns the same value and to 0 otherwise.

23.3.2.2. UDF Calling Sequences for Aggregate Functions

This section describes the different functions that you need to define when you create an aggregate UDF. [Section 23.3.2, “Adding a New User-Defined Function”](#), describes the order in which MySQL calls these functions.

- `xxx_reset()`

This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given `UDF_ARGS` argument as the first value in your internal summary value for the group. Declare `xxx_reset()` as follows:

```
void xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

`xxx_reset()` is not needed or used in MySQL 5.5, in which the UDF interface uses `xxx_clear()` instead. However, you can define both `xxx_reset()` and `xxx_clear()` if you want to have your UDF work with older versions of the server. (If you do include both functions, the `xxx_reset()` function in many cases can be implemented internally by calling `xxx_clear()` to reset all variables, and then calling `xxx_add()` to add the `UDF_ARGS` argument as the first value in the group.)

- `xxx_clear()`

This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare `xxx_clear()` as follows:

```
void xxx_clear(UDF_INIT *initid, char *is_null, char *error);
```

`is_null` is set to point to `CHAR(0)` before calling `xxx_clear()`.

If something went wrong, you can store a value in the variable to which the `error` argument points. `error` points to a single-byte variable, not to a string buffer.

`xxx_clear()` is required by MySQL 5.5.

- `xxx_add()`

This function is called for all rows that belong to the same group. You should use it to add the value in the `UDF_ARGS` argument to your internal summary variable.

```
void xxx_add(UDF_INIT *initid, UDF_ARGS *args,
            char *is_null, char *error);
```

The `xxx()` function for an aggregate UDF should be declared the same way as for a nonaggregate UDF. See [Section 23.3.2.1, “UDF Calling Sequences for Simple Functions”](#).

For an aggregate UDF, MySQL calls the `xxx()` function after all rows in the group have been processed. You should normally never access its `UDF_ARGS` argument here but instead return a value based on your internal summary variables.

Return value handling in `xxx()` should be done the same way as for a nonaggregate UDF. See [Section 23.3.2.4, “UDF Return Values and Error Handling”](#).

The `xxx_reset()` and `xxx_add()` functions handle their `UDF_ARGS` argument the same way as functions for nonaggregate UDFs. See [Section 23.3.2.3, “UDF Argument Processing”](#).

The pointer arguments to `is_null` and `error` are the same for all calls to `xxx_reset()`, `xxx_clear()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or whether the `xxx()` function should return `NULL`. You should not store a string into `*error`! `error` points to a single-byte variable, not to a string buffer.

`*is_null` is reset for each group (before calling `xxx_clear()`). `*error` is never reset.

If `*is_null` or `*error` are set when `xxx()` returns, MySQL returns `NULL` as the result for the group function.

23.3.2.3. UDF Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

- `unsigned int arg_count`

The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

```
if (args->arg_count != 2)
{
    strcpy(message, "XXX() requires two arguments");
    return 1;
}
```

For other `UDF_ARGS` member values that are arrays, array references are zero-based. That is, refer to array members using index values from 0 to `args->arg_count - 1`.

- `enum Item_result *arg_type`

A pointer to an array containing the types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT`, `REAL_RESULT`, and `DECIMAL_RESULT`.

To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
```

```
strcpy(message, "XXX() requires a string and an integer");
return 1;
}
```

Arguments of type `DECIMAL_RESULT` are passed as strings, so you should handle them the same way as `STRING_RESULT` values.

As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes MySQL to coerce arguments to those types for each call to `xxx()`. For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

Exact-value decimal arguments such as `1.3` or `DECIMAL` column values are passed with a type of `DECIMAL_RESULT`. However, the values are passed as strings. If you want to receive a number, use the initialization function to specify that the argument should be coerced to a `REAL_RESULT` value:

```
args->arg_type[2] = REAL_RESULT;
```

- `char **args`

`args->args` communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument `i`, `args->args[i]` points to the argument value. (See later for instructions on how to access the value properly.) For a nonconstant argument, `args->args[i]` is 0. A constant argument is an expression that uses only constants, such as `3` or `4*7-2` or `SIN(3.14)` [1130]. A nonconstant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with nonconstant arguments.

For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

If argument `i` represents `NULL`, `args->args[i]` is a null pointer (0). If the argument is not `NULL`, functions can refer to it as follows:

- An argument of type `STRING_RESULT` is given as a string pointer plus a length, to enable handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. Do not assume that the string is null-terminated.
- For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

```
long long int_val;
int_val = *((long long*) args->args[i]);
```

- For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

```
double real_val;
real_val = *((double*) args->args[i]);
```

- For an argument of type `DECIMAL_RESULT`, the value is passed as a string and should be handled like a `STRING_RESULT` value.
 - `ROW_RESULT` arguments are not implemented.
- `unsigned long *lengths`

For the initialization function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual

lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

- `char *maybe_null`

For the initialization function, the `maybe_null` array indicates for each argument whether the argument value might be null (0 if no, 1 if yes).

- `char **attributes`

`args->attributes` communicates information about the names of the UDF arguments. For argument `i`, the attribute name is available as a string in `args->attributes[i]` and the attribute length is `args->attribute_lengths[i]`. Do not assume that the string is null-terminated.

By default, the name of a UDF argument is the text of the expression used to specify the argument. For UDFs, an argument may also have an optional `[AS] alias_name` clause, in which case the argument name is `alias_name`. The `attributes` value for each argument thus depends on whether an alias was given.

Suppose that a UDF `my_udf()` is invoked as follows:

```
SELECT my_udf(expr1, expr2 AS alias1, expr3 alias2);
```

In this case, the `attributes` and `attribute_lengths` arrays will have these values:

```
args->attributes[0] = "expr1"
args->attribute_lengths[0] = 5

args->attributes[1] = "alias1"
args->attribute_lengths[1] = 6

args->attributes[2] = "alias2"
args->attribute_lengths[2] = 6
```

- `unsigned long *attribute_lengths`

The `attribute_lengths` array indicates the length of each argument name.

23.3.2.4. UDF Return Values and Error Handling

The initialization function should return 0 if no error occurred and 1 otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message is returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. A string function should return a pointer to the result and set `*length` to the length (in bytes) of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

MySQL passes a buffer to the `xxx()` function using the `result` parameter. This buffer is sufficiently long to hold 255 characters, which can be multi-byte characters. The `xxx()` function can store the result in this buffer if it fits, in which case the return value should be a pointer to the buffer. If the function stores the result in a different buffer, it should return a pointer to that buffer.

If your string function does not use the supplied buffer (for example, if it needs to return a string longer than 255 characters), you must allocate the space for your own buffer with `malloc()` in your `xxx_init()`

function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See [Section 23.3.2.1, “UDF Calling Sequences for Simple Functions”](#).

To indicate a return value of `NULL` in the main function, set `*is_null` to 1:

```
*is_null = 1;
```

To indicate an error return in the main function, set `*error` to 1:

```
*error = 1;
```

If `xxx()` sets `*error` to 1 for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` is not even called for subsequent rows.)

23.3.2.5. Compiling and Installing User-Defined Functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `sql/udf_example.c` that is included in MySQL source distributions.

If a UDF will be referred to in statements that will be replicated to slave servers, you must ensure that every slave also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.c` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it is more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP address for a host name.
- `reverse_lookup()` returns the host name for an IP address. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.
- `avgcost()` returns an average cost. This is an aggregate function.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so udf_example.c
```

If you are using `gcc` with `CMake` (which is how MySQL is configured), you should be able to create `udf_example.so` with a simpler command:

```
shell> make udf_example
```

After you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.c` using `gcc` directly produces a file named `udf_example.so`. Copy the shared object to the server's plugin directory and name it `udf_example.so`. This directory is given by the value of the `plugin_dir` [537] system variable.

On some systems, the `ldconfig` program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, you can compile user-defined functions by using the following procedure:

1. Obtain a MySQL source distribution. See [Section 2.1.3, "How to Get MySQL"](#).
2. Obtain the `CMake` build utility, if necessary, from <http://www.cmake.org>. (Version 2.6 or later is required).
3. In the source tree, look in the `sql` directory. There are files named `udf_example.def` and `udf_example.c` there. Copy both files from this directory to your working directory.
4. Create a `CMake` makefile (`CMakeLists.txt`) with these contents:

```
PROJECT(udf_example)

# Path for MySQL include directory
INCLUDE_DIRECTORIES("c:/mysql/include")

ADD_DEFINITIONS("-DHAVE_DLOPEN")
ADD_LIBRARY(udf_example MODULE udf_example.c udf_example.def)
TARGET_LINK_LIBRARIES(udf_example wsock32)
```

5. Create the VC project and solution files:

```
cmake -G "<Generator>"
```

Invoking `cmake --help` shows you a list of valid Generators.

6. Create `udf_example.dll`:

```
devenv udf_example.sln /build Release
```

After the shared object file has been installed, notify `mysqld` about the new functions with the following statements. If object files have a suffix different from `.so` on your system, substitute the correct suffix throughout (for example, `.dll` on Windows).

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION sequence RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION reverse_lookup
-> RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE AGGREGATE FUNCTION avgcost
-> RETURNS REAL SONAME 'udf_example.so';
```

To delete functions, use `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION sequence;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
```

```
mysql> DROP FUNCTION avgcost;
```

The `CREATE FUNCTION` and `DROP FUNCTION` statements update the `func` system table in the `mysql` database. The function's name, type and shared library name are saved in the table. You must have the `INSERT [696]` or `DELETE [696]` privilege for the `mysql` database to create or drop functions, respectively.

You should not use `CREATE FUNCTION` to add a function that has previously been created. If you need to reinstall a function, you should remove it with `DROP FUNCTION` and then reinstall it with `CREATE FUNCTION`. You would need to do this, for example, if you recompile a new version of your function, so that `mysqld` gets the new version. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables [461]` option. In this case, UDF initialization is skipped and UDFs are unavailable.

23.3.2.6. User-Defined Function Security Precautions

MySQL takes the following measures to prevent misuse of user-defined functions.

You must have the `INSERT [696]` privilege to be able to use `CREATE FUNCTION` and the `DELETE [696]` privilege to be able to use `DROP FUNCTION`. This is necessary because these statements add and delete rows from the `mysql.func` table.

UDFs should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. `mysqld` also supports an `--allow-suspicious-udfs [434]` option that controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared object files other than those containing legitimate UDFs. If you have older UDFs that contain only the `xxx` symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the `--allow-suspicious-udfs [434]` option. Otherwise, you should avoid enabling this capability.

UDF object files cannot be placed in arbitrary directories. They must be located in the server's plugin directory. This directory is given by the value of the `plugin_dir [537]` system variable.

23.3.3. Adding a New Native Function

To add a new native MySQL function, use the procedure described here, which requires that you use a source distribution. You cannot add native functions to a binary distribution because it is necessary to modify MySQL source code and compile MySQL from the modified source. If you migrate to another version of MySQL (for example, when a new version is released), you must repeat the procedure with the new version.

If the new native function will be referred to in statements that will be replicated to slave servers, you must ensure that every slave server also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

To add a new native function, follow these steps to modify source files in the `sql` directory:

1. Create a subclass for the function in `item_create.cc`:
 - If the function takes a fixed number of arguments, create a subclass of `Create_func_arg0`, `Create_func_arg1`, `Create_func_arg2`, or `Create_func_arg3`, respectively, depending on whether the function takes zero, one, two, or three arguments. For examples, see the `Create_func_uuid`, `Create_func_abs`, `Create_func_pow`, and `Create_func_lpad` classes.

- If the function takes a variable number of arguments, create a subclass of `Create_native_func`. For an example, see `Create_func_concat`.
2. To provide a name by which the function can be referred to in SQL statements, register the name in `item_create.cc` by adding a line to this array:

```
static Native_func_registry func_array[]
```

You can register several names for the same function. For example, see the lines for "LCASE" and "LOWER", which are aliases for `Create_func_lcase`.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.
4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

```
double    Item_func_newname::val()
longlong Item_func_newname::val_int()
String    *Item_func_newname::Str(String *str)
```

If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. If the function is nondeterministic, include the following statement in the item constructor to indicate that function results should not be cached:

```
current_thd->lex->safe_to_cache_query=0;
```

A function is nondeterministic if, given fixed values for its arguments, it can return different results for different invocations.

6. You should probably also define the following object function:

```
void Item_func_newname::fix_length_and_dec()
```

This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments' `maybe_null` variable. Look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe. In other words, do not use any global or static variables in the functions without protecting them with mutexes.

If you want to return `NULL` from `::val()`, `::val_int()`, or `::str()`, you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the `sql_string.h` file.)
- The `::str()` function should return the string that holds the result, or `(char*) 0` if the result is `NULL`.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

23.4. Debugging and Porting MySQL

This appendix helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See [Section 2.1.1, “Operating Systems Supported by MySQL Community Server”](#). If you have created a new port of MySQL, please let us know so that we can list it here and on our Web site (<http://www.mysql.com/>), recommending it to other users.



Note

If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copyright holder of MySQL.

A working POSIX thread library is needed for the server.

To build MySQL from source, your system must satisfy the tool requirements listed at [Section 2.10, “Installing MySQL from Source”](#).



Important

If you are trying to build MySQL 5.5 with `icc` on the IA64 platform, and need support for MySQL Cluster, you should first ensure that you are using `icc` version 9.1.043 or later. (For details, see Bug #21875.)

If you run into problems with a new port, you may have to do some debugging of MySQL! See [Section 23.4.1, “Debugging a MySQL Server”](#).



Note

Before you start debugging `mysqld`, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This ensures that your thread installation has even a remote chance to work!

23.4.1. Debugging a MySQL Server

If you are using some functionality that is very new in MySQL, you can try to run `mysqld` with the `--skip-new` (which disables all new, potentially unsafe functionality). See [Section C.5.4.2, “What to Do If MySQL Keeps Crashing”](#).

If `mysqld` doesn't want to start, you should verify that you don't have any `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`

If `mysqld` starts to eat up CPU or memory or if it “hangs,” you can use `mysqladmin processlist status` to find out if someone is executing a query that takes a long time. It may be a good idea to run `mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients can't connect.

The command `mysqladmin debug` dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or `myisamchk`. See [Chapter 5, MySQL Server Administration](#). You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See [Section 2.1, “General Installation Guidance”](#).

23.4.1.1. Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `-DWITH_DEBUG=1` [155] option. You can check whether MySQL was compiled with debugging by doing: `mysqld --help`. If the `--debug` [439] flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql ... --debug` in this case.

If `mysqld` stops crashing when you compile it with `-DWITH_DEBUG=1` [155], you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add `-g` to the `CFLAGS` and `CXXFLAGS` environment variables and not use `-DWITH_DEBUG=1` [155]. If `mysqld` dies, you can at least attach to it with `gdb` or use `gdb` on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something “unexpected,” an entry is written to `stderr`, which `mysqld_safe` directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging! (The second thing is to send mail to a MySQL mailing list and ask for help. See Section 1.6.1, “MySQL Mailing Lists”. If you believe that you have found a bug, please use the instructions at Section 1.7, “How to Report Bugs or Problems”.

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

23.4.1.2. Creating Trace Files

If the `mysqld` server doesn't start or if you can cause it to crash quickly, you can try to create a trace file to find the problem.

To do this, you must have a `mysqld` that has been compiled with debugging support. You can check this by executing `mysqld -v`. If the version number ends with `-debug`, it is compiled with support for trace files. (On Windows, the debugging server is named `mysqld-debug` rather than `mysqld` as of MySQL 4.1.)

Start the `mysqld` server with a trace log in `/tmp/mysqld.trace` on Unix or `C:\mysqld.trace` on Windows:

```
shell> mysqld --debug
```

On Windows, you should also use the `--standalone` [462] flag to not start `mysqld` as a service. In a console window, use this command:

```
C:\> mysqld-debug --debug --standalone
```

After this, you can use the `mysql.exe` command-line tool in a second console window to reproduce the problem. You can stop the `mysqld` server with `mysqladmin shutdown`.

The trace file can become **very large!** To generate a smaller trace file, you can use debugging options something like this:

```
mysqld --debug=d,info,error,query,general,where:0,/tmp/mysqld.trace
```

This only prints information with the most interesting tags to the trace file.

If you make a bug report about this, please only send the lines from the trace file to the appropriate mailing list where something seems to go wrong! If you can't locate the wrong place, you can open a bug report and upload the trace file to the report, so that a MySQL developer can take a look at it. For instructions, see Section 1.7, “How to Report Bugs or Problems”.

The trace file is made with the **DEBUG** package by Fred Fish. See [Section 23.4.3, “The DEBUG Package”](#).

23.4.1.3. Using `pdb` to create a Windows crashdump

Program Database files (extension `pdb`) are included in the Noinstall distribution of MySQL. These files provide information for debugging your MySQL installation in the event of a problem.

The PDB file contains more detailed information about `mysqld` and other tools that enables more detailed trace and dump files to be created. You can use these with Dr Watson, `WinDbg` and Visual Studio to debug `mysqld`.

For more information on PDB files, see [Microsoft Knowledge Base Article 121366](#). For more information on the debugging options available, see [Debugging Tools for Windows](#).

Dr Watson is installed with all Windows distributions, but if you have installed Windows development tools, Dr Watson may have been replaced with `WinDbg`, the debugger included with Visual Studio, or the debugging tools provided with Borland or Delphi.

To generate a crash file using Dr Watson, follow these steps:

1. Start Dr Watson by running `drwtsn32.exe` interactively using the `-i` option:

```
C:\> drwtsn32 -i
```

2. Set the **Log File Path** to the directory where you want to store trace files.
3. Make sure **Dump All Thread Contexts** and **Append To Existing Log File**.
4. Uncheck **Dump Symbol Table**, **Visual Notification**, **Sound Notification** and **Create Crash Dump File**.
5. Set the **Number of Instructions** to a suitable value to capture enough calls in the stacktrace. A value of at 25 should be enough.

Note that the file generated can become very large.

23.4.1.4. Debugging `mysqld` under `gdb`

On most systems you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some older `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case, you can only have one thread active at a time. It is best to upgrade to `gdb` 5.1 because thread debugging works much better with this version!

NPTL threads (the new thread library on Linux) may cause problems while running `mysqld` under `gdb`. Some symptoms are:

- `mysqld` hangs during startup (before it writes `ready for connections`).
- `mysqld` crashes during a `pthread_mutex_lock()` or `pthread_mutex_unlock()` call.

In this case, you should set the following environment variable in the shell before starting `gdb`:

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

When running `mysqld` under `gdb`, you should disable the stack trace with `--skip-stack-trace [463]` to be able to catch segfaults within `gdb`.

In MySQL 4.0.14 and above you should use the `--gdb` [444] option to `mysqld`. This installs an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling.

It is very hard to debug MySQL under `gdb` if you do a lot of new connections the whole time as `gdb` doesn't free the memory for old threads. You can avoid this problem by starting `mysqld` with `thread_cache_size` [561] set to a value equal to `max_connections` [519] + 1. In most cases just using `--thread_cache_size=5` [561] helps a lot!

If you want to get a core dump on Linux if `mysqld` dies with a `SIGSEGV` signal, you can start `mysqld` with the `--core-file` [439] option. This core file can be used to make a backtrace that may help you find out why `mysqld` died:

```
shell> gdb mysqld core
gdb> backtrace full
gdb> quit
```

See [Section C.5.4.2, "What to Do If MySQL Keeps Crashing"](#).

If you are using `gdb` 4.17.x or above on Linux, you should install a `.gdb` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with `gdb`, you should download `gdb` 5.x and try this instead. The new `gdb` version has very improved thread handling!

Here is an example how to debug `mysqld`:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

Include the preceding output in a bug report, which you can file using the instructions in [Section 1.7, "How to Report Bugs or Problems"](#).

If `mysqld` hangs, you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable.

23.4.1.5. Using a Stack Trace

On some operating systems, the error log contains a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See [Section 5.2.2, "The Error Log"](#). To get a stack trace, you must not compile `mysqld` with the `-fomit-frame-pointer` option to `gcc`. See [Section 23.4.1.1, "Compiling MySQL for Debugging"](#).

A stack trace in the error log looks something like this:

```
mysqlld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqlld died. If you see no messages after
this, something went terribly wrong...

stack_bottom = 0x41fd0110 thread_stack 0x40000
mysqlld(my_print_stacktrace+0x32)[0x9da402]
mysqlld(handle_segfault+0x28a)[0x6648e9]
/lib/libpthread.so.0[0x7f1a5af000f0]
/lib/libc.so.6(strcmp+0x2)[0x7f1a5a10f0f2]
mysqlld(_Z21check_change_passwordP3THDPKcS2_Pcj+0x7c)[0x7412cb]
mysqlld(_ZN16set_var_password5checkEP3THD+0xd0)[0x688354]
mysqlld(_Z17sql_set_variablesP3THDP4ListI12set_var_baseE+0x68)[0x688494]
mysqlld(_Z21mysql_execute_commandP3THD+0x41a0)[0x67a170]
mysqlld(_Z11mysql_parseP3THDPKcjPS2_+0x282)[0x67f0ad]
mysqlld(_Z16dispatch_commandl9enum_server_commandP3THDPcj+0xbb7[0x67fdf8]
mysqlld(_Z10do_commandP3THD+0x24d)[0x6811b6]
mysqlld(handle_one_connection+0x11c)[0x66e05e]
```

If resolution of function names for the trace fails, the trace contains less information:

```
mysqlld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqlld died. If you see no messages after
this, something went terribly wrong...

stack_bottom = 0x41fd0110 thread_stack 0x40000
[0x9da402]
[0x6648e9]
[0x7f1a5af000f0]
[0x7f1a5a10f0f2]
[0x7412cb]
[0x688354]
[0x688494]
[0x67a170]
[0x67f0ad]
[0x67fdf8]
[0x6811b6]
[0x66e05e]
```

In the latter case, you can use the [resolve_stack_dump](#) utility to determine where `mysqlld` died by using the following procedure:

1. Copy the numbers from the stack trace to a file, for example `mysqlld.stack`. The numbers should not include the surrounding square brackets:

```
0x9da402
0x6648e9
0x7f1a5af000f0
0x7f1a5a10f0f2
0x7412cb
0x688354
0x688494
0x67a170
0x67f0ad
0x67fdf8
0x6811b6
0x66e05e
```

2. Make a symbol file for the `mysqlld` server:

```
shell> nm -n libexec/mysqlld > /tmp/mysqlld.sym
```

If `mysqlld` is not linked statically, use the following command instead:

```
shell> nm -D -n libexec/mysqld > /tmp/mysqld.sym
```

If you want to decode C++ symbols, use the `--demangle`, if available, to `nm`. If your version of `nm` does not have this option, you will need to use the `c++filt` command after the stack dump has been produced to demangle the C++ names.

- Execute the following command:

```
shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack
```

If you were not able to include demangled C++ names in your symbol file, process the `resolve_stack_dump` output using `c++filt`:

```
shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack | c++filt
```

This prints out where `mysqld` died. If that does not help you find out why `mysqld` died, you should create a bug report and include the output from the preceding command with the bug report.

However, in most cases it does not help us to have just a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, in most cases we need to know the statement that killed `mysqld` and preferably a test case so that we can repeat the problem! See [Section 1.7, “How to Report Bugs or Problems”](#).

Newer versions of `glibc` stack trace functions also print the address as relative to the object. On `glibc`-based systems (Linux), the trace for a crash within a plugin looks something like:

```
plugin/auth/auth_test_plugin.so(+0x9a6)[0x7ff4d11c29a6]
```

To translate the relative address (`+0x9a6`) into a file name and line number, use this command:

```
shell> addr2line -fie auth_test_plugin.so 0x9a6
auth_test_plugin
mysql-trunk/plugin/auth/test_plugin.c:65
```

The `addr2line` utility is part of the `binutils` package on Linux.

On Solaris, the procedure is similar. The Solaris `printstack()` already prints relative addresses:

```
plugin/auth/auth_test_plugin.so:0x1510
```

To translate, use this command:

```
shell> gaddr2line -fie auth_test_plugin.so 0x1510
mysql-trunk/plugin/auth/test_plugin.c:88
```

Windows already prints the address, function name and line:

```
000007FEF07E10A4 auth_test_plugin.dll!auth_test_plugin()[test_plugin.c:72]
```

23.4.1.6. Using Server Logs to Find Causes of Errors in `mysqld`

Note that before starting `mysqld` with the general query log enabled, you should check all your tables with `myisamchk`. See [Chapter 5, MySQL Server Administration](#).

If `mysqld` dies or hangs, you should start `mysqld` with the general query log enabled. See [Section 5.2.3, “The General Query Log”](#). When `mysqld` dies again, you can examine the end of the log file for the query that killed `mysqld`.

If you use the default general query log file, the log is stored in the database directory as `host_name.log`. In most cases it is the last query in the log file that killed `mysqld`, but if possible you should verify this by

restarting `mysqld` and executing the found query from the `mysql` command-line tools. If this works, you should also test all complicated queries that didn't complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that `mysqld` is using indexes properly. See [Section 13.8.2, “EXPLAIN Syntax”](#).

You can find the queries that take a long time to execute by starting `mysqld` with the slow query log enabled. See [Section 5.2.5, “The Slow Query Log”](#).

If you find the text `mysqld restarted` in the error log file (normally named `hostname.err`) you probably have found a query that causes `mysqld` to fail. If this happens, you should check all your tables with `myisamchk` (see [Chapter 5, MySQL Server Administration](#)), and test the queries in the MySQL log files to see whether one fails. If you find such a query, try first upgrading to the newest MySQL version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to a MySQL mailing list. The mailing lists are described at <http://lists.mysql.com/>, which also has links to online list archives.

If you have started `mysqld` with `--myisam-recover-options [454]`, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further. See [Section 5.1.3, “Server Command Options”](#).

As of MySQL 5.5.3, when the server detects `MyISAM` table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: `Got an error from thread_id=1, mi_dynrec.c:368`. This is useful information to include in bug reports.

It is not a good sign if `mysqld` did die unexpectedly, but in this case, you should not investigate the `Checking table...` messages, but instead try to find out why `mysqld` died.

23.4.1.7. Making a Test Case If You Experience Table Corruption

If you get corrupted tables or if `mysqld` always fails after some update commands, you can test whether this bug is reproducible by doing the following:

- Take down the MySQL daemon (with `mysqladmin shutdown`).
- Make a backup of the tables (to guard against the very unlikely case that the repair does something bad).
- Check all tables with `myisamchk -s database/*.MYI`. Repair any wrong tables with `myisamchk -r database/table.MYI`.
- Make a second backup of the tables.
- Remove (or move away) any old log files from the MySQL data directory if you need more space.
- Start `mysqld` with the binary log enabled. If you want to find a query that crashes `mysqld`, you should start the server with both the general query log enabled as well. See [Section 5.2.3, “The General Query Log”](#), and [Section 5.2.4, “The Binary Log”](#).
- When you have gotten a crashed table, stop the `mysqld server`.
- Restore the backup.

- Restart the `mysqld` server **without** the binary log enabled.
- Re-execute the commands with `mysqlbinlog binary-log-file | mysql`. The binary log is saved in the MySQL database directory with the name `hostname-bin.NNNNNNN`.
- If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found a reproducible bug that should be easy to fix! FTP the tables and the binary log to our bugs database using the instructions given in [Section 1.7, “How to Report Bugs or Problems”](#). If you are a support customer, you can use the MySQL Customer Support Center <https://support.mysql.com/> to alert the MySQL team about the problem and have it fixed as soon as possible.

You can also use the script `mysql_find_rows` to just execute some of the update statements if you want to narrow down the problem.

23.4.2. Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `-DWITH_DEBUG=1` [155]. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:0,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:0,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See [Section 1.7, “How to Report Bugs or Problems”](#).

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

23.4.3. The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is debugging. See [Section 23.4.1.2, “Creating Trace Files”](#).

This section summarizes the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support. For more information about programming with the DBUG package, see the DBUG manual in the `dbug` directory of MySQL source distributions. It is best to use a recent distribution to get the most updated DBUG manual.

You use the debug package by invoking a program with the `--debug="..."` or the `-#...` option.

Most MySQL programs have a default debug string that is used if you don't specify an option to `--debug`. The default trace file is usually `/tmp/program_name.trace` on Unix and `\program_name.trace` on Windows.

The debug control string is a sequence of colon-separated fields as follows:

```
<field_1>:<field_2>:...:<field_N>
```

Each field consists of a mandatory flag character followed by an optional “,” and comma-separated list of modifiers:

```
flag[,modifier,modifier,...,modifier]
```

The following table shows the currently recognized flag characters.

| Flag | Description |
|------|---|
| d | Enable output from DBUG_<N> macros for the current state. May be followed by a list of keywords which selects output only for the DBUG macros with that keyword. An empty list of keywords implies output for all macros. |
| D | Delay after each debugger output line. The argument is the number of tenths of seconds to delay, subject to machine capabilities. For example, <code>-#D,20</code> specifies a delay of two seconds. |
| f | Limit debugging, tracing, and profiling to the list of named functions. Note that a null list disables all functions. The appropriate <code>d</code> or <code>t</code> flags must still be given; this flag only limits their actions if they are enabled. |
| F | Identify the source file name for each line of debug or trace output. |
| i | Identify the process with the PID or thread ID for each line of debug or trace output. |
| g | Enable profiling. Create a file called <code>dbugmon.out</code> containing information that can be used to profile the program. May be followed by a list of keywords that select profiling only for the functions in that list. A null list implies that all functions are considered. |
| L | Identify the source file line number for each line of debug or trace output. |
| n | Print the current function nesting depth for each line of debug or trace output. |
| N | Number each line of debug output. |
| o | Redirect the debugger output stream to the specified file. The default output is <code>stderr</code> . |
| O | Like <code>o</code> , but the file is really flushed between each write. When needed, the file is closed and reopened between each write. |
| p | Limit debugger actions to specified processes. A process must be identified with the <code>DBUG_PROCESS</code> macro and match one in the list for debugger actions to occur. |
| P | Print the current process name for each line of debug or trace output. |
| r | When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin. |
| S | Do function <code>_sanity(_file_,_line_)</code> at each debugged function until <code>_sanity()</code> returns something that differs from 0. |
| t | Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option. |

Some examples of debug control strings that might appear on a shell command line (the `-#` is typically used to introduce a control string to an application program) are:

```
-#d:t
-#d:f,main,subr1:F:L:t,20
-#d,input,output,files:n
-#d:t:i:0,\\mysql.d.trace
```

In MySQL, common tags to print (with the `d` option) are `enter`, `exit`, `error`, `warning`, `info`, and `loop`.

Chapter 24. MySQL Enterprise Edition

Table of Contents

| | |
|---------------------------------------|------|
| 24.1. MySQL Enterprise Monitor | 3863 |
| 24.2. MySQL Enterprise Backup | 3864 |
| 24.3. MySQL Enterprise Security | 3865 |
| 24.4. MySQL Enterprise Audit | 3865 |
| 24.5. MySQL Thread Pool | 3865 |

MySQL Enterprise Edition is a commercial product. Like MySQL Community Edition, MySQL Enterprise Edition includes MySQL Server, a fully integrated transaction-safe, ACID-compliant database with full commit, rollback, crash-recovery, and row-level locking capabilities. In addition, MySQL Enterprise Edition includes the following components designed to provide monitoring and online backup, as well as improved security and scalability:

The following sections briefly discuss each of these components and indicate where to find more detailed information. To learn more about commercial products, see <http://www.mysql.com/products/>.

- [MySQL Enterprise Monitor](#)
- [MySQL Enterprise Backup](#)
- [MySQL Enterprise Security](#)
- [MySQL Enterprise Audit](#)
- [MySQL Thread Pool](#)

24.1. MySQL Enterprise Monitor

MySQL Enterprise Monitor is an enterprise monitoring system for MySQL that keeps an eye on your MySQL servers, notifies you of potential issues and problems, and advises you how to fix the issues. MySQL Enterprise Monitor can monitor all kinds of configurations, from a single MySQL server that is important to your business, all the way up to a huge farm of MySQL servers powering a busy web site.

The following discussion briefly summarizes the basic components that make up the MySQL Enterprise Monitor product. For more information, see the MySQL Enterprise Monitor manual, available at <http://dev.mysql.com/doc/mysql-monitor/en/>.

MySQL Enterprise Monitor components can be installed in various configurations depending on your database and network topology, to give you the best combination of reliable and responsive monitoring data, with minimal overhead on the database server machines. A typical MySQL Enterprise Monitor installation consists of:

- One or more MySQL servers to monitor. MySQL Enterprise Monitor can monitor both Community and Enterprise MySQL server releases.
- A MySQL Enterprise Agent for each monitored MySQL server.
- A single MySQL Enterprise Service Manager, which collates information from the agents and provides the user interface to the collected data.

MySQL Enterprise Monitor is designed to monitor one or more MySQL servers. The monitoring information is collected by using an agent, *MySQL Enterprise Agent*. The agent communicates with the MySQL server

that it monitors, collecting variables, status and health information, and sending this information to the MySQL Enterprise Service Manager. If you have multiple MySQL servers, then you have multiple MySQL Enterprise Agent processes monitoring each MySQL server.

The information collected by the agent about each MySQL server you are monitoring is sent to the *MySQL Enterprise Service Manager*. This server collates all of the information from the agents. As it collates the information sent by the agents, the MySQL Enterprise Service Manager continually tests the collected data, comparing the status of the server to reasonable values. When thresholds are reached, the server can trigger an event (including an alarm and notification) to highlight a potential issue, such as low memory, high CPU usage, or more complex conditions such as insufficient buffer sizes and status information. We call each test, with its associated threshold value, a *rule*.

These rules, and the alarms and notifications, are each known as a *MySQL Enterprise Advisor*. Advisors form a critical part of the MySQL Enterprise Service Manager, as they provide warning information and troubleshooting advice about potential problems.

The MySQL Enterprise Service Manager includes a web server, and you interact with it through any web browser. This interface, the MySQL Enterprise Dashboard, displays all of the information collected by the agents, and lets you view all of your servers and their current status as a group or individually. You control and configure all aspects of the service using the MySQL Enterprise Dashboard.

The information supplied by the MySQL Enterprise Agent processes also includes statistical and query information, which you can view in the form of graphs. For example, you can view aspects such as server load, query numbers, or index usage information as a graph over time. The graph lets you pinpoint problems or potential issues on your server, and can help diagnose the impact from database or external problems (such as external system or network failure) by examining the data from a specific time interval.

The MySQL Enterprise Agent can also be configured to collect detailed information about the queries executed on your server, including the row counts and performance times for executing each query. You can correlate the detailed query data with the graphical information to identify which queries were executing when you experienced a particularly high load, index or other issue. The query data is supported by a system called Query Analyzer, and the data can be presented in different ways depending on your needs.

24.2. MySQL Enterprise Backup

MySQL Enterprise Backup performs hot backup operations for MySQL databases. The product is architected for efficient and reliable backups of tables created by the InnoDB storage engine. For completeness, it can also back up tables from MyISAM and other storage engines.

The following discussion briefly summarizes MySQL Enterprise Backup. For more information, see the MySQL Enterprise Backup manual, available at <http://dev.mysql.com/doc/mysql-enterprise-backup/en/>.

Hot backups are performed while the database is running and applications are reading and writing to it. This type of backup does not block normal database operations, and it captures even changes that occur while the backup is happening. For these reasons, hot backups are desirable when your database “grows up” -- when the data is large enough that the backup takes significant time, and when your data is important enough to your business that you must capture every last change, without taking your application, web site, or web service offline.

MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables using MyISAM or other non-InnoDB storage engines, it does a “warm” backup, where the database continues to run, but those tables cannot be modified while being backed up. For efficient backup operations, you can designate InnoDB as the default storage engine for new tables, or convert existing tables to use the InnoDB storage engine.

24.3. MySQL Enterprise Security

MySQL Enterprise Edition provides plugins that implement authentication using external services:

- MySQL Enterprise Edition includes an authentication plugin that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as Unix passwords or an LDAP directory.
- MySQL Enterprise Edition includes an authentication plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password.

For more information, see [Section 6.3.6.2, “The PAM Authentication Plugin”](#), and [Section 6.3.6.3, “The Windows Native Authentication Plugin”](#).

24.4. MySQL Enterprise Audit

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

For more information, see [Section 6.3.10, “MySQL Enterprise Audit”](#).

24.5. MySQL Thread Pool

MySQL Enterprise Edition includes the MySQL Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. In MySQL Enterprise Edition, a thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

For more information, see [Section 8.11.6, “The Thread Pool Plugin”](#).

Chapter 25. MySQL Workbench

MySQL Workbench provides a graphical tool for working with MySQL Servers and databases. MySQL Workbench fully supports MySQL Server versions 5.1 and above. It is also compatible with MySQL Server 5.0, but not every feature of 5.0 may be supported. It does not support MySQL Server versions 4.x.

The following discussion briefly describes MySQL Workbench capabilities. For more information, see the MySQL Workbench manual, available at <http://dev.mysql.com/doc/workbench/en/>.

MySQL Workbench provides three main areas of functionality:

- **SQL Development:** Enables you to create and manage connections to database servers. As well as enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor. This functionality replaces that previously provided by the Query Browser standalone application.
- **Data Modeling:** Enables you to create models of your database schema graphically, reverse and forward engineer between a schema and a live database, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views.
- **Server Administration:** Enables you to create and administer server instances. This functionality replaces that previously provided by the MySQL Administrator standalone application.

MySQL Workbench is available in two editions, the Community Edition and the Standard Edition. The Community Edition is available free of charge. The Standard Edition provides additional Enterprise features, such as database documentation generation, at low cost.

Chapter 26. MySQL for Excel

Table of Contents

| | |
|--|------|
| 26.1. Installing and Configuring | 3869 |
| 26.2. Edit MySQL Data | 3870 |
| 26.3. Import MySQL Data | 3871 |
| 26.4. Append MySQL Data | 3872 |
| 26.5. Export MySQL Data | 3876 |

MySQL for Excel enables you to work with a MySQL database from within Microsoft Excel. MySQL data can be imported into Excel, Excel data can be exported into MySQL as a new table or appended to a current table, and MySQL for Excel enables you to edit the MySQL data directly from within Excel.

Visit the [MySQL for Excel forum](#) for additional MySQL for Excel help and support.

For release notes detailing the changes in each release of MySQL for Excel, see [MySQL for Excel Release Notes](#).

26.1. Installing and Configuring

MySQL for Excel is a product for Microsoft Windows, and it is installed with MySQL Installer. And typically you will not be required to install or configure additional tools to use MySQL for Excel.



Note

To install, download and execute the [MySQL Installer](#). Select the MySQL for Excel product and then proceed with the installation. See the [MySQL Installer manual](#) for additional details.

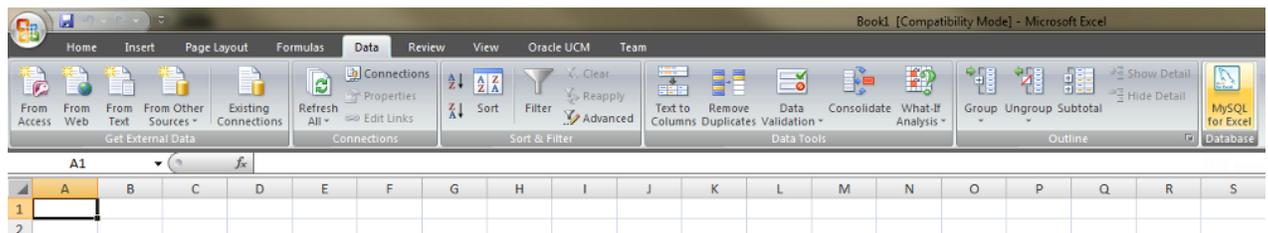
MySQL for Excel Requirements

The MySQL Installer installation process will check if these requirements are met, or notify you if further action is required before proceeding with the installation.

- .NET Framework 4.0 (Client or Full Profile).
- Microsoft Office Excel 2007 or greater, for Microsoft Windows.
- Visual Studio Tools for Office 4.0, and MySQL Installer may install this for you.
- An available MySQL Server connection.

MySQL for Excel is loaded and executed by selecting the **Data** menu tab in Excel, and then choosing the "MySQL for Excel" Database icon. This opens a new Excel sidebar with the available MySQL for Excel options. The navigation bar with the MySQL for Excel icon is shown in the following screenshot:

Figure 26.1. The MySQL for Excel navigation bar



26.2. Edit MySQL Data

MySQL for Excel enables you to load and edit MySQL data directly from Microsoft Excel. Changes are immediately committed if the **Auto-Commit** option is enabled, or done manually by pressing **Commit Changes**.

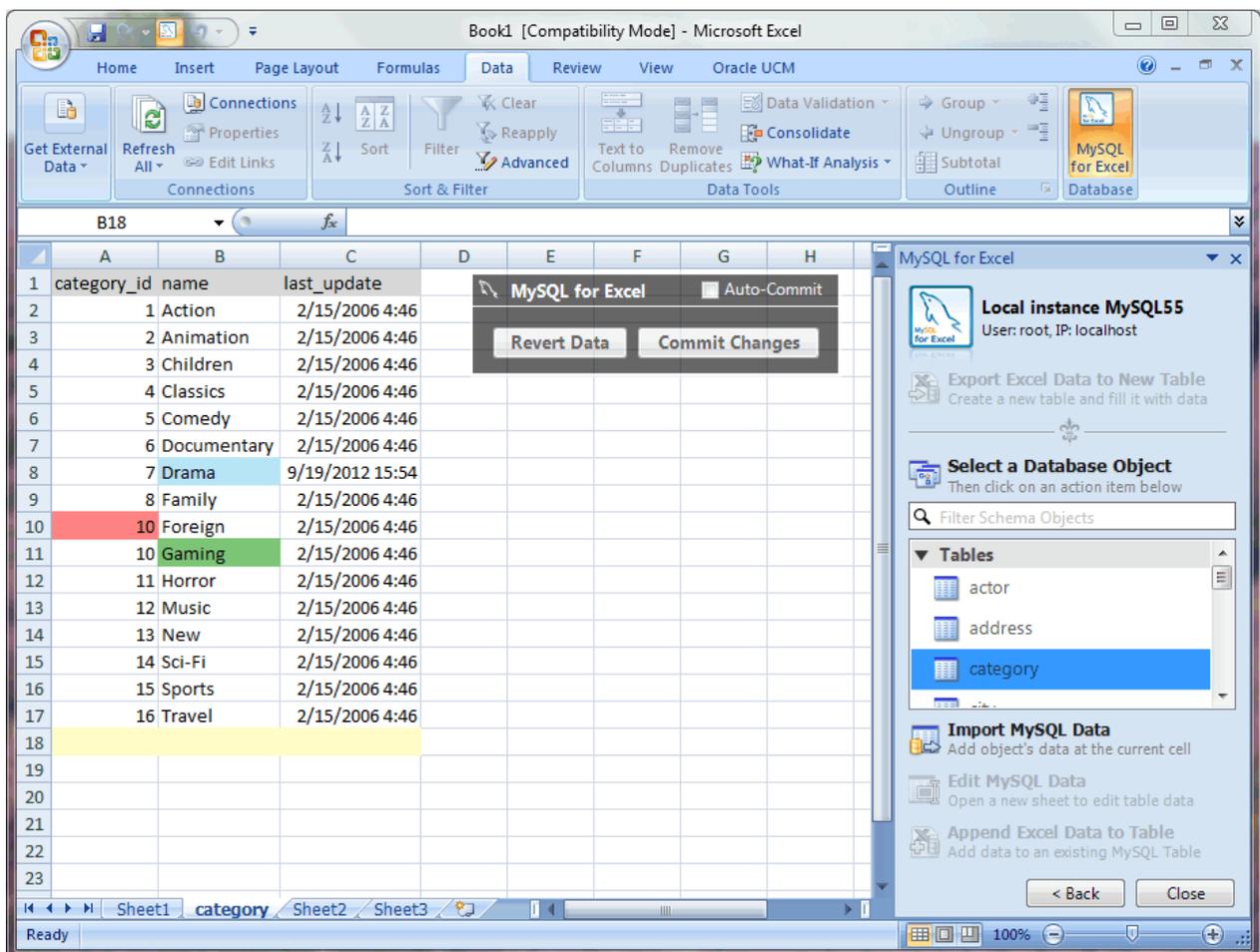


Note

The Edit MySQL Data feature exists as of MySQL for Excel 1.1.0.

The example below uses the `category` table of the example `sakila` database, but the screen will look the same for any table. Within MySQL for Excel, **Open a MySQL Connection**, click the `sakila` schema, Next, select the `category` table, click **Edit MySQL Data**, then choose **Import** to import the data into a new Microsoft Excel worksheet for editing.

Figure 26.2. Editing table data with MySQL for Excel



The background color represents the status of each cell, and there are four distinct colors that are used while editing table data:

Table 26.1. Background cell colors

| Color | Description |
|--------|---|
| White | Default color for all cells. This is either the original data, or the data after Refresh from DB is clicked. |
| Blue | Cells that were committed with success. |
| Green | Cells that were modified but have not yet been committed. |
| Red | Cells that generated an error when a commit was attempted. An error dialog is also displayed while the commit is attempted. |
| Yellow | Cells that accept new data. Data entered here is inserted into the MySQL table. |

In our example, the blue "Drama" field was changed and then committed first, then the green "Gaming" field was changed but not committed, and then **Auto-Commit** was enabled before changing the "9" to a "10" in column 10, which generated an error because this commit would have added a duplicate value as primary key.

26.3. Import MySQL Data

Data can be imported from MySQL into a Microsoft Excel spreadsheet by using the **Import MySQL Data** option after selecting either a table, view, or procedure to import.

Choosing columns to import

By default, all columns are selected and will be imported. Specific columns may be selected (or unselected) using the standard Microsoft Windows method of either **Control + Mouse click** to toggle the selection of individual columns, or **Shift + Mouse click** to select a range of columns.

The background color of a column shows the status of each column. The color white means that the column has been selected, and therefore it will be imported. Conversely, a gray background means that the column will not be imported.

Right-clicking anywhere in the preview grid opens a context-menu with either a **Select None** or **Select All** option, depending on the current status.

Importing a table

The dialog while importing a table includes the following options:

- **Include Column Names as Headers:** Enabled by default, this inserts the column names at the top of the Microsoft Excel spreadsheet as a "headers" row.
- **Limit to ___ Rows and Start with Row ___:** Disabled by default, this limits the range of imported data. The **Limit to** option defaults to **1**, and defines the number of rows to import. The **Start with Row** option defaults to **1** (the first row), and defines where the import begins. Each option has a maximum value of COUNT(rows) in the table.

Importing a table displays a dialog similar to the following:

Figure 26.3. Importing table data with MySQL for Excel

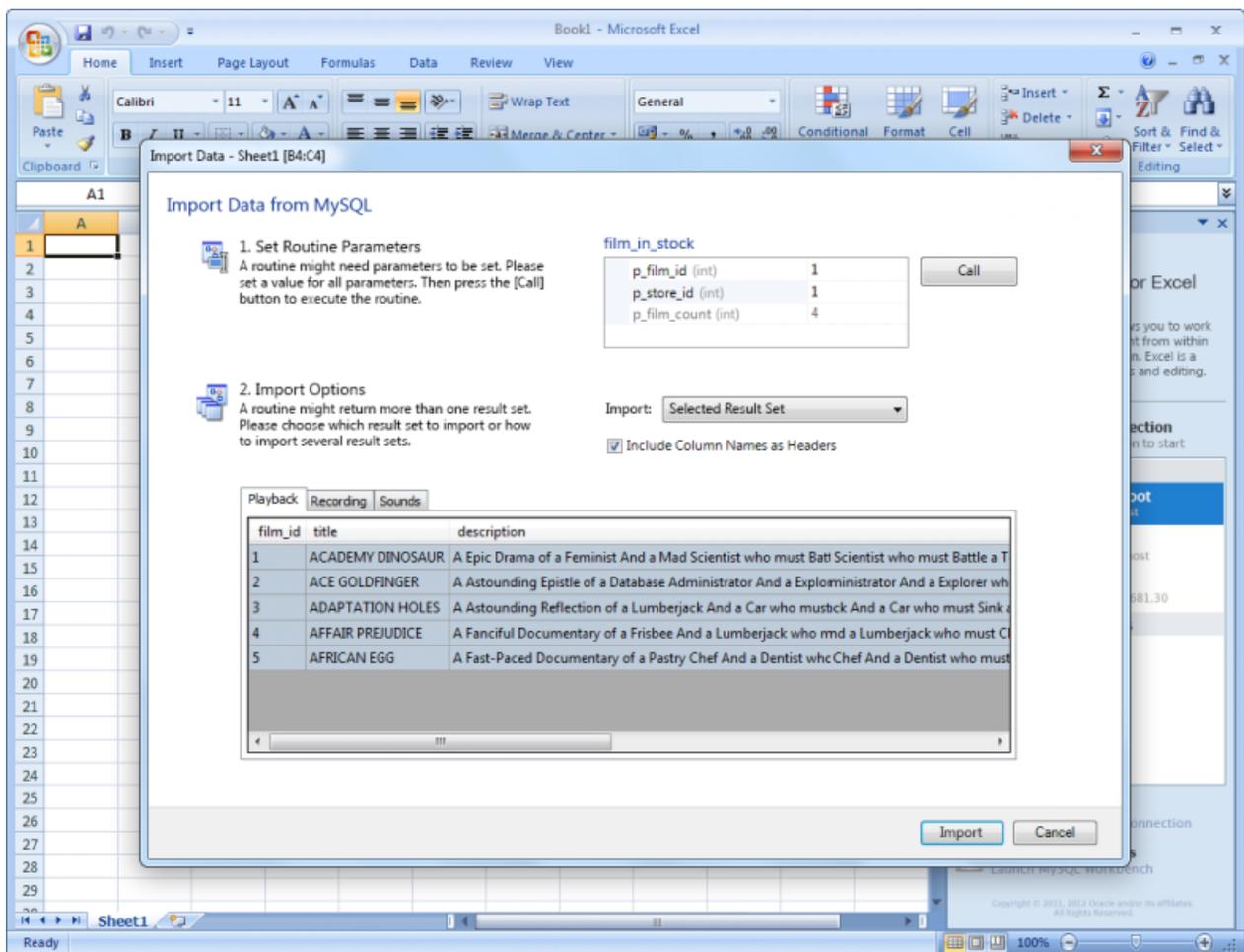
Importing a view or procedure

Importing a view or procedure displays a similar dialogue, but with the following options:

- **Include Column Names as Headers:** Enabled by default, this will insert the column names at the top of the Excel spreadsheet as a "headers" row.
- **Import:** Because a procedure might return multiple result sets, the import options include:
 - **Selected Result Set:** Imports the selected tab sheet. This is the default behavior.
 - **All Result Sets - Arranged Horizontally:** Imports all result sets into the Excel Worksheet horizontally, and inserts one empty column between each result set.
 - **All Result Sets - Arranged Vertically:** Imports all result sets into the Excel Worksheet vertically, and inserts one empty row between each result set.

For example, a dialogue like the following is displayed after importing a procedure and pressing the **Call** button to invoke the stored procedure:

Figure 26.4. Importing called stored procedure data with MySQL for Excel



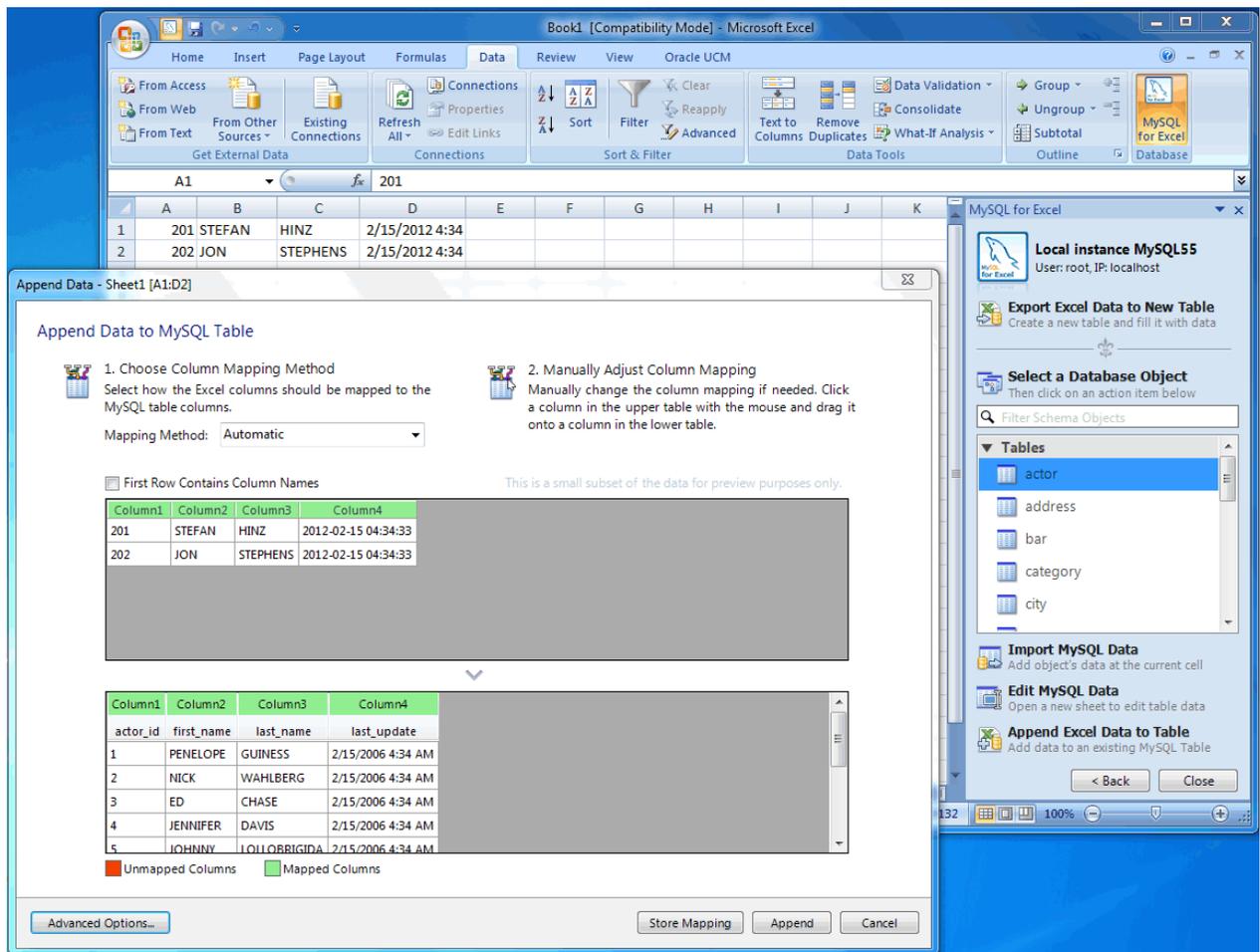
26.4. Append MySQL Data

Data from a Microsoft Excel spreadsheet can be appended to a MySQL database table by using the **Append Excel MySQL Data to Table** option.

Column mappings

Mapping the Excel columns to the MySQL columns can be executed automatically (default), manually, or by using a stored mapping routine. An automatic mapping routine is the default, and can be tweaked if every column cannot be matched automatically. The following screenshot shows two columns of Excel data, and the preview dialog after choosing **Append Excel Data to Table**:

Figure 26.5. Appending Excel data to MySQL (Automatic mapping)



General mapping information

It is common to tweak the column mappings. A few notes about the manual mapping process:

- Manual mapping is performed by dragging a column from the upper source grid (Excel spreadsheet) and dropping it into the lower target column MySQL table grid. Click anywhere within the column to initiate this dragging routine.
- The color of the header field for each column defines the current mapping status of the column. The colors include:
 - Green: A source column is mapped to a target column.
 - Red: A target column is not mapped.

- **Gray:** A source column is not mapped.
- A source column may be mapped to multiple target columns, although this action generates a warning dialog.
- Right-clicking a target column shows a context menu with options to either **Remove Column Mapping** for a single column, or to **Clear All Mappings** for all columns. Dragging a target column outside of the grid removes the mapping.

Mapping methods

The three mapping methods are described below:

- **Automatic:** The automatic mapping method attempts to match the Excel source column names with the MySQL target table column names. It is then possible to manually tweak the mapping afterwards.

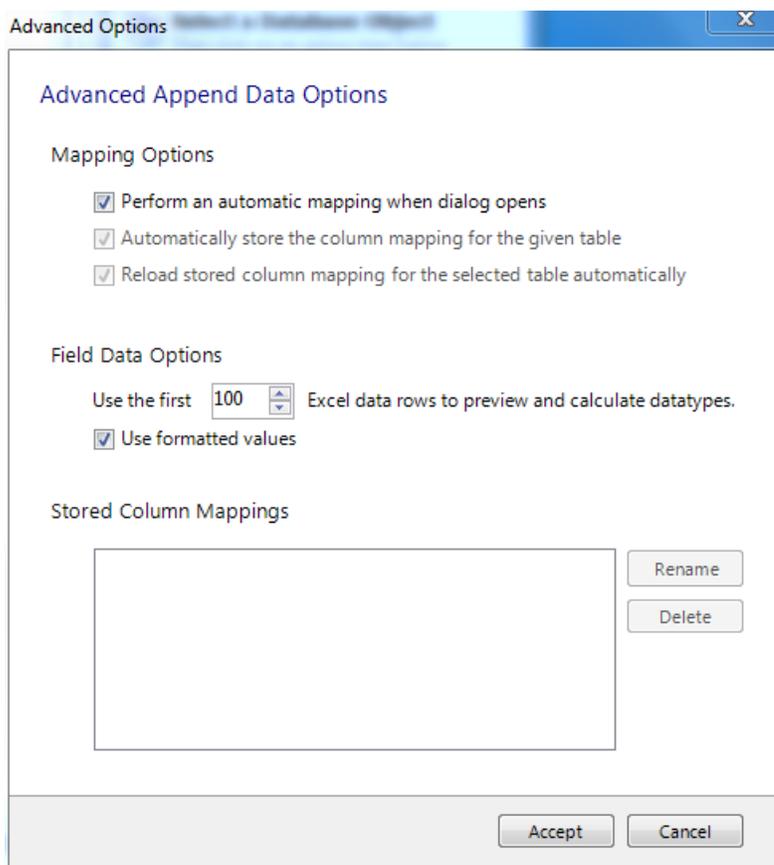
If the automatic process finds zero columns to match, then a simple 1 to 1 matching routine is attempted. Meaning, SourceColumn #1 to TargetColumn #1, SourceColumn #2 to TargetColumn #2, and so on.

- **Manual:** The source column names are manually dragged (matched) with the target column names. Manual dragging can also be performed after the **Automatic** method is selected.
- **Stored:** Manual mapping styles may be saved using the **Store Mapping** button, which will also prompt for a name and then save it using a "*name* (dbname.tablename)" naming scheme. The saved mapping style will then be available alongside the **Automatic** and **Manual** options.

Stored mappings may be deleted or renamed within the **Advanced Options** dialog.

Advanced Options

There are several advanced options that are configured and stored between sessions for each Excel user. The dialog looks similar to:

Figure 26.6. Appending Excel data to MySQL (Advanced Options)

The advanced **Mapping Options**:

- **Perform an automatic mapping when dialog opens**: Automatically attempt to map the target and source when the **Append Data** dialog is opened. This feature is enabled by default.
- **Automatically store the column mapping for the given table**: Stores each mapping routine after executing the **Append** operation. The mapping routine is saved using the "tablenameMapping (dbname.tablename)" format. This may also be performed manually using the **Store Mapping** button. It is enabled by default, and this feature was added in MySQL for Excel 1.1.0.
- **Reload stored column mapping for the selected table automatically**: If a stored mapping routine exists that matches all column names in the source grid with the target grid, then it is automatically be loaded. This is enabled by default, and this feature was added in MySQL for Excel 1.1.0.

The advanced **Field Data Options**:

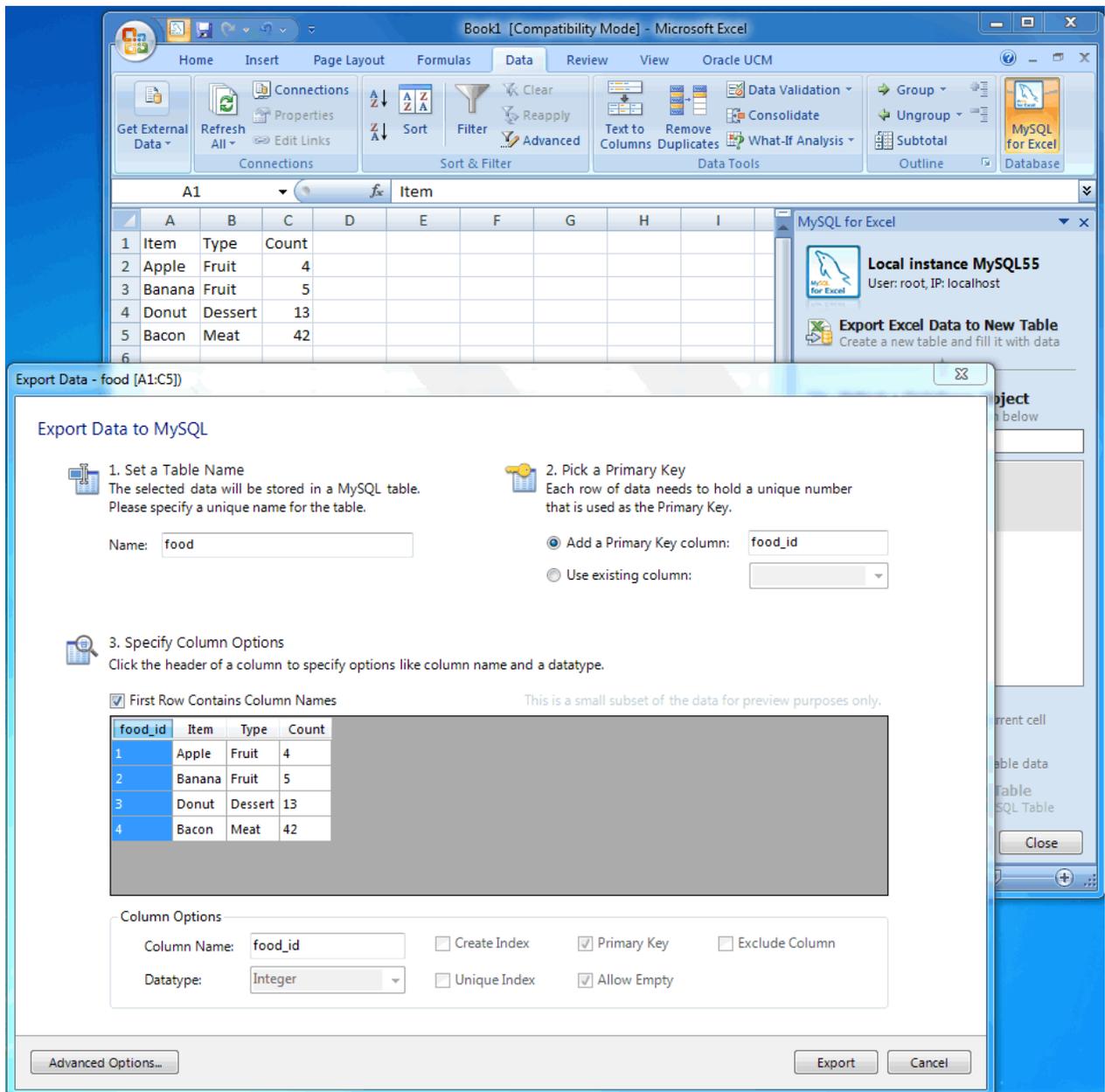
- **Use the first 100** (default) Excel data rows to preview and calculate data types. This determines the number of rows that the preview displays, and the values that affect the automatic mapping feature.
- **Use formatted values**: The data from Excel is treated as **Text**, **Double**, or **Date**. This is enabled by default. When disabled, data is never treated as a **Date** type, so for example, this means that a date would be represented as a number.

The **Stored Column Mappings** is a list of saved column mappings that were saved with with the "Automatically store the column mapping for the given table" feature, or manually with the **Store Mapping** option.

26.5. Export MySQL Data

Data from a Microsoft Excel spreadsheet can be exported to a new MySQL database table by using the **Export Excel Data to New Table** option. Exporting data looks like so:

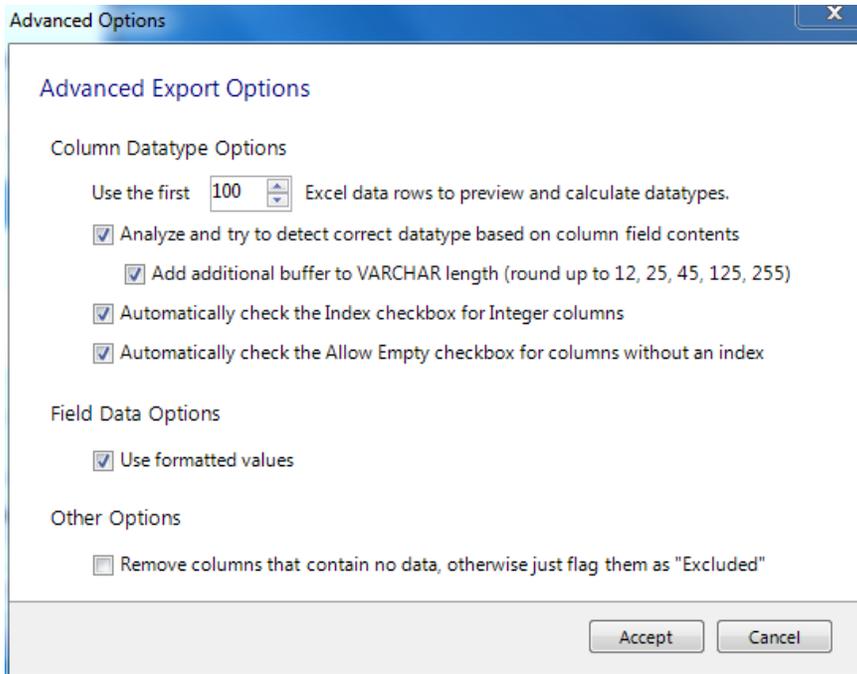
Figure 26.7. Exporting Excel data to MySQL



Advanced Export options

Several advanced options enables you to tweak the exported data. The advanced options dialog looks like so:

Figure 26.8. Exporting Excel data to MySQL (Advanced options)



- **Column Datatype Options:**

- Use the first `100` (default) Excel data rows to preview and calculate data types: This determines the number of rows that the preview displays, and the values that affect the automatic mapping feature.
- Analyze and try to detect correct datatype based on column field contents: Attempts to analyze the data and determine the data type for the column. The column type is defined as `VARCHAR` if it contains multiple types.
- Add additional buffer to `VARCHAR` length (round up to 12, 25, 45, 125, 255): When the data type is automatically detected and is set to `VARCHAR`, then it calculates the maximum length for all rows within the column, and rounds up the maximum length to one of the defined lengths above.

If disabled, then the `VARCHAR` length is set to the length of the longest entry in the Excel spreadsheet.

- Automatically check the Index checkbox for Integer columns: If enabled (default), columns with an Integer data type will have the **Create Index** option enabled by default.
- Automatically check the Allow Empty checkbox for columns without an index: If enabled (default), columns without the **Create Index** checkbox checked will automatically enable the **Allow Empty** configuration option.
- **Field Data options:**
 - Use formatted values: When enabled (default), the data from Excel is treated as `Text`, `Double`, or `Date`. When disabled, data is never treated as a `Date` type, so for example this means that a date would be represented as a number.

- **Other options:**

- Remove columns that contain no data, otherwise flag them as "Excluded": If enabled, columns without data in Excel are removed and not shown in the preview panel. If disabled (default), these columns will exist but have the **Exclude Column** option checked. This option was added in MySQL for Excel 1.1.0.

Appendix A. Licenses for Third-Party Components

Table of Contents

| | |
|--|------|
| A.1. Ant-Contrib License | 3882 |
| A.2. ANTLR 3 License | 3883 |
| A.3. ANTLR 3.3 License | 3884 |
| A.4. Boost Library License | 3884 |
| A.5. c3p0 JDBC Library License | 3885 |
| A.6. dtoa.c License | 3885 |
| A.7. Editline Library (libedit) License | 3885 |
| A.8. FindGTest.cmake License | 3887 |
| A.9. Fred Fish's Dbug Library License | 3888 |
| A.10. getarg License | 3889 |
| A.11. GLib License (for MySQL Proxy) | 3890 |
| A.12. GNU General Public License Version 2.0, June 1991 | 3890 |
| A.13. GNU Lesser General Public License Version 2.1, February 1999 | 3896 |
| A.14. GNU Libtool License | 3904 |
| A.15. GNU Readline License | 3905 |
| A.16. Google Controlling Master Thread I/O Rate Patch License | 3905 |
| A.17. Google Perftools (TCMalloc utility) License | 3905 |
| A.18. Google SMP Patch License | 3906 |
| A.19. jboss-common-jdbc-wrapper.jar License | 3907 |
| A.20. lib_sql.cc License | 3907 |
| A.21. Libaio License | 3907 |
| A.22. libevent License | 3907 |
| A.23. Libiconv License | 3909 |
| A.24. libintl License | 3910 |
| A.25. Linux-PAM License | 3910 |
| A.26. LPeg Library License | 3911 |
| A.27. Lua (liblua) License | 3911 |
| A.28. LuaFileSystem Library License | 3912 |
| A.29. md5 (Message-Digest Algorithm 5) License | 3912 |
| A.30. memcached License | 3913 |
| A.31. nt_servc (Windows NT Service class library) License | 3913 |
| A.32. OpenPAM License | 3913 |
| A.33. Paramiko License | 3914 |
| A.34. PCRE License | 3914 |
| A.35. Percona Multiple I/O Threads Patch License | 3915 |
| A.36. Python License | 3916 |
| A.37. Red HAT RPM Spec File License | 3926 |
| A.38. RegEX-Spencer Library License | 3927 |
| A.39. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) License | 3927 |
| A.40. Richard A. O'Keefe String Library License | 3928 |
| A.41. SHA-1 in C License | 3928 |
| A.42. Simple Logging Facade for Java (SLF4J) License | 3928 |
| A.43. zlib License | 3929 |
| A.44. ZLIB.NET License | 3929 |

The following is a list of the libraries we have included with the MySQL Server source and components used to test MySQL. We are thankful to all individuals that have created these. Some of the components

require that their licensing terms be included in the documentation of products that include them. Cross references to these licensing terms are given with the applicable items in the list.

- GroupLens Research Project

The MySQL Quality Assurance team would like to acknowledge the use of the MovieLens Data Sets (10 million ratings and 100,000 tags for 10681 movies by 71567 users) to help test MySQL products and to thank the GroupLens Research Project at the University of Minnesota for making the data sets available.

MySQL 5.5

- [Section A.6, “`dtoa.c` License”](#)
- [Section A.7, “Editline Library \(`libedit`\) License”](#)
- [Section A.8, “`FindGTest.cmake` License”](#)
- [Section A.9, “Fred Fish’s Dbug Library License”](#)
- [Section A.10, “`getarg` License”](#)
- [Section A.12, “GNU General Public License Version 2.0, June 1991”](#)
- [Section A.13, “GNU Lesser General Public License Version 2.1, February 1999”](#)
- [Section A.14, “GNU Libtool License”](#)
- [Section A.15, “GNU Readline License”](#)
- [Section A.16, “Google Controlling Master Thread I/O Rate Patch License”](#)
- [Section A.17, “Google Perftools \(TCMalloc utility\) License”](#)
- [Section A.18, “Google SMP Patch License”](#)
- [Section A.20, “`lib_sql.cc` License”](#)
- [Section A.21, “Libaio License”](#)
- [Section A.22, “`libevent` License”](#)
- [Section A.25, “Linux-PAM License”](#)
- [Section A.29, “md5 \(Message-Digest Algorithm 5\) License”](#)
- [Section A.31, “`nt_servc` \(Windows NT Service class library\) License”](#)
- [Section A.32, “OpenPAM License”](#)
- [Section A.35, “Percona Multiple I/O Threads Patch License”](#)
- [Section A.37, “Red HAT RPM Spec File License”](#)
- [Section A.38, “RegEX-Spencer Library License”](#)
- [Section A.39, “RFC 3174 - US Secure Hash Algorithm 1 \(SHA1\) License”](#)
- [Section A.40, “Richard A. O’Keefe String Library License”](#)
- [Section A.41, “SHA-1 in C License”](#)
- [Section A.43, “`zlib` License”](#)

MySQL Cluster 7.2

- [Section A.2, “ANTLR 3 License”](#)
- [Section A.6, “`dtoa.c` License”](#)
- [Section A.7, “Editline Library \(`libedit`\) License”](#)
- [Section A.8, “`FindGTest.cmake` License”](#)
- [Section A.9, “Fred Fish's Dbug Library License”](#)
- [Section A.10, “`getarg` License”](#)
- [Section A.12, “GNU General Public License Version 2.0, June 1991”](#)
- [Section A.13, “GNU Lesser General Public License Version 2.1, February 1999”](#)
- [Section A.14, “GNU Libtool License”](#)
- [Section A.15, “GNU Readline License”](#)
- [Section A.16, “Google Controlling Master Thread I/O Rate Patch License”](#)
- [Section A.17, “Google Perftools \(TCMalloc utility\) License”](#)
- [Section A.18, “Google SMP Patch License”](#)
- [Section A.20, “`lib_sql.cc` License”](#)
- [Section A.22, “`libevent` License”](#)
- [Section A.25, “Linux-PAM License”](#)
- [Section A.29, “md5 \(Message-Digest Algorithm 5\) License”](#)
- [Section A.30, “`memcached` License”](#)
- [Section A.31, “`nt_servc` \(Windows NT Service class library\) License”](#)
- [Section A.32, “OpenPAM License”](#)
- [Section A.33, “Paramiko License”](#)
- [Section A.35, “Percona Multiple I/O Threads Patch License”](#)
- [Section A.36, “Python License”](#)
- [Section A.38, “RegEX-Spencer Library License”](#)
- [Section A.39, “RFC 3174 - US Secure Hash Algorithm 1 \(SHA1\) License”](#)
- [Section A.40, “Richard A. O'Keefe String Library License”](#)
- [Section A.41, “SHA-1 in C License”](#)
- [Section A.43, “`zlib` License”](#)

MySQL Connector/C

- [Section A.9, “Fred Fish's Dbug Library License”](#)

- [Section A.38, “RegEX-Spencer Library License”](#)
- [Section A.39, “RFC 3174 - US Secure Hash Algorithm 1 \(SHA1\) License”](#)
- [Section A.43, “zlib License”](#)

MySQL Connector/CPP

- [Section A.4, “Boost Library License”](#)

MySQL Connector/J

- [Section A.1, “Ant-Contrib License”](#)
- [Section A.5, “c3p0 JDBC Library License”](#)
- [Section A.13, “GNU Lesser General Public License Version 2.1, February 1999”](#)
- [Section A.19, “jboss-common-jdbc-wrapper.jar License”](#)
- [Section A.42, “Simple Logging Facade for Java \(SLF4J\) License”](#)

MySQL Connector/Net

- [Section A.3, “ANTLR 3.3 License”](#)
- [Section A.39, “RFC 3174 - US Secure Hash Algorithm 1 \(SHA1\) License”](#)
- [Section A.43, “zlib License”](#)
- [Section A.44, “ZLIB.NET License”](#)

MySQL Connector/ODBC

- [Section A.14, “GNU Libtool License”](#)

MySQL Proxy

- [Section A.11, “GLib License \(for MySQL Proxy\)”](#)
- [Section A.13, “GNU Lesser General Public License Version 2.1, February 1999”](#)
- [Section A.22, “libevent License”](#)
- [Section A.23, “Libiconv License”](#)
- [Section A.24, “libintl License”](#)
- [Section A.26, “LPeg Library License”](#)
- [Section A.27, “Lua \(liblua\) License”](#)
- [Section A.28, “LuaFileSystem Library License”](#)
- [Section A.34, “PCRE License”](#)

A.1. Ant-Contrib License

The following software may be included in this product: Ant-Contrib

```
Ant-Contrib
Copyright (c) 2001-2003 Ant-Contrib project. All rights reserved.
Licensed under the Apache 1.1 License Agreement, a copy of which is reproduced below.
```

```
The Apache Software License, Version 1.1
```

```
Copyright (c) 2001-2003 Ant-Contrib project. All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgement:
"This product includes software developed by the
Ant-Contrib project (<http://sourceforge.net/projects/ant-contrib>)."
Alternately, this acknowledgement may appear in the software itself, if and wherever such third-party acknowledgements normally appear.
4. The name Ant-Contrib must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact ant-contrib-developers@lists.sourceforge.net.
5. Products derived from this software may not be called "Ant-Contrib" nor may "Ant-Contrib" appear in their names without prior written permission of the Ant-Contrib project.

```
THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE ANT-CONTRIB PROJECT OR ITS
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

A.2. ANTLR 3 License

```
The following software may be included in this product:
```

```
ANTLR 3
```

```
ANTLR 3 License
```

```
[The BSD License]
```

```
Copyright (c) 2003-2007, Terence Parr
```

```
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

```
* Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in
the documentation and/or other materials provided with the
distribution.
* Neither the name of the author nor the names of its contributors
may be used to endorse or promote products derived from this
software without specific prior written permission.
```

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

A.3. ANTLR 3.3 License

The following software may be included in this product:

ANTLR 3.3

ANTLR 3.3 License

Copyright (c) 2010 Terence Parr
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

Neither the name of the author nor the names of its contributors may be used
to endorse or promote products derived from this software without specific
prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

A.4. Boost Library License

The following software may be included in this product:

Boost C++ Libraries

Use of any of this software is governed by the terms of the license below:

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or
organization obtaining a copy of the software and accompanying

```
documentation covered by this license (the "Software") to use,
reproduce, display, distribute, execute, and transmit the Software,
and to prepare derivative works of the Software, and to permit
third-parties to whom the Software is furnished to do so, all
subject to the following:
```

```
The copyright notices in the Software and this entire statement,
including the above license grant, this restriction and the
following disclaimer, must be included in all copies of the
Software, in whole or in part, and all derivative works of the
Software, unless such copies or derivative works are solely in the
form of machine-executable object code generated by a source
language processor.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND
NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE
DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER
LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.
```

A.5. c3p0 JDBC Library License

You are receiving a copy of `c3p0-0.9.1-pre6.jar` in both source and object code in the following `/src/lib/c3p0-0.9.1-pre6.jar`. The terms of the Oracle license do NOT apply to `c3p0-0.9.1-pre6.jar`; it is licensed under the following license, separately from the Oracle programs you receive. If you do not wish to install this library, you may remove the file `/src/lib/c3p0-0.9.1-pre6.jar`, but the Oracle program might not operate properly or at all without the library.

This component is licensed under [Section A.13, "GNU Lesser General Public License Version 2.1, February 1999"](#).

A.6. dtoa.c License

The following software may be included in this product:

`dtoa.c`

```
The author of this software is David M. Gay.
```

```
Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
```

```
Permission to use, copy, modify, and distribute this software for
any purpose without fee is hereby granted, provided that this entire
notice is included in all copies of any software which is or includes
a copy or modification of this software and in all copies of the
supporting documentation for such software.
```

```
THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR
IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT
MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE
MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR
PURPOSE.
```

A.7. Editline Library (`libedit`) License

The following software may be included in this product:

Editline Library (`libedit`)

Some files are:

Copyright (c) 1992, 1993
The Regents of the University of California. All rights reserved.

This code is derived from software contributed to
Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain the
above copyright notice, this list of conditions
and the following disclaimer.
2. Redistributions in binary form must reproduce the
above copyright notice, this list of conditions and
the following disclaimer in the documentation and/or
other materials provided with the distribution.
3. Neither the name of the University nor the names of
its contributors may be used to endorse or promote
products derived from this software without specific
prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.

Some files are:

Copyright (c) 2001 The NetBSD Foundation, Inc.
All rights reserved.

This code is derived from software contributed to The NetBSD Foundation
by Anthony Mallet.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain the
above copyright notice, this list of conditions
and the following disclaimer.
2. Redistributions in binary form must reproduce the
above copyright notice, this list of conditions and the
following disclaimer in the documentation and/or
other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC.
AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER

```
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

Some files are:

```
Copyright (c) 1997 The NetBSD Foundation, Inc.
All rights reserved.
```

```
This code is derived from software contributed to The NetBSD Foundation
by Jaromir Dolecek.
```

```
Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

```
THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC.
AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

Some files are:

```
Copyright (c) 1998 Todd C. Miller <Todd.Miller@courtesan.com>
```

```
Permission to use, copy, modify, and distribute this
software for any purpose with or without fee is hereby
granted, provided that the above copyright notice and
this permission notice appear in all copies.
```

```
THE SOFTWARE IS PROVIDED "AS IS" AND TODD C. MILLER
DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL TODD C. MILLER BE LIABLE
FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION
OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

A.8. FindGTest.cmake License

The following software may be included in this product:

```
FindGTest.cmake helper script (part of CMake)

Copyright 2009 Kitware, Inc.
Copyright 2009 Philip Lowman
Copyright 2009 Daniel Blezek

Distributed under the OSI-approved BSD License (the "License");
see accompanying file Copyright.txt for details.

This software is distributed WITHOUT ANY WARRANTY; without even the
implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the License for more information.
=====
(To distributed this file outside of CMake, substitute the full
License text for the above reference.)

Thanks to Daniel Blezek for the GTEST_ADD_TESTS code

Text of Copyright.txt mentioned above:

CMake - Cross Platform Makefile Generator
Copyright 2000-2009 Kitware, Inc., Insight Software Consortium
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

* Neither the names of Kitware, Inc., the Insight Software Consortium,
nor the names of their contributors may be used to endorse or promote
products derived from this software without specific prior written
permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.9. Fred Fish's Dbug Library License

The following software may be included in this product:

Fred Fish's Dbug Library

N O T I C E

Copyright Abandoned, 1987, Fred Fish

This previously copyrighted work has been placed into the public domain by the author and may be freely used for any purpose, private or commercial.

Because of the number of inquiries I was receiving about the use of this product in commercially developed works I have decided to simply make it public domain to further its unrestricted use. I specifically would be most happy to see this material become a part of the standard Unix distributions by AT&T and the Berkeley Computer Science Research Group, and a standard part of the GNU system from the Free Software Foundation.

I would appreciate it, as a courtesy, if this notice is left in all copies and derivative works. Thank you.

The author makes no warranty of any kind with respect to this product and explicitly disclaims any implied warranties of merchantability or fitness for any particular purpose.

The `debug_analyze.c` file is subject to the following notice:

Copyright June 1987, Binayak Banerjee
All rights reserved.

This program may be freely distributed under the same terms and conditions as Fred Fish's Dbug package.

A.10. **getarg** License

The following software may be included in this product:

getarg Function (`getarg.h`, `getarg.c` files)

Copyright (c) 1997 - 2000 Kungliga Tekniska Högskolan
(Royal Institute of Technology, Stockholm, Sweden).
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Institute nor the names of its

contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.11. GLib License (for MySQL Proxy)

The following software may be included in this product:

GLib

You are receiving a copy of the GLib library in both source and object code in the following [proxy install dir]/lib/ and [proxy install dir]/licenses/lgpl folders. The terms of the Oracle license do NOT apply to the GLib library; it is licensed under the following license, separately from the Oracle programs you receive. If you do not wish to install this library, you may create an "exclude" file and run tar with the X option, as in the following example, but the Oracle program might not operate properly or at all without the library:

```
tar -xvfX <package-tar-file> <exclude-file>
```

where the exclude-file contains, e.g.:

```
<package-name>/lib/libglib-2.0.so.0.1600.6
<package-name>/lib/libglib-2.0.so.0
...
```

Example:

```
tar -xvfX mysql-proxy-0.8.1-solaris10-x86-64bit.tar.gz Exclude
```

Exclude File:

```
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libglib-2.0.so
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libglib-2.0.so.0
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libglib-2.0.so.0.1600.6
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgmodule-2.0.so
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgmodule-2.0.so.0
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgmodule-2.0.so.0.1600.6
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgthread-2.0.so
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgthread-2.0.so.0
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgthread-2.0.so.0.1600.6
mysql-proxy-0.8.1-solaris10-x86-64bit/licenses/lgpl/glib-2.16.6.tar.gz
```

This component is licensed under [Section A.13, "GNU Lesser General Public License Version 2.1, February 1999"](#).

A.12. GNU General Public License Version 2.0, June 1991

The following applies to all products licensed under the GNU General Public License, Version 2.0: You may not use the identified files except in compliance with the GNU General Public License, Version 2.0 (the "License.") You may obtain a copy of the License at <http://www.gnu.org/licenses/gpl-2.0.txt>. A copy of the license is

also reproduced below. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim
copies of this license document, but changing it is not
allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO

WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it
does.>
```

```
Copyright (C) <year> <name of author>
```

```
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version
2 of the License, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
02110-1301 USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'. This is free software, and you are welcome
to redistribute it under certain conditions; type 'show c'
for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever

suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
program 'Gnomovision' (which makes passes at compilers) written
by James Hacker.
```

```
<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

A.13. GNU Lesser General Public License Version 2.1, February 1999

The following applies to all products licensed under the GNU Lesser General Public License, Version 2.1: You may not use the identified files except in compliance with the GNU Lesser General Public License, Version 2.1 (the "License"). You may obtain a copy of the License at <http://www.gnu.org/licenses/lgpl-2.1.html>. A copy of the license is also reproduced below. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get

it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free

programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1

above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a

medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the

Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status

of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>  
Copyright (C) <year> <name of author>
```

```
This library is free software; you can redistribute it and/or  
modify it under the terms of the GNU Lesser General Public  
License as published by the Free Software Foundation; either  
version 2.1 of the License, or (at your option) any later version.
```

```
This library is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU  
Lesser General Public License for more details.
```

```
You should have received a copy of the GNU Lesser General Public  
License along with this library; if not, write to the Free Software  
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  
02110-1301 USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
```

```
library `Frob' (a library for tweaking knobs) written by James  
Random Hacker.
```

```
<signature of Ty Coon>, 1 April 1990  
Ty Coon, President of Vice
```

That's all there is to it!

A.14. GNU Libtool License

The following software may be included in this product:

GNU Libtool (The GNU Portable Library Tool)

If you are receiving a copy of the Oracle software in source code, you are also receiving a copy of two files (ltmain.sh and ltdl.h) generated by the GNU Libtool in source code. If you received the Oracle software under a license other than a commercial (non-GPL) license, then the terms of the Oracle license do NOT apply to these files from GNU Libtool; they are licensed under the following licenses, separately from the Oracle programs you receive.

Oracle elects to use GNU General Public License version 2 (GPL) for any software where a choice of GPL or GNU Lesser/Library General Public License (LGPL) license versions are made available with the language indicating that GPL/LGPL or any later version may be used, or where a choice of which version of the GPL/LGPL is applied is unspecified.

From GNU Libtool:

ltmain.sh - Provide generalized library-building support services.

NOTE: Changing this file will not affect anything until you rerun configure.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006, 2007 Free Software Foundation, Inc.
Originally by Gordon Matzigkeit, 1996

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

As a special exception to the GNU General Public License, if you distribute this file as part of a program that contains a configuration script generated by Autoconf, you may include it under the same distribution terms that you use for the rest of that program.

This component is licensed under [Section A.12, "GNU General Public License Version 2.0, June 1991"](#)

A.15. GNU Readline License

The following software may be included in this product:

GNU Readline Library

```
GNU Readline Library
With respect to MySQL Server/Cluster software licensed
under GNU General Public License, you are receiving a
copy of the GNU Readline Library in source code. The
terms of any Oracle license that might accompany the
Oracle programs do NOT apply to the GNU Readline Library;
it is licensed under the following license, separately
from the Oracle programs you receive. Oracle elects to
use GNU General Public License version 2 (GPL) for any
software where a choice of GPL license versions are
made available with the language indicating that GPLv2
or any later version may be used, or where a choice of
which version of the GPL is applied is unspecified.
```

This component is licensed under [Section A.12, "GNU General Public License Version 2.0, June 1991"](#)

A.16. Google Controlling Master Thread I/O Rate Patch License

The following software may be included in this product:

Google Controlling master thread I/O rate patch

```
Copyright (c) 2009, Google Inc.
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

A.17. Google Perftools (TCMalloc utility) License

The following software may be included in this product:

```
Google Perftools (TCMalloc utility)
Copyright (c) 1998-2006, Google Inc.
```

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.18. Google SMP Patch License

The following software may be included in this product:

Google SMP Patch

Google SMP patch

Copyright (c) 2008, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.19. jboss-common-jdbc-wrapper.jar License

You are receiving a copy of `jboss-common-jdbc-wrapper.jar` in both source and object code in the following `/src/lib/jboss-common-jdbc-wrapper.jar`. The terms of the Oracle license do NOT apply to `jboss-common-jdbc-wrapper.jar`; it is licensed under the following license, separately from the Oracle programs you receive. If you do not wish to install this library, you may remove the file `/src/lib/jboss-common-jdbc-wrapper.jar`, but the Oracle program might not operate properly or at all without the library.

This component is licensed under [Section A.13, "GNU Lesser General Public License Version 2.1, February 1999"](#).

A.20. lib_sql.cc License

The following software may be included in this product:

`lib_sql.cc`

```
Copyright (c) 2000
SWsoft company
```

```
This material is provided "as is", with absolutely no warranty
expressed or implied. Any use is at your own risk.
```

```
Permission to use or copy this software for any purpose is hereby
granted without fee, provided the above notices are retained on
all copies. Permission to modify the code and to distribute modified
code is granted, provided the above notices are retained, and a
notice that the code was modified is included with the above copyright
notice.
```

```
This code was modified by the MySQL team.
```

A.21. Libaio License

The following software may be included in this product:

`libaio`

This component is licensed under [Section A.13, "GNU Lesser General Public License Version 2.1, February 1999"](#).

A.22. libevent License

The following software may be included in this product:

`libevent`

```
Copyright (c) 2000-2007 Niels Provos <provos@citi.umich.edu>
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE
```

```
==
```

```
Parts developed by Adam Langley
```

```
==
```

```
==
```

```
log.c
```

```
Based on err.c, which was adapted from OpenBSD libc *err*warncode.
```

```
Copyright (c) 2005 Nick Mathewson
```

```
Copyright (c) 2000 Dug Song
```

```
Copyright (c) 1993 The Regents of the University of California.
```

```
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.
```

```
==
```

```
==
```

```
min_heap.h
```

```
Copyright (c) 2006 Maxim Yegorushkin
```

```
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote

```
products derived from this software without specific prior
written permission.
```

```
THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
==
```

```
==
```

```
win32.c
```

```
Copyright 2000-2002 Niels Provos
Copyright 2003 Michael A. Davis
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
==
```

A.23. Libiconv License

The following software may be included in this product:

```
Libiconv
```

```
You are receiving a copy of the GNU LIBICONV Library. The terms of the Oracle
license do NOT apply to the GNU LIBICONV Library; it is licensed under the
following license, separately from the Oracle programs you receive. If you do
not wish to install this program, you may delete [agent install
dir]/lib/libiconv.* and [agent install dir]/licenses/lgpl/iconv files.
```

This component is licensed under [Section A.13, "GNU Lesser General Public License Version 2.1, February 1999"](#).

A.24. libintl License

The following software may be included in this product:

libintl

```
Copyright (C) 1994 X Consortium
```

```
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:
```

```
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM
BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
Except as contained in this notice, the name of the X Consortium shall not be
used in advertising or otherwise to promote the sale, use or other dealings in
this Software without prior written authorization from the X Consortium.
```

```
FSF changes to this file are in the public domain.
```

```
Copyright 1996-2007 Free Software Foundation, Inc. Taken from GNU libtool, 2001
```

```
Originally by Gordon Matzigkeit <gord@gnu.ai.mit.edu>, 1996
```

```
This file is free software; the Free Software Foundation gives unlimited
permission to copy and/or distribute it, with or without modifications, as long
as this notice is preserved.
```

```
You are receiving a copy of the libintl library. The terms of the Oracle license
do NOT apply to the libintl library; it is licensed under the following license,
separately from the Oracle programs you receive. If you do not wish to install
this program, you may create an "exclude" file and run tar with the X option.
```

```
This component is licensed under Section A.13, "GNU Lesser General Public License Version 2.1, February 1999".
```

A.25. Linux-PAM License

The following software may be included in this product:

```
Linux-PAM (pam-devel, Pluggable authentication modules for Linux)
```

```
Copyright Theodore Ts'o, 1996. All rights reserved.
```

```
(For the avoidance of doubt, Oracle uses and distributes this
component under the terms below and elects not to do so under
the GPL even though the GPL is referenced as an option below.)
```

```
Redistribution and use in source and binary forms, with or
without modification, are permitted provided that the following
conditions are met:
```

1. Redistributions of source code must retain the above copyright notice, and the entire permission notice in its entirety,

- including the disclaimer of warranties.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
 3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

ALTERNATIVELY, this product may be distributed under the terms of the GNU Public License, in which case the provisions of the GPL are required INSTEAD OF the above restrictions. (This clause is necessary due to a potential bad interaction between the GPL and the restrictions contained in a BSD-style copyright.)

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.26. LPeg Library License

The following software may be included in this product:

LPeg

Use of any of this software is governed by the terms of the license below:

Copyright © 2008 Lua.org, PUC-Rio.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.27. Lua (liblua) License

The following software may be included in this product:

Lua (liblua)

Copyright © 1994-2008 Lua.org, PUC-Rio.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including

```
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject
to the following conditions:
```

```
The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
```

A.28. LuaFileSystem Library License

The following software may be included in this product:

```
LuaFileSystem
```

```
Copyright © 2003 Kepler Project.
```

```
Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject
to the following conditions:
```

```
The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

A.29. md5 (Message-Digest Algorithm 5) License

The following software may be included in this product:

```
md5 (Message-Digest Algorithm 5)
```

```
This code implements the MD5 message-digest algorithm.
The algorithm is due to Ron Rivest. This code was
written by Colin Plumb in 1993, no copyright is claimed.
This code is in the public domain; do with it what you wish.
```

```
Equivalent code is available from RSA Data Security, Inc.
This code has been tested against that, and is equivalent,
except that you don't need to include two pages of legalese
with every copy.
```

```
The code has been modified by Mikael Ronstroem to handle
calculating a hash value of a key that is always a multiple
of 4 bytes long. Word 0 of the calculated 4-word hash value
is returned as the hash value.
```

A.30. memcached License

The following software may be included in this product:

memcached

```
Copyright (c) 2003, Danga Interactive, Inc.
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

```
* Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
```

```
* Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the
distribution.
```

```
* Neither the name of the Danga Interactive nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.
```

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.31. nt_servc (Windows NT Service class library) License

The following software may be included in this product:

nt_servc (Windows NT Service class library)

```
Windows NT Service class library
Copyright Abandoned 1998 Irena Pancirov - Irnet Snc
This file is public domain and comes with NO WARRANTY of any kind
```

A.32. OpenPAM License

The following software may be included in this product:

OpenPAM

```
Copyright (c) 2002-2003 Networks Associates Technology, Inc.
Copyright (c) 2004-2007 Dag-Erling Smørgrav
All rights reserved.
```

```
This software was developed for the FreeBSD Project by
ThinkSec AS and Network Associates Laboratories, the
Security Research Division of Network Associates, Inc.
under DARPA/SPAWAR contract N66001-01-C-8035 ("CBOSS"),
as part of the DARPA CHATS research program.
```

```
Redistribution and use in source and binary forms,
```

with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.33. Paramiko License

The following software may be included in this product:

Paramiko

You are receiving a copy of Paramiko in both source and object code. The terms of the Oracle license do NOT apply to the Paramiko program; it is licensed under the following license, separately from the Oracle programs you receive. If you do not wish to install this program, you may delete the Paramiko folder and all its contents.

This component is licensed under [Section A.13, "GNU Lesser General Public License Version 2.1, February 1999"](#).

A.34. PCRE License

The following software may be included in this product:

PCRE (Perl Compatible Regular Expressions) Library

PCRE LICENCE

PCRE is a library of functions to support regular expressions whose syntax and semantics are as close as possible to those of the Perl 5 language.

Release 7 of PCRE is distributed under the terms of the "BSD" licence, as specified below. The documentation for PCRE, supplied in the "doc" directory, is distributed under the same terms as the software itself.

The basic library functions are written in C and are freestanding. Also included in the distribution is a set of C++ wrapper functions.

THE BASIC LIBRARY FUNCTIONS

```
-----  
Written by:      Philip Hazel  
Email local part: ph10  
Email domain:   cam.ac.uk  
  
University of Cambridge Computing Service,  
Cambridge, England. Phone: +44 1223 334714.  
  
Copyright (c) 1997-2006 University of Cambridge  
All rights reserved.  
  
THE C++ WRAPPER FUNCTIONS  
-----  
Contributed by:  Google Inc.  
  
Copyright (c) 2006, Google Inc.  
All rights reserved.  
  
THE "BSD" LICENCE  
-----  
  
Redistribution and use in source and binary forms,  
with or without modification, are permitted provided  
that the following conditions are met:  
  
* Redistributions of source code must retain the above  
  copyright notice, this list of conditions and the  
  following disclaimer.  
* Redistributions in binary form must reproduce the  
  above copyright notice, this list of conditions and  
  the following disclaimer in the documentation and/or  
  other materials provided with the distribution.  
* Neither the name of the University of Cambridge nor  
  the name of Google Inc. nor the names of their contributors  
  may be used to endorse or promote products derived from  
  this software without specific prior written permission.  
  
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND  
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,  
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF  
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE  
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS  
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,  
OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT  
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;  
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF  
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF  
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
  
End
```

A.35. Percona Multiple I/O Threads Patch License

The following software may be included in this product:

Percona Multiple I/O threads patch

```
Copyright (c) 2008, 2009 Percona Inc  
All rights reserved.  
  
Redistribution and use of this software in source and binary forms,  
with or without modification, are permitted provided that the  
following conditions are met:  
  
* Redistributions of source code must retain the above copyright
```

notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
 * Neither the name of Percona Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission of Percona Inc.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.36. Python License

The following software may be included in this product:

Python Programming Language

This is the official license for the Python 2.7 release:

A. HISTORY OF THE SOFTWARE

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see <http://www.cwi.nl>) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see <http://www.cnri.reston.va.us>) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation, see <http://www.zope.com>). In 2001, the Python Software Foundation (PSF, see <http://www.python.org/psf/>) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see <http://www.opensource.org> for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

| Release | Derived from | Year | Owner | GPL-compatible? (1) |
|----------------|--------------|-----------|------------|---------------------|
| 0.9.0 thru 1.2 | | 1991-1995 | CWI | yes |
| 1.3 thru 1.5.2 | 1.2 | 1995-1999 | CNRI | yes |
| 1.6 | 1.5.2 | 2000 | CNRI | no |
| 2.0 | 1.6 | 2000 | BeOpen.com | no |
| 1.6.1 | 1.6 | 2001 | CNRI | yes (2) |
| 2.1 | 2.0+1.6.1 | 2001 | PSF | no |
| 2.0.1 | 2.0+1.6.1 | 2001 | PSF | yes |
| 2.1.1 | 2.1+2.0.1 | 2001 | PSF | yes |
| 2.2 | 2.1.1 | 2001 | PSF | yes |

Python License

| | | | | |
|-------|-------|-----------|-----|-----|
| 2.1.2 | 2.1.1 | 2002 | PSF | yes |
| 2.1.3 | 2.1.2 | 2002 | PSF | yes |
| 2.2.1 | 2.2 | 2002 | PSF | yes |
| 2.2.2 | 2.2.1 | 2002 | PSF | yes |
| 2.2.3 | 2.2.2 | 2003 | PSF | yes |
| 2.3 | 2.2.2 | 2002-2003 | PSF | yes |
| 2.3.1 | 2.3 | 2002-2003 | PSF | yes |
| 2.3.2 | 2.3.1 | 2002-2003 | PSF | yes |
| 2.3.3 | 2.3.2 | 2002-2003 | PSF | yes |
| 2.3.4 | 2.3.3 | 2004 | PSF | yes |
| 2.3.5 | 2.3.4 | 2005 | PSF | yes |
| 2.4 | 2.3 | 2004 | PSF | yes |
| 2.4.1 | 2.4 | 2005 | PSF | yes |
| 2.4.2 | 2.4.1 | 2005 | PSF | yes |
| 2.4.3 | 2.4.2 | 2006 | PSF | yes |
| 2.5 | 2.4 | 2006 | PSF | yes |
| 2.5.1 | 2.5 | 2007 | PSF | yes |
| 2.5.2 | 2.5.1 | 2008 | PSF | yes |
| 2.5.3 | 2.5.2 | 2008 | PSF | yes |
| 2.6 | 2.5 | 2008 | PSF | yes |
| 2.6.1 | 2.6 | 2008 | PSF | yes |
| 2.6.2 | 2.6.1 | 2009 | PSF | yes |
| 2.6.3 | 2.6.2 | 2009 | PSF | yes |
| 2.6.4 | 2.6.3 | 2010 | PSF | yes |
| 2.7 | 2.6 | 2010 | PSF | yes |

Footnotes:

- (1) GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.
- (2) According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of

Python License

the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant

Python License

permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright (c) 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>".

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License

Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Licenses and Acknowledgements for Incorporated Software
=====

This section is an incomplete, but growing list of licenses and acknowledgements for third-party software incorporated in the Python distribution.

Mersenne Twister
=====

The `_random` module includes code based on a download from <http://www.math.keio.ac.jp/matsumoto/MT2002/emt19937ar.html>. The following are the verbatim comments from the original code:

A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using `init_genrand(seed)` or `init_by_array(init_key, key_length)`.

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright

notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any feedback is very welcome.
<http://www.math.keio.ac.jp/matamoto/emt.html>
email: matamoto@math.keio.ac.jp

Sockets
=====

The socket module uses the functions, `getaddrinfo()`, and `getnameinfo()`, which are coded in separate source files from the WIDE Project, <http://www.wide.ad.jp/>.

Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Floating point exception control

=====
The source for the `fpectl` module includes the following notice:

```
-----  
/                               Copyright (c) 1996.                               \  
|                               The Regents of the University of California.          \  
|                               All rights reserved.                                \  
|  
| Permission to use, copy, modify, and distribute this software for  
| any purpose without fee is hereby granted, provided that this en-
```

tire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software.

This work was produced at the University of California, Lawrence Livermore National Laboratory under contract no. W-7405-ENG-48 between the U.S. Department of Energy and The Regents of the University of California for the operation of UC LLNL.

DISCLAIMER

This software was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

MD5 message digest algorithm
=====

The source code for the md5 module contains the following notice:

Copyright (C) 1999, 2002 Aladdin Enterprises. All rights reserved.

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

L. Peter Deutsch
ghost@aladdin.com

Independent implementation of MD5 (RFC 1321).

This code implements the MD5 Algorithm defined in RFC 1321, whose text is available at

<http://www.ietf.org/rfc/rfc1321.txt>

The code is derived from the text of the RFC, including the test suite (section A.5) but excluding the rest of Appendix A. It does not include any code or documentation that is identified in the RFC as being copyrighted.

The original and principal author of md5.h is L. Peter Deutsch <ghost@aladdin.com>. Other authors are noted in the change history

that follows (in reverse chronological order):

2002-04-13 lpd Removed support for non-ANSI compilers; removed references to Ghostscript; clarified derivation from RFC 1321; now handles byte order either statically or dynamically.
1999-11-04 lpd Edited comments slightly for automatic TOC extraction.
1999-10-18 lpd Fixed typo in header comment (ansi2knr rather than md5); added conditionalization for C++ compilation from Martin Purschke <purschke@bnl.gov>.
1999-05-03 lpd Original version.

Asynchronous socket services
=====

The asynchat and asyncore modules contain the following notice:

Copyright 1996 by Sam Rushing

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Sam Rushing not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Cookie management
=====

The Cookie module contains the following notice:

Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Timothy O'Malley not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Profiling
=====

The profile and pstats modules contain the following notice:

Copyright 1994, by InfoSeek Corporation, all rights reserved.

Python License

Written by James Roskind

Permission to use, copy, modify, and distribute this Python software and its associated documentation for any purpose (subject to the restriction in the following sentence) without fee is hereby granted, provided that the above copyright notice appears in all copies, and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of InfoSeek not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. This permission is explicitly restricted to the copying and modification of the software to remain in Python, compiled Python, or other languages (such as C) wherein the modified or derived code is exclusively imported into a Python module.

INFOSEEK CORPORATION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL INFOSEEK CORPORATION BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Execution tracing

=====

The trace module contains the following notice:

portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err... reserved and offered to the public under the terms of the
Python 2.2 license.

Author: Zooko O'Whielacronx

<http://zooko.com/>

<mailto:zooko@zooko.com>

Copyright 2000, Mojam Media, Inc., all rights reserved.

Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.

Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.

Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.

Permission to use, copy, modify, and distribute this Python software and its associated documentation for any purpose without fee is hereby granted, provided that the above copyright notice appears in all copies, and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of neither Automatrix, Bioreason or Mojam Media be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

UUencode and UUdecode functions

=====

The uu module contains the following notice:

Copyright 1994 by Lance Ellinghouse

Cathedral City, California Republic, United States of America.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Lance Ellinghouse not be used in advertising or publicity pertaining to distribution

of the software without specific, written prior permission.
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Modified by Jack Jansen, CWI, July 1995:

- Use binascii module to do the actual line-by-line conversion between ascii and binary. This results in a 1000-fold speedup. The C version is still 5 times faster, though.
- Arguments more compliant with Python standard

XML Remote Procedure Calls¶

The xmlrpclib module contains the following notice:

The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its associated documentation, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its associated documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies, and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Secret Labs AB or the author not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

test_epoll
=====

The test_epoll contains the following notice:

Copyright (c) 2001-2006 Twisted Matrix Laboratories.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE

```
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
Select kqueue
=====
```

The select and contains the following notice for the kqueue interface:

Copyright (c) 2000 Doug White, 2006 James Knight, 2007 Christian Heimes
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

```
strtod and dtoa
=====
```

The file Python/dtoa.c, which supplies C functions dtoa and strtod for conversion
of C doubles to and from strings, is derived from the file of the same name by
David M. Gay, currently available from <http://www.netlib.org/fp/>. The original
file, as retrieved on March 16, 2009, contains the following copyright and
licensing notice:

```
/*
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
 *
 * Permission to use, copy, modify, and distribute this software for
 * any purpose without fee is hereby granted, provided that this entire
 * notice is included in all copies of any software which is or
 * includes a copy or modification of this software and in all copies
 * of the supporting documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT
 * MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE
 * MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR
 * PURPOSE.
 */
```

A.37. Red HAT RPM Spec File License

The following software may be included in this product:

Red Hat RPM Spec File

You are receiving a copy of the Red Hat spec file. The terms of the Oracle license do NOT apply to the Red Hat spec file; it is licensed under the following license, separately from the Oracle programs you receive.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
[for rest of text, see following link]

This component is licensed under [Section A.12, "GNU General Public License Version 2.0, June 1991"](#)

A.38. RegEX-Spencer Library License

The following software may be included in this product: Henry Spencer's Regular-Expression Library (RegEX-Spencer)

Copyright 1992, 1993, 1994 Henry Spencer. All rights reserved.
This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it, subject to the following restrictions:

1. The author is not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
4. This notice may not be removed or altered.

A.39. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) License

The following software may be included in this product:

RFC 3174 - US Secure Hash Algorithm 1 (SHA1)

RFC 3174 - US Secure Hash Algorithm 1 (SHA1)

Copyright (C) The Internet Society (2001). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

```
The limited permissions granted above are perpetual and will not be
revoked by the Internet Society or its successors or assigns.
```

```
This document and the information contained herein is provided on an
"AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING
TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
```

Acknowledgement

```
Funding for the RFC Editor function is currently provided by the
Internet Society.
```

A.40. Richard A. O'Keefe String Library License

The following software may be included in this product:

Richard A. O'Keefe String Library

```
The Richard O'Keefe String Library is subject to the following notice:
```

```
These files are in the public domain. This includes getopt.c, which
is the work of Henry Spencer, University of Toronto Zoology, who
says of it "None of this software is derived from Bell software. I
had no access to the source for Bell's versions at the time I wrote
it. This software is hereby explicitly placed in the public domain.
It may be used for any purpose on any machine by anyone." I would
greatly prefer it if *my* material received no military use.
```

```
The t_ctype.h file is subject to the following notice:
```

```
Copyright (C) 1998, 1999 by Pruet Boonma, all rights reserved.
Copyright (C) 1998 by Theppitak Karoonboonyanan, all rights reserved.
```

```
Permission to use, copy, modify, distribute and sell this software and its
documentation for any purpose is hereby granted without fee, provided that the above
copyright notice appear in all copies.
```

```
Smaphan Raruenrom and Pruet Boonma makes no representations about
the suitability of this software for any purpose. It is provided
"as is" without express or implied warranty.
```

A.41. SHA-1 in C License

The following software may be included in this product:

SHA-1 in C

```
SHA-1 in C
By Steve Reid <steve@edmweb.com>
100% Public Domain
```

A.42. Simple Logging Facade for Java (SLF4J) License

The following software may be included in this product:

```
Simple Logging Facade for Java (SLF4J)
```

```
Copyright (c) 2004-2008 QOS.ch
All rights reserved.
```

```
Permission is hereby granted, free of charge,
to any person obtaining a copy of this software
and associated documentation files (the "Software"),
to deal in the Software without restriction, including
without limitation the rights to use, copy, modify,
merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom
the Software is furnished to do so, subject to the
following conditions:
```

```
The above copyright notice and this permission notice
shall be included in all copies or substantial portions
of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY
OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT
LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.
```

A.43. **zlib** License

The following software may be included in this product:

zlib

Oracle gratefully acknowledges the contributions of Jean-loup Gailly and Mark Adler in creating the zlib general purpose compression library which is used in this product.

```
zlib.h -- interface of the 'zlib' general purpose compression library
Copyright (C) 1995-2004 Jean-loup Gailly and Mark Adler
```

```
zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.3, July 18th, 2005
Copyright (C) 1995-2005 Jean-loup Gailly and Mark Adler
```

```
zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.5, April 19th, 2010
Copyright (C) 1995-2010 Jean-loup Gailly and Mark Adler
```

```
This software is provided 'as-is', without any express or implied warranty.
In no event will the authors be held liable for any damages arising from the
use of this software. Permission is granted to anyone to use this software
for any purpose, including commercial applications, and to alter it and
redistribute it freely, subject to the following restrictions:
```

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

```
Jean-loup Gailly jloup@gzip.org
Mark Adler madler@alumni.caltech.edu
```

A.44. **ZLIB.NET** License

The following software may be included in this product:

ZLIB.NET

Copyright (c) 2006-2007, ComponentAce
<http://www.componentace.com>
All rights reserved.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

- * Redistributions of source code must retain the
above copyright notice, this list of conditions and
the following disclaimer.
- * Redistributions in binary form must reproduce the
above copyright notice, this list of conditions and
the following disclaimer in the documentation and/or
other materials provided with the distribution.
- * Neither the name of ComponentAce nor the names of its
contributors may be used to endorse or promote products
derived from this software without specific prior written
permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Appendix B. MySQL 5.5 Frequently Asked Questions

Table of Contents

| | |
|---|------|
| B.1. MySQL 5.5 FAQ: General | 3931 |
| B.2. MySQL 5.5 FAQ: Storage Engines | 3933 |
| B.3. MySQL 5.5 FAQ: Server SQL Mode | 3934 |
| B.4. MySQL 5.5 FAQ: Stored Procedures and Functions | 3935 |
| B.5. MySQL 5.5 FAQ: Triggers | 3939 |
| B.6. MySQL 5.5 FAQ: Views | 3941 |
| B.7. MySQL 5.5 FAQ: INFORMATION_SCHEMA | 3942 |
| B.8. MySQL 5.5 FAQ: Migration | 3943 |
| B.9. MySQL 5.5 FAQ: Security | 3944 |
| B.10. MySQL FAQ: MySQL 5.5 and MySQL Cluster | 3945 |
| B.11. MySQL 5.5 FAQ: MySQL Chinese, Japanese, and Korean Character Sets | 3957 |
| B.12. MySQL 5.5 FAQ: Connectors & APIs | 3969 |
| B.13. MySQL 5.5 FAQ: Replication | 3969 |
| B.14. MySQL 5.5 FAQ: MySQL Enterprise Scalability Thread Pool | 3973 |

B.1. MySQL 5.5 FAQ: General

Questions

- [B.1.1: \[3931\]](#) Which version of MySQL is production-ready (GA)?
- [B.1.2: \[3932\]](#) What is the state of development (non-GA) versions?
- [B.1.3: \[3932\]](#) Can MySQL 5.5 do subqueries?
- [B.1.4: \[3932\]](#) Can MySQL 5.5 perform multiple-table inserts, updates, and deletes?
- [B.1.5: \[3932\]](#) Does MySQL 5.5 have a Query Cache? Does it work on Server, Instance or Database?
- [B.1.6: \[3932\]](#) Does MySQL 5.5 have Sequences?
- [B.1.7: \[3932\]](#) Does MySQL 5.5 have a `NOW()` [\[1143\]](#) function with fractions of seconds?
- [B.1.8: \[3932\]](#) Does MySQL 5.5 work with multi-core processors?
- [B.1.9: \[3933\]](#) Why do I see multiple processes for `mysqld`?
- [B.1.10: \[3933\]](#) Have there been there any improvements in error reporting when foreign keys fail? Does MySQL now report which column and reference failed?
- [B.1.11: \[3933\]](#) Can MySQL 5.5 perform ACID transactions?

Questions and Answers

B.1.1: Which version of MySQL is production-ready (GA)?

At the time of writing this (December 2011), MySQL 5.5, MySQL 5.1, and MySQL 5.0 are supported for production use.

MySQL 5.5 achieved General Availability (GA) status with MySQL 5.5.8, which was released for production use on 3 December 2010.

MySQL 5.1 achieved General Availability (GA) status with MySQL 5.1.30, which was released for production use on 14 November 2008.

MySQL 5.0 achieved General Availability (GA) status with MySQL 5.0.15, which was released for production use on 19 October 2005. Note that active development for MySQL 5.0 has ended.

B.1.2: What is the state of development (non-GA) versions?

MySQL follows a milestone release model that introduces pre-production-quality features and stabilizes them to release quality (see <http://dev.mysql.com/doc/mysql-development-cycle/en/index.html>). This process then repeats, so releases cycle between pre-production and release quality status. Please check the change logs to identify the status of a given release.

MySQL 5.4 was a development series. Work on this series has ceased.

MySQL 5.5 is being actively developed using the milestone release methodology described above.

MySQL 5.6 is being actively developed using the milestone release methodology described above.

MySQL 5.7 is being actively developed using the milestone release methodology described above.

MySQL 6.0 was a development series. Work on this series has ceased.

B.1.3: Can MySQL 5.5 do subqueries?

Yes. See [Section 13.2.10, "Subquery Syntax"](#).

B.1.4: Can MySQL 5.5 perform multiple-table inserts, updates, and deletes?

Yes. For the syntax required to perform multiple-table updates, see [Section 13.2.11, "UPDATE Syntax"](#); for that required to perform multiple-table deletes, see [Section 13.2.2, "DELETE Syntax"](#).

A multiple-table insert can be accomplished using a trigger whose `FOR EACH ROW` clause contains multiple `INSERT` statements within a `BEGIN . . . END` block. See [Section 19.3, "Using Triggers"](#).

B.1.5: Does MySQL 5.5 have a Query Cache? Does it work on Server, Instance or Database?

Yes. The query cache operates on the server level, caching complete result sets matched with the original query string. If an exactly identical query is made (which often happens, particularly in web applications), no parsing or execution is necessary; the result is sent directly from the cache. Various tuning options are available. See [Section 8.9.3, "The MySQL Query Cache"](#).

B.1.6: Does MySQL 5.5 have Sequences?

No. However, MySQL has an `AUTO_INCREMENT` system, which in MySQL 5.5 can also handle inserts in a multi-master replication setup. With the `auto_increment_increment` [1865] and `auto_increment_offset` [1868] system variables, you can set each server to generate auto-increment values that don't conflict with other servers. The `auto_increment_increment` [1865] value should be greater than the number of servers, and each server should have a unique offset.

B.1.7: Does MySQL 5.5 have a `NOW()` [1143] function with fractions of seconds?

No. This is on the MySQL roadmap as a "rolling feature". This means that it is not a flagship feature, but will be implemented, development time permitting. Specific customer demand may change this scheduling.

However, MySQL does parse time strings with a fractional component. See [Section 11.3.2, "The `TIME` Type"](#).

B.1.8: Does MySQL 5.5 work with multi-core processors?

Yes. MySQL is fully multi-threaded, and will make use of multiple CPUs, provided that the operating system supports them.

B.1.9: Why do I see multiple processes for `mysqld`?

When using LinuxThreads, you should see a minimum of three `mysqld` processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

B.1.10: Have there been any improvements in error reporting when foreign keys fail? Does MySQL now report which column and reference failed?

The foreign key support in `InnoDB` has seen improvements in each major version of MySQL. Foreign key support generic to all storage engines is scheduled for MySQL 6.x; this should resolve any inadequacies in the current storage engine specific implementation.

B.1.11: Can MySQL 5.5 perform ACID transactions?

Yes. All current MySQL versions support transactions. The `InnoDB` storage engine offers full ACID transactions with row-level locking, multi-versioning, nonlocking repeatable reads, and all four SQL standard isolation levels.

The `NDB` storage engine supports the `READ COMMITTED [1402]` transaction isolation level only.

B.2. MySQL 5.5 FAQ: Storage Engines

Questions

- **B.2.1: [3933]** Where can I obtain complete documentation for MySQL storage engines?
- **B.2.2: [3933]** Are there any new storage engines in MySQL 5.5?
- **B.2.3: [3933]** Have any storage engines been removed in MySQL 5.5?
- **B.2.4: [3934]** What are the unique benefits of the `ARCHIVE` storage engine?
- **B.2.5: [3934]** Do the new features in MySQL 5.5 apply to all storage engines?

Questions and Answers

B.2.1: Where can I obtain complete documentation for MySQL storage engines?

See [Chapter 14, *Storage Engines*](#). That chapter contains information about all MySQL storage engines except for the `NDB` storage engine used for MySQL Cluster; `NDB` is covered in [Chapter 17, *MySQL Cluster NDB 7.2*](#).

B.2.2: Are there any new storage engines in MySQL 5.5?

The features from the optional `InnoDB Plugin` from MySQL 5.1 are folded into the built-in `InnoDB` storage engine, so you can take advantage of features such as the Barracuda file format, `InnoDB` table compression, and the new configuration options for performance. See [Section 14.4, “New Features of InnoDB 1.1”](#) for details. `InnoDB` also becomes the default storage engine for new tables. See [Section 14.3.1, “InnoDB as the Default MySQL Storage Engine”](#) for details.

B.2.3: Have any storage engines been removed in MySQL 5.5?

No.

B.2.4: What are the unique benefits of the [ARCHIVE](#) storage engine?

The [ARCHIVE](#) storage engine is ideally suited for storing large amounts of data without indexes; it has a very small footprint, and performs selects using table scans. See [Section 14.8](#), “The [ARCHIVE Storage Engine](#)”, for details.

B.2.5: Do the new features in MySQL 5.5 apply to all storage engines?

The general new features such as views, stored procedures, triggers, [INFORMATION_SCHEMA](#), precision math ([DECIMAL](#) column type), and the [BIT](#) column type, apply to all storage engines. There are also additions and changes for specific storage engines.

B.3. MySQL 5.5 FAQ: Server SQL Mode

Questions

- [B.3.1: \[3934\]](#) What are server SQL modes?
- [B.3.2: \[3934\]](#) How many server SQL modes are there?
- [B.3.3: \[3934\]](#) How do you determine the server SQL mode?
- [B.3.4: \[3934\]](#) Is the mode dependent on the database or connection?
- [B.3.5: \[3934\]](#) Can the rules for strict mode be extended?
- [B.3.6: \[3935\]](#) Does strict mode impact performance?
- [B.3.7: \[3935\]](#) What is the default server SQL mode when My SQL 5.5 is installed?

Questions and Answers**B.3.1: What are server SQL modes?**

Server SQL modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers. The MySQL Server apply these modes individually to different clients. For more information, see [Section 5.1.7](#), “[Server SQL Modes](#)”.

B.3.2: How many server SQL modes are there?

Each mode can be independently switched on and off. See [Section 5.1.7](#), “[Server SQL Modes](#)”, for a complete list of available modes.

B.3.3: How do you determine the server SQL mode?

You can set the default SQL mode (for `mysqld` startup) with the `--sql-mode` [\[464\]](#) option. Using the statement `SET [GLOBAL|SESSION] sql_mode='modes'`, you can change the settings from within a connection, either locally to the connection, or to take effect globally. You can retrieve the current mode by issuing a `SELECT @@sql_mode` statement.

B.3.4: Is the mode dependent on the database or connection?

A mode is not linked to a particular database. Modes can be set locally to the session (connection), or globally for the server. you can change these settings using `SET [GLOBAL|SESSION] sql_mode='modes'`.

B.3.5: Can the rules for strict mode be extended?

When we refer to *strict mode*, we mean a mode where at least one of the modes [TRADITIONAL \[618\]](#), [STRICT_TRANS_TABLES \[617\]](#), or [STRICT_ALL_TABLES \[616\]](#) is enabled. Options can be combined, so you can add restrictions to a mode. See [Section 5.1.7, “Server SQL Modes”](#), for more information.

B.3.6: Does strict mode impact performance?

The intensive validation of input data that some settings requires more time than if the validation is not done. While the performance impact is not that great, if you do not require such validation (perhaps your application already handles all of this), then MySQL gives you the option of leaving strict mode disabled. However—if you do require it—strict mode can provide such validation.

B.3.7: What is the default server SQL mode when MySQL 5.5 is installed?

By default, no special modes are enabled. See [Section 5.1.7, “Server SQL Modes”](#), for information about all available modes and MySQL's default behavior.

B.4. MySQL 5.5 FAQ: Stored Procedures and Functions

Questions

- [B.4.1: \[3936\]](#) Does MySQL 5.5 support stored procedures and functions?
- [B.4.2: \[3936\]](#) Where can I find documentation for MySQL stored procedures and stored functions?
- [B.4.3: \[3936\]](#) Is there a discussion forum for MySQL stored procedures?
- [B.4.4: \[3936\]](#) Where can I find the ANSI SQL 2003 specification for stored procedures?
- [B.4.5: \[3936\]](#) How do you manage stored routines?
- [B.4.6: \[3936\]](#) Is there a way to view all stored procedures and stored functions in a given database?
- [B.4.7: \[3937\]](#) Where are stored procedures stored?
- [B.4.8: \[3937\]](#) Is it possible to group stored procedures or stored functions into packages?
- [B.4.9: \[3937\]](#) Can a stored procedure call another stored procedure?
- [B.4.10: \[3937\]](#) Can a stored procedure call a trigger?
- [B.4.11: \[3937\]](#) Can a stored procedure access tables?
- [B.4.12: \[3937\]](#) Do stored procedures have a statement for raising application errors?
- [B.4.13: \[3937\]](#) Do stored procedures provide exception handling?
- [B.4.14: \[3937\]](#) Can MySQL 5.5 stored routines return result sets?
- [B.4.15: \[3937\]](#) Is [WITH RECOMPILE](#) supported for stored procedures?
- [B.4.16: \[3937\]](#) Is there a MySQL equivalent to using [mod_plsql](#) as a gateway on Apache to talk directly to a stored procedure in the database?
- [B.4.17: \[3937\]](#) Can I pass an array as input to a stored procedure?
- [B.4.18: \[3937\]](#) Can I pass a cursor as an [IN](#) parameter to a stored procedure?
- [B.4.19: \[3938\]](#) Can I return a cursor as an [OUT](#) parameter from a stored procedure?

- [B.4.20: \[3938\]](#) Can I print out a variable's value within a stored routine for debugging purposes?
- [B.4.21: \[3938\]](#) Can I commit or roll back transactions inside a stored procedure?
- [B.4.22: \[3938\]](#) Do MySQL 5.5 stored procedures and functions work with replication?
- [B.4.23: \[3938\]](#) Are stored procedures and functions created on a master server replicated to a slave?
- [B.4.24: \[3938\]](#) How are actions that take place inside stored procedures and functions replicated?
- [B.4.25: \[3938\]](#) Are there special security requirements for using stored procedures and functions together with replication?
- [B.4.26: \[3938\]](#) What limitations exist for replicating stored procedure and function actions?
- [B.4.27: \[3939\]](#) Do the preceding limitations affect MySQL's ability to do point-in-time recovery?
- [B.4.28: \[3939\]](#) What is being done to correct the aforementioned limitations?

Questions and Answers

B.4.1: Does MySQL 5.5 support stored procedures and functions?

Yes. MySQL 5.5 supports two types of stored routines—stored procedures and stored functions.

B.4.2: Where can I find documentation for MySQL stored procedures and stored functions?

See [Section 19.2, “Using Stored Routines \(Procedures and Functions\)”](#).

B.4.3: Is there a discussion forum for MySQL stored procedures?

Yes. See <http://forums.mysql.com/list.php?98>.

B.4.4: Where can I find the ANSI SQL 2003 specification for stored procedures?

Unfortunately, the official specifications are not freely available (ANSI makes them available for purchase). However, there are books—such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer—which give a comprehensive overview of the standard, including coverage of stored procedures.

B.4.5: How do you manage stored routines?

It is always good practice to use a clear naming scheme for your stored routines. You can manage stored procedures with `CREATE [FUNCTION|PROCEDURE]`, `ALTER [FUNCTION|PROCEDURE]`, `DROP [FUNCTION|PROCEDURE]`, and `SHOW CREATE [FUNCTION|PROCEDURE]`. You can obtain information about existing stored procedures using the `ROUTINES` table in the `INFORMATION_SCHEMA` database (see [Section 20.17, “The INFORMATION_SCHEMA ROUTINES Table”](#)).

B.4.6: Is there a way to view all stored procedures and stored functions in a given database?

Yes. For a database named `dbname`, use this query on the `INFORMATION_SCHEMA.ROUTINES` table:

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_SCHEMA='dbname';
```

For more information, see [Section 20.17, “The INFORMATION_SCHEMA ROUTINES Table”](#).

The body of a stored routine can be viewed using `SHOW CREATE FUNCTION` (for a stored function) or `SHOW CREATE PROCEDURE` (for a stored procedure). See [Section 13.7.5.11, “SHOW CREATE PROCEDURE Syntax”](#), for more information.

B.4.7: Where are stored procedures stored?

In the `proc` table of the `mysql` system database. However, you should not access the tables in the system database directly. Instead, use `SHOW CREATE FUNCTION` to obtain information about stored functions, and `SHOW CREATE PROCEDURE` to obtain information about stored procedures. See [Section 13.7.5.11](#), “`SHOW CREATE PROCEDURE Syntax`”, for more information about these statements.

You can also query the `ROUTINES` table in the `INFORMATION_SCHEMA` database—see [Section 20.17](#), “[The INFORMATION_SCHEMA ROUTINES Table](#)”, for information about this table.

B.4.8: Is it possible to group stored procedures or stored functions into packages?

No. This is not supported in MySQL 5.5.

B.4.9: Can a stored procedure call another stored procedure?

Yes.

B.4.10: Can a stored procedure call a trigger?

A stored procedure can execute an SQL statement, such as an `UPDATE`, that causes a trigger to activate.

B.4.11: Can a stored procedure access tables?

Yes. A stored procedure can access one or more tables as required.

B.4.12: Do stored procedures have a statement for raising application errors?

Yes. MySQL 5.5 implements the SQL standard `SIGNAL` and `RESIGNAL` statements. See [Section 13.6.7](#), “[Condition Handling](#)”.

B.4.13: Do stored procedures provide exception handling?

MySQL implements `HANDLER` definitions according to the SQL standard. See [Section 13.6.7.2](#), “`DECLARE . . . HANDLER Syntax`”, for details.

B.4.14: Can MySQL 5.5 stored routines return result sets?

Stored procedures can, but stored functions cannot. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You need to use the MySQL 4.1 (or above) client/server protocol for this to work. This means that—for instance—in PHP, you need to use the `mysql_i` extension rather than the old `mysql` extension.

B.4.15: Is `WITH RECOMPILE` supported for stored procedures?

Not in MySQL 5.5.

B.4.16: Is there a MySQL equivalent to using `mod_plsql` as a gateway on Apache to talk directly to a stored procedure in the database?

There is no equivalent in MySQL 5.5.

B.4.17: Can I pass an array as input to a stored procedure?

Not in MySQL 5.5.

B.4.18: Can I pass a cursor as an `IN` parameter to a stored procedure?

In MySQL 5.5, cursors are available inside stored procedures only.

B.4.19: Can I return a cursor as an `OUT` parameter from a stored procedure?

In MySQL 5.5, cursors are available inside stored procedures only. However, if you do not open a cursor on a `SELECT`, the result will be sent directly to the client. You can also `SELECT INTO` variables. See [Section 13.2.9, “SELECT Syntax”](#).

B.4.20: Can I print out a variable's value within a stored routine for debugging purposes?

Yes, you can do this in a *stored procedure*, but not in a stored function. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You will need to use the MySQL 4.1 (or above) client/server protocol for this to work. This means that—for instance—in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

B.4.21: Can I commit or roll back transactions inside a stored procedure?

Yes. However, you cannot perform transactional operations within a stored function.

B.4.22: Do MySQL 5.5 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a master MySQL server to a slave server. There are a few limitations that are described in detail in [Section 19.7, “Binary Logging of Stored Programs”](#).

B.4.23: Are stored procedures and functions created on a master server replicated to a slave?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a master server are replicated to a slave, so the objects will exist on both servers. `ALTER` and `DROP` statements for stored procedures and functions are also replicated.

B.4.24: How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a slave server. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

B.4.25: Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a slave server has authority to execute any statement read from a master's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active, then MySQL DBAs have two security options open to them:

1. Any user wishing to create stored functions must be granted the `SUPER` [\[697\]](#) privilege.
2. Alternatively, a DBA can set the `log_bin_trust_function_creators` [\[514\]](#) system variable to 1, which enables anyone with the standard `CREATE ROUTINE` [\[695\]](#) privilege to create stored functions.

B.4.26: What limitations exist for replicating stored procedure and function actions?

Nondeterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced, and therefore, random actions replicated to a slave will not mirror those performed on a master. Note that declaring stored functions to be `DETERMINISTIC` or setting the `log_bin_trust_function_creators` [\[514\]](#) system variable to 0 will not allow random-valued operations to be invoked.

In addition, time-based actions cannot be reproduced on a slave because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, nontransactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a master may be partially updated from DML activity, but no updates are done to the slave because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the `IGNORE` keyword so that updates on the master that cause errors are ignored and updates that do not cause errors are replicated to the slave.

B.4.27: Do the preceding limitations affect MySQL's ability to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

B.4.28: What is being done to correct the aforementioned limitations?

You can choose either statement-based replication or row-based replication. The original replication implementation is based on statement-based binary logging. Row-based binary logging resolves the limitations mentioned earlier.

Mixed replication is also available (by starting the server with `--binlog-format=mixed` [436]). This hybrid, "smart" form of replication "knows" whether statement-level replication can safely be used, or row-level replication is required.

For additional information, see [Section 16.1.2, "Replication Formats"](#).

B.5. MySQL 5.5 FAQ: Triggers

Questions

- [B.5.1: \[3940\]](#) Where can I find the documentation for MySQL 5.5 triggers?
- [B.5.2: \[3940\]](#) Is there a discussion forum for MySQL Triggers?
- [B.5.3: \[3940\]](#) Does MySQL 5.5 have statement-level or row-level triggers?
- [B.5.4: \[3940\]](#) Are there any default triggers?
- [B.5.5: \[3940\]](#) How are triggers managed in MySQL?
- [B.5.6: \[3940\]](#) Is there a way to view all triggers in a given database?
- [B.5.7: \[3940\]](#) Where are triggers stored?
- [B.5.8: \[3940\]](#) Can a trigger call a stored procedure?
- [B.5.9: \[3940\]](#) Can triggers access tables?
- [B.5.10: \[3940\]](#) Can triggers call an external application through a UDF?
- [B.5.11: \[3940\]](#) Is it possible for a trigger to update tables on a remote server?
- [B.5.12: \[3941\]](#) Do triggers work with replication?
- [B.5.13: \[3941\]](#) How are actions carried out through triggers on a master replicated to a slave?

Questions and Answers

B.5.1: Where can I find the documentation for MySQL 5.5 triggers?

See [Section 19.3, “Using Triggers”](#).

B.5.2: Is there a discussion forum for MySQL Triggers?

Yes. It is available at <http://forums.mysql.com/list.php?99>.

B.5.3: Does MySQL 5.5 have statement-level or row-level triggers?

In MySQL 5.5, all triggers are `FOR EACH ROW`—that is, the trigger is activated for each row that is inserted, updated, or deleted. MySQL 5.5 does not support triggers using `FOR EACH STATEMENT`.

B.5.4: Are there any default triggers?

Not explicitly. MySQL does have specific special behavior for some `TIMESTAMP` columns, as well as for columns which are defined using `AUTO_INCREMENT`.

B.5.5: How are triggers managed in MySQL?

In MySQL 5.5, triggers can be created using the `CREATE TRIGGER` statement, and dropped using `DROP TRIGGER`. See [Section 13.1.19, “CREATE TRIGGER Syntax”](#), and [Section 13.1.30, “DROP TRIGGER Syntax”](#), for more about these statements.

Information about triggers can be obtained by querying the `INFORMATION_SCHEMA.TRIGGERS` table. See [Section 20.25, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

B.5.6: Is there a way to view all triggers in a given database?

Yes. You can obtain a listing of all triggers defined on database `dbname` using a query on the `INFORMATION_SCHEMA.TRIGGERS` table such as the one shown here:

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
FROM INFORMATION_SCHEMA.TRIGGERS
WHERE TRIGGER_SCHEMA='dbname';
```

For more information about this table, see [Section 20.25, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

You can also use the `SHOW TRIGGERS` statement, which is specific to MySQL. See [Section 13.7.5.39, “SHOW TRIGGERS Syntax”](#).

B.5.7: Where are triggers stored?

Triggers for a table are currently stored in `.TRG` files, with one such file one per table.

B.5.8: Can a trigger call a stored procedure?

Yes.

B.5.9: Can triggers access tables?

A trigger can access both old and new data in its own table. A trigger can also affect other tables, but it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

B.5.10: Can triggers call an external application through a UDF?

Yes. For example, a trigger could invoke the `sys_exec()` UDF.

B.5.11: Is it possible for a trigger to update tables on a remote server?

Yes. A table on a remote server could be updated using the `FEDERATED` storage engine. (See [Section 14.11, “The `FEDERATED` Storage Engine”](#)).

B.5.12: Do triggers work with replication?

Yes. However, the way in which they work depends whether you are using MySQL's “classic” statement-based replication available in all versions of MySQL, or the row-based replication format introduced in MySQL 5.1.

When using statement-based replication, triggers on the slave are executed by statements that are executed on the master (and replicated to the slave).

When using row-based replication, triggers are not executed on the slave due to statements that were run on the master and then replicated to the slave. Instead, when using row-based replication, the changes caused by executing the trigger on the master are applied on the slave.

For more information, see [Section 16.4.1.32, “Replication and Triggers”](#).

B.5.13: How are actions carried out through triggers on a master replicated to a slave?

Again, this depends on whether you are using statement-based or row-based replication.

Statement-based replication. First, the triggers that exist on a master must be re-created on the slave server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table `EMP` that has an `AFTER` insert trigger, which exists on a master MySQL server. The same `EMP` table and `AFTER` insert trigger exist on the slave server as well. The replication flow would be:

1. An `INSERT` statement is made to `EMP`.
2. The `AFTER` trigger on `EMP` activates.
3. The `INSERT` statement is written to the binary log.
4. The replication slave picks up the `INSERT` statement to `EMP` and executes it.
5. The `AFTER` trigger on `EMP` that exists on the slave activates.

Row-based replication. When you use row-based replication, the changes caused by executing the trigger on the master are applied on the slave. However, the triggers themselves are not actually executed on the slave under row-based replication. This is because, if both the master and the slave applied the changes from the master and—in addition—the trigger causing these changes were applied on the slave, the changes would in effect be applied twice on the slave, leading to different data on the master and the slave.

In most cases, the outcome is the same for both row-based and statement-based replication. However, if you use different triggers on the master and slave, you cannot use row-based replication. (This is because the row-based format replicates the changes made by triggers executing on the master to the slaves, rather than the statements that caused the triggers to execute, and the corresponding triggers on the slave are not executed.) Instead, any statements causing such triggers to be executed must be replicated using statement-based replication.

For more information, see [Section 16.4.1.32, “Replication and Triggers”](#).

B.6. MySQL 5.5 FAQ: Views

Questions

- [B.6.1: \[3942\]](#) Where can I find documentation covering MySQL Views?
- [B.6.2: \[3942\]](#) Is there a discussion forum for MySQL Views?
- [B.6.3: \[3942\]](#) What happens to a view if an underlying table is dropped or renamed?
- [B.6.4: \[3942\]](#) Does MySQL 5.5 have table snapshots?
- [B.6.5: \[3942\]](#) Does MySQL 5.5 have materialized views?
- [B.6.6: \[3942\]](#) Can you insert into views that are based on joins?

Questions and Answers

B.6.1: Where can I find documentation covering MySQL Views?

See [Section 19.5, "Using Views"](#).

B.6.2: Is there a discussion forum for MySQL Views?

Yes. See <http://forums.mysql.com/list.php?100>

B.6.3: What happens to a view if an underlying table is dropped or renamed?

After a view has been created, it is possible to drop or alter a table or view to which the definition refers. To check a view definition for problems of this kind, use the `CHECK TABLE` statement. (See [Section 13.7.2.2, "CHECK TABLE Syntax"](#).)

B.6.4: Does MySQL 5.5 have table snapshots?

No.

B.6.5: Does MySQL 5.5 have materialized views?

No.

B.6.6: Can you insert into views that are based on joins?

It is possible, provided that your `INSERT` statement has a column list that makes it clear there is only one table involved.

You *cannot* insert into multiple tables with a single insert on a view.

B.7. MySQL 5.5 FAQ: [INFORMATION_SCHEMA](#)

Questions

- [B.7.1: \[3943\]](#) Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?
- [B.7.2: \[3943\]](#) Is there a discussion forum for `INFORMATION_SCHEMA`?
- [B.7.3: \[3943\]](#) Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?
- [B.7.4: \[3943\]](#) What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?
- [B.7.5: \[3943\]](#) Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

Questions and Answers

B.7.1: Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?

See [Chapter 20, `INFORMATION_SCHEMA` Tables](#)

B.7.2: Is there a discussion forum for `INFORMATION_SCHEMA`?

See <http://forums.mysql.com/list.php?101>.

B.7.3: Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available—such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer—which give a comprehensive overview of the standard, including `INFORMATION_SCHEMA`.

B.7.4: What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. MySQL's implementation is more similar to those found in DB2 and SQL Server, which also support `INFORMATION_SCHEMA` as defined in the SQL standard.

B.7.5: Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, *we cannot support bugs or other issues which result from modifying `INFORMATION_SCHEMA` tables or data.*

B.8. MySQL 5.5 FAQ: Migration

Questions

- **B.8.1:** [\[3943\]](#) Where can I find information on how to migrate from MySQL 5.1 to MySQL 5.5?
- **B.8.2:** [\[3943\]](#) How has storage engine (table type) support changed in MySQL 5.5 from previous versions?

Questions and Answers

B.8.1: Where can I find information on how to migrate from MySQL 5.1 to MySQL 5.5?

For detailed upgrade information, see [Section 2.12.1, “Upgrading MySQL”](#). Do not skip a major version when upgrading, but rather complete the process in steps, upgrading from one major version to the next in each step. This may seem more complicated, but it will you save time and trouble—if you encounter problems during the upgrade, their origin will be easier to identify, either by you or—if you have a MySQL Enterprise subscription—by MySQL support.

B.8.2: How has storage engine (table type) support changed in MySQL 5.5 from previous versions?

Storage engine support has changed as follows:

- Support for `ISAM` tables was removed in MySQL 5.0 and you should now use the `MyISAM` storage engine in place of `ISAM`. To convert a table `tblname` from `ISAM` to `MyISAM`, simply issue a statement such as this one:

```
ALTER TABLE tblname ENGINE=MYISAM;
```

- Internal `RAID` for `MyISAM` tables was also removed in MySQL 5.0. This was formerly used to allow large tables in file systems that did not support file sizes greater than 2GB. All modern file systems allow for larger tables; in addition, there are now other solutions such as `MERGE` tables and views.

- The `VARCHAR` column type now retains trailing spaces in all storage engines.
- `MEMORY` tables (formerly known as `HEAP` tables) can also contain `VARCHAR` columns.

B.9. MySQL 5.5 FAQ: Security

Questions

- **B.9.1:** [3944] Where can I find documentation that addresses security issues for MySQL?
- **B.9.2:** [3944] Does MySQL 5.5 have native support for SSL?
- **B.9.3:** [3944] Is SSL support be built into MySQL binaries, or must I recompile the binary myself to enable it?
- **B.9.4:** [3945] Does MySQL 5.5 have built-in authentication against LDAP directories?
- **B.9.5:** [3945] Does MySQL 5.5 include support for Roles Based Access Control (RBAC)?

Questions and Answers

B.9.1: Where can I find documentation that addresses security issues for MySQL?

The best place to start is [Chapter 6, Security](#).

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

- [Section 6.1.1, “Security Guidelines”](#).
- [Section 6.1.3, “Making MySQL Secure Against Attackers”](#).
- [Section C.5.4.1, “How to Reset the Root Password”](#).
- [Section 6.1.5, “How to Run MySQL as a Normal User”](#).
- [Section 23.3.2.6, “User-Defined Function Security Precautions”](#).
- [Section 6.1.4, “Security-Related `mysqld` Options and Variables”](#).
- [Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”](#).
- [Section 2.11, “Postinstallation Setup and Testing”](#).
- [Section 6.3.8.1, “Basic SSL Concepts”](#).

B.9.2: Does MySQL 5.5 have native support for SSL?

Most 5.5 binaries have support for SSL connections between the client and server. See [Section 6.3.8, “Using SSL for Secure Connections”](#).

You can also tunnel a connection using SSH, if (for example) the client application does not support SSL connections. For an example, see [Section 6.3.9, “Connecting to MySQL Remotely from Windows with SSH”](#).

B.9.3: Is SSL support be built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 5.5 binaries have SSL enabled for client-server connections that are secured, authenticated, or both. See [Section 6.3.8, “Using SSL for Secure Connections”](#).

B.9.4: Does MySQL 5.5 have built-in authentication against LDAP directories?

Not at this time.

B.9.5: Does MySQL 5.5 include support for Roles Based Access Control (RBAC)?

Not at this time.

B.10. MySQL FAQ: MySQL 5.5 and MySQL Cluster

In the following section, we answer questions that are frequently asked about MySQL Cluster and the [NDBCLUSTER](#) storage engine.

Questions

- [B.10.1: \[3946\]](#) Which versions of the MySQL software support Cluster? Do I have to compile from source?
- [B.10.2: \[3947\]](#) What do “NDB” and “NDBCLUSTER” mean?
- [B.10.3: \[3947\]](#) What is the difference between using MySQL Cluster vs using MySQL Replication?
- [B.10.4: \[3947\]](#) Do I need any special networking to run MySQL Cluster? How do computers in a cluster communicate?
- [B.10.5: \[3948\]](#) How many computers do I need to run a MySQL Cluster, and why?
- [B.10.6: \[3948\]](#) What do the different computers do in a MySQL Cluster?
- [B.10.7: \[3948\]](#) When I run the `SHOW` command in the MySQL Cluster management client, I see a line of output that looks like this:

```
id=2 @10.100.10.32 (Version: 5.5.31, Nodegroup: 0, Master)
```

What is a “master node”, and what does it do? How do I configure a node so that it is the master?

- [B.10.8: \[3949\]](#) With which operating systems can I use MySQL Cluster?
- [B.10.9: \[3949\]](#) What are the hardware requirements for running MySQL Cluster?
- [B.10.10: \[3949\]](#) How much RAM do I need to use MySQL Cluster? Is it possible to use disk memory at all?
- [B.10.11: \[3950\]](#) What file systems can I use with MySQL Cluster? What about network file systems or network shares?
- [B.10.12: \[3950\]](#) Can I run MySQL Cluster nodes inside virtual machines (such as those created by VMWare, Parallels, or Xen)?
- [B.10.13: \[3951\]](#) I am trying to populate a MySQL Cluster database. The loading process terminates prematurely and I get an error message like this one: `ERROR 1114: The table 'my_cluster_table' is full` Why is this happening?
- [B.10.14: \[3951\]](#) MySQL Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

- [B.10.15: \[3951\]](#) Do I have to learn a new programming or query language to use MySQL Cluster?
- [B.10.16: \[3952\]](#) How do I find out what an error or warning message means when using MySQL Cluster?
- [B.10.17: \[3952\]](#) Is MySQL Cluster transaction-safe? What isolation levels are supported?
- [B.10.18: \[3952\]](#) What storage engines are supported by MySQL Cluster?
- [B.10.19: \[3952\]](#) In the event of a catastrophic failure—say, for instance, the whole city loses power *and* my UPS fails—would I lose all my data?
- [B.10.20: \[3952\]](#) Is it possible to use `FULLTEXT` indexes with MySQL Cluster?
- [B.10.21: \[3952\]](#) Can I run multiple nodes on a single computer?
- [B.10.22: \[3953\]](#) Are there any limitations that I should be aware of when using MySQL Cluster?
- [B.10.23: \[3953\]](#) How do I import an existing MySQL database into a MySQL Cluster?
- [B.10.24: \[3954\]](#) How do MySQL Cluster nodes communicate with one another?
- [B.10.25: \[3954\]](#) What is an *arbitrator*?
- [B.10.26: \[3954\]](#) What data types are supported by MySQL Cluster?
- [B.10.27: \[3954\]](#) How do I start and stop MySQL Cluster?
- [B.10.28: \[3955\]](#) What happens to MySQL Cluster data when the MySQL Cluster is shut down?
- [B.10.29: \[3955\]](#) Is it a good idea to have more than one management node for a MySQL Cluster?
- [B.10.30: \[3955\]](#) Can I mix different kinds of hardware and operating systems in one MySQL Cluster?
- [B.10.31: \[3956\]](#) Can I run two data nodes on a single host? Two SQL nodes?
- [B.10.32: \[3956\]](#) Can I use host names with MySQL Cluster?
- [B.10.33: \[3956\]](#) Does MySQL Cluster support IPv6?
- [B.10.34: \[3956\]](#) How do I handle MySQL users in a MySQL Cluster having multiple MySQL servers?
- [B.10.35: \[3956\]](#) How do I continue to send queries in the event that one of the SQL nodes fails?
- [B.10.36: \[3956\]](#) How do I back up and restore a MySQL Cluster?
- [B.10.37: \[3956\]](#) What is an “angel process”?

Questions and Answers

B.10.1: Which versions of the MySQL software support Cluster? Do I have to compile from source?

MySQL Cluster is not supported in standard MySQL Server 5.5 releases. Instead, MySQL Cluster is provided as a separate product. Currently, the following MySQL Cluster release series are available for production use:

- **MySQL Cluster NDB 6.3.** This series is still maintained and available for use but is not recommended for new deployments. The most recent MySQL Cluster NDB 6.3 release can be obtained from <http://dev.mysql.com/downloads/cluster/>.

- **MySQL Cluster NDB 7.0.** This series is still available for use in production, although MySQL Cluster NDB 7.2 is recommended for new deployments. The most recent MySQL Cluster NDB 7.0 release can be obtained from <http://dev.mysql.com/downloads/cluster/>.
- **MySQL Cluster NDB 7.1.** This series is the previous Generally Available (GA) version of MySQL Cluster, still available for production, although we recommend that new deployments use the latest MySQL Cluster NDB 7.2 release. The most recent MySQL Cluster NDB 7.1 release can be obtained from <http://dev.mysql.com/downloads/cluster/>.

MySQL Cluster NDB 7.2. This series is the latest Generally Available (GA) version of MySQL Cluster, based on version 7.2 of the `NDBCLUSTER` storage engine and MySQL Server 5.5. New deployments should use the latest release in this series. The most recent MySQL Cluster NDB 7.2 release can be obtained from <http://dev.mysql.com/downloads/cluster/>.

You should use MySQL NDB Cluster NDB 7.1 or MySQL NDB Cluster NDB 7.2 for any new deployments; if you are using an older version of MySQL Cluster, you should upgrade to one of these soon as possible. For an overview of improvements made in MySQL Cluster NDB 7.1, see [MySQL Cluster Development in MySQL Cluster NDB 7.1](#); for information about improvements made in MySQL Cluster NDB 7.2, see [Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”](#).

You can determine whether your MySQL Server has `NDBCLUSTER` support using one of the statements `SHOW VARIABLES LIKE 'have_%%'`, `SHOW ENGINES`, or `SHOW PLUGINS`.

B.10.2: What do “NDB” and “NDBCLUSTER” mean?

“NDB” stands for “**N**etwork **D**atabase”. `NDB` and `NDBCLUSTER` are both names for the storage engine that enables clustering support in MySQL. While our developers prefer `NDB`, either name is correct; both names appear in our documentation, and either name can be used in the `ENGINE` option of a `CREATE TABLE` statement for creating a MySQL Cluster table.

B.10.3: What is the difference between using MySQL Cluster vs using MySQL Replication?

In traditional MySQL replication, a master MySQL server updates one or more slaves. Transactions are committed sequentially, and a slow transaction can cause the slave to lag behind the master. This means that if the master fails, it is possible that the slave might not have recorded the last few transactions. If a transaction-safe engine such as `InnoDB` is being used, a transaction will either be complete on the slave or not applied at all, but replication does not guarantee that all data on the master and the slave will be consistent at all times. In MySQL Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is *asynchronous*, MySQL Cluster is *synchronous*.

Asynchronous replication is also available in MySQL Cluster. *MySQL Cluster Replication* (also sometimes known as “geo-replication”) includes the capability to replicate both between two MySQL Clusters, and from a MySQL Cluster to a non-Cluster MySQL server. See [Section 17.6, “MySQL Cluster Replication”](#).

B.10.4: Do I need any special networking to run MySQL Cluster? How do computers in a cluster communicate?

MySQL Cluster is intended to be used in a high-bandwidth environment, with computers connecting using TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for MySQL Cluster include a typical 100-megabit Ethernet network or the equivalent. We recommend you use gigabit Ethernet whenever available.

The faster SCI protocol is also supported, but requires special hardware. See [Section 17.3.5, “Using High-Speed Interconnects with MySQL Cluster”](#), for more information about SCI.

B.10.5: How many computers do I need to run a MySQL Cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum *recommended* number of computers in a MySQL Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

B.10.6: What do the different computers do in a MySQL Cluster?

A MySQL Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as *nodes*, and a computer housing a cluster node is sometimes referred to as a *cluster host*. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- **Management node.** This node provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application `ndb_mgmd`; the management client used to control MySQL Cluster is `ndb_mgm`. See [Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”](#), and [Section 17.4.5, “ndb_mgm — The MySQL Cluster Management Client”](#), for information about these programs.
- **Data node.** This type of node stores and replicates data. Data node functionality is handled by instances of the NDB data node process `ndbd`. For more information, see [Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”](#).
- **SQL node.** This is simply an instance of MySQL Server (`mysqld`) that is built with support for the `NDBCLUSTER` storage engine and started with the `--ndb-cluster` option to enable the engine and the `--ndb-connectstring` option to enable it to connect to a MySQL Cluster management server. For more about these options, see [Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”](#).

**Note**

An *API node* is any application that makes direct use of Cluster data nodes for data storage and retrieval. An SQL node can thus be considered a type of API node that uses a MySQL Server to provide an SQL interface to the Cluster. You can write such applications (that do not depend on a MySQL Server) using the NDB API, which supplies a direct, object-oriented transaction and scanning interface to MySQL Cluster data; see [MySQL Cluster API Overview: The NDB API](#), for more information.

B.10.7: When I run the `SHOW` command in the MySQL Cluster management client, I see a line of output that looks like this:

```
id=2      @10.100.10.32 (Version: 5.5.31, Nodegroup: 0, Master)
```

What is a “master node”, and what does it do? How do I configure a node so that it is the master?

The simplest answer is, “It’s not something you can control, and it’s nothing that you need to worry about in any case, unless you’re a software engineer writing or analyzing the MySQL Cluster source code”.

If you don’t find that answer satisfactory, here’s a longer and more technical version:

A number of mechanisms in MySQL Cluster require distributed coordination among the data nodes. These distributed algorithms and protocols include global checkpointing, DDL (schema) changes, and node restart handling. To make this coordination simpler, the data nodes “elect” one of their number to be a “master”. There is no user-facing mechanism for influencing this selection, which is completely automatic; the fact that it *is* automatic is a key part of MySQL Cluster's internal architecture.

When a node acts as a master for any of these mechanisms, it is usually the point of coordination for the activity, and the other nodes act as “servants”, carrying out their parts of the activity as directed by the master. If the node acting as master fails, then the remaining nodes elect a new master. Tasks in progress that were being coordinated by the old master may either fail or be continued by the new master, depending on the actual mechanism involved.

It is possible for some of these different mechanisms and protocols to have different master nodes, but in general the same master is chosen for all of them. The node indicated as the master in the output of `SHOW` in the management client is actually the `DICT` master (see [The `DBDICT` Block](#), in the *MySQL Cluster API Developer Guide*, for more information), responsible for coordinating DDL and metadata activity.

MySQL Cluster is designed in such a way that the choice of master has no discernable effect outside the cluster itself. For example, the current master does not have significantly higher CPU or resource usage than the other data nodes, and failure of the master should not have a significantly different impact on the cluster than the failure of any other data node.

B.10.8: With which operating systems can I use MySQL Cluster?

MySQL Cluster is supported on most Unix-like operating systems. MySQL Cluster NDB 7.2 is also supported in production settings on Microsoft Windows operating systems.

For more detailed information concerning the level of support which is offered for MySQL Cluster on various operating system versions, operating system distributions, and hardware platforms, please refer to <http://www.mysql.com/support/supportedplatforms/cluster.html>.

B.10.9: What are the hardware requirements for running MySQL Cluster?

MySQL Cluster should run on any platform for which `NDB`-enabled binaries are available. For data nodes and API nodes, faster CPUs and more memory are likely to improve performance, and 64-bit CPUs are likely to be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see *How much RAM do I Need?* for more information). For a computer which is used only for running the MySQL Cluster management server, the requirements are minimal; a common desktop PC (or the equivalent) is generally sufficient for this task. Nodes can communicate through the standard TCP/IP network and hardware. They can also use the high-speed SCI protocol; however, special networking hardware and software are required to use SCI (see [Section 17.3.5, “Using High-Speed Interconnects with MySQL Cluster”](#)).

B.10.10: How much RAM do I need to use MySQL Cluster? Is it possible to use disk memory at all?

Formerly MySQL Cluster was in-memory only. MySQL 5.1 and later also provide the ability to store MySQL Cluster on disk. (Note that we have no plans to backport this capability to previous releases.) See [Section 17.5.12, “MySQL Cluster Disk Data Tables”](#), for more information.

For in-memory `NDB` tables, you can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

```
(SizeofDatabase × NumberOfReplicas × 1.1 ) / NumberOfDataNodes
```

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see [Section 11.6, “Data Type Storage Requirements”](#), for

details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an `NDBCLUSTER` table requires 21–25 bytes per record. These indexes use `IndexMemory`.
- Each ordered index requires 10 bytes storage per record, using `DataMemory`.
- Creating a primary key or unique index also creates an ordered index, unless this index is created with `USING HASH`. In other words:
 - A primary key or unique index on a Cluster table normally takes up 31 to 35 bytes per record.
 - However, if the primary key or unique index is created with `USING HASH`, then it requires only 21 to 25 bytes per record.

Note that creating MySQL Cluster tables with `USING HASH` for all primary keys and unique indexes will generally cause table updates to run more quickly—in some cases by a much as 20 to 30 percent faster than updates on tables where `USING HASH` was not used in creating primary and unique keys. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated). However, it also means that queries that could otherwise use range scans must be satisfied by other means, which can result in slower selects.

When calculating Cluster memory requirements, you may find useful the `ndb_size.pl` utility which is available in recent MySQL 5.5 releases. This Perl script connects to a current (non-Cluster) MySQL database and creates a report on how much space that database would require if it used the `NDBCLUSTER` storage engine. For more information, see [Section 17.4.22, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”](#).

It is especially important to keep in mind that *every MySQL Cluster table must have a primary key*. The `NDB` storage engine creates a primary key automatically if none is defined; this primary key is created without `USING HASH`.

There is no easy way to determine exactly how much memory is being used for storage of Cluster indexes at any given time; however, warnings are written to the Cluster log when 80% of available `DataMemory` or `IndexMemory` is in use, and again when use reaches 85%, 90%, and so on.

B.10.11: What file systems can I use with MySQL Cluster? What about network file systems or network shares?

Generally, any file system that is native to the host operating system should work well with MySQL Cluster. If you find that a given file system works particularly well (or not so especially well) with MySQL Cluster, we invite you to discuss your findings in the [MySQL Cluster Forums](#).

For Windows, we recommend that you use `NTFS` file systems for MySQL Cluster, just as we do for standard MySQL. We do not test MySQL Cluster with `FAT` or `VFAT` file systems. Because of this, we do not recommend their use with MySQL or MySQL Cluster.

MySQL Cluster is implemented as a shared-nothing solution; the idea behind this is that the failure of a single piece of hardware should not cause the failure of multiple cluster nodes, or possibly even the failure of the cluster as a whole. For this reason, the use of network shares or network file systems is not supported for MySQL Cluster. This also applies to shared storage devices such as SANs.

B.10.12: Can I run MySQL Cluster nodes inside virtual machines (such as those created by VMWare, Parallels, or Xen)?

MySQL Cluster is supported for use in virtual machines beginning with MySQL Cluster NDB 7.2. We currently support and test using [Oracle VM](#).

Some MySQL Cluster users have successfully deployed MySQL Cluster using other virtualization products; in such cases, Oracle can provide MySQL Cluster support, but issues specific to the virtual environment must be referred to that product's vendor.

B.10.13: I am trying to populate a MySQL Cluster database. The loading process terminates prematurely and I get an error message like this one: `ERROR 1114: The table 'my_cluster_table' is full` Why is this happening?

The cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, *including the primary key required by the NDB storage engine and automatically created in the event that the table definition does not include the definition of a primary key.*

It is also worth noting that all data nodes should have the same amount of RAM, since no data node in a cluster can use more memory than the least amount available to any individual data node. For example, if there are four computers hosting Cluster data nodes, and three of these have 3GB of RAM available to store Cluster data while the remaining data node has only 1GB RAM, then each data node can devote at most 1GB to MySQL Cluster data and indexes.

In some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free `DataMemory` [2043]. You can force NDB to create extra partitions for MySQL Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. In general, setting `MAX_ROWS` to twice the number of rows that you expect to store in the table should be sufficient.

For similar reasons, you can also sometimes encounter problems with data node restarts on nodes that are heavily loaded with data. In MySQL Cluster NDB 7.1 and MySQL Cluster NDB 7.2, the addition of the `MinFreePct` [2045] parameter helps with this issue by reserving a portion (5% by default) of `DataMemory` [2043] and `IndexMemory` [2044] for use in restarts. This reserved memory is not available for storing NDB tables or data.

B.10.14: MySQL Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

It is very unlikely that a cluster would perform reliably under such conditions, as MySQL Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet—preferably the latter. We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in a MySQL Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you should take the same precautions as you would with any other instance of the MySQL server.) For more information, see [Section 17.5.11, "MySQL Cluster Security Issues"](#).

B.10.15: Do I have to learn a new programming or query language to use MySQL Cluster?

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL statements are required for the following operations:

- Creating, altering, and dropping tables
- Inserting, updating, and deleting table data

- Creating, changing, and dropping primary and unique indexes

Some specialized configuration parameters and files are required to set up a MySQL Cluster—see [Section 17.3.2, “MySQL Cluster Configuration Files”](#), for information about these.

A few simple commands are used in the MySQL Cluster management client (`ndb_mgm`) for tasks such as starting and stopping cluster nodes. See [Section 17.5.2, “Commands in the MySQL Cluster Management Client”](#).

B.10.16: How do I find out what an error or warning message means when using MySQL Cluster?

There are two ways in which this can be done:

- From within the `mysql` client, use `SHOW ERRORS` or `SHOW WARNINGS` immediately upon being notified of the error or warning condition.
- From a system shell prompt, use `pererror --ndb error_code`.

B.10.17: Is MySQL Cluster transaction-safe? What isolation levels are supported?

Yes. For tables created with the `NDB` storage engine, transactions are supported. Currently, MySQL Cluster supports only the `READ COMMITTED [1402]` transaction isolation level.

B.10.18: What storage engines are supported by MySQL Cluster?

Clustering with MySQL is supported only by the `NDB` storage engine. That is, in order for a table to be shared between nodes in a MySQL Cluster, the table must be created using `ENGINE=NDB` (or the equivalent option `ENGINE=NDBCLUSTER`).

It is possible to create tables using other storage engines (such as `MyISAM` or `InnoDB`) on a MySQL server being used with a MySQL Cluster, but these non-`NDB` tables do *not* participate in clustering; each such table is strictly local to the individual MySQL server instance on which it is created.

B.10.19: In the event of a catastrophic failure—say, for instance, the whole city loses power and my UPS fails—would I lose all my data?

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

B.10.20: Is it possible to use `FULLTEXT` indexes with MySQL Cluster?

`FULLTEXT` indexing is currently not supported by any MySQL storage engine other than `MyISAM`.

B.10.21: Can I run multiple nodes on a single computer?

It is possible but not always advisable. One of the chief reasons to run a cluster is to provide redundancy. To obtain the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. For this reason, if you do run multiple data nodes on a single machine, it is *extremely* important that they be set up in such a way that the failure of this machine does not cause the loss of all the data nodes in a given node group.

Given that MySQL Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 300 MHz Pentium or equivalent CPU and sufficient RAM for the operating system, plus a small amount of overhead for the `ndb_mgmd` and `ndb_mgm` processes.

It is acceptable to run multiple cluster data nodes on a single host that has multiple CPUs, cores, or both. MySQL Cluster NDB 7.0 and later also provide a multi-threaded version of the data node binary intended for use on such systems. For more information, see [Section 17.4.3, “ndbmt.d — The MySQL Cluster Data Node Daemon \(Multi-Threaded\)”](#).

It is also possible in some cases to run data nodes and SQL nodes concurrently on the same machine; how well such an arrangement performs is dependent on a number of factors such as number of cores and CPUs as well as the amount of disk and memory available to the data node and SQL node processes, and you must take these factors into account when planning such a configuration.

B.10.22: Are there any limitations that I should be aware of when using MySQL Cluster?

Limitations on [NDB](#) tables in MySQL MySQL Cluster NDB 7.2 include the following:

- Temporary tables are not supported; a `CREATE TEMPORARY TABLE` statement using `ENGINE=NDB` or `ENGINE=NDBCLUSTER` fails with an error.
- The only types of user-defined partitioning supported for [NDBCLUSTER](#) tables are `KEY` and `LINEAR KEY`. Trying to create an [NDB](#) table using any other partitioning type fails with an error.
- `FULLTEXT` indexes are not supported.
- Index prefixes are not supported. Only complete columns may be indexed.
- Spatial indexes are not supported (although spatial columns can be used). See [Section 12.17, “Spatial Extensions”](#).
- Support for partial transactions and partial rollbacks is comparable to that of other transactional storage engines such as [InnoDB](#) that can roll back individual statements.
- The maximum number of attributes allowed per table is 512. Attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- The maximum size for a table row is 14 kilobytes, not counting `BLOB` values.

There is no set limit for the number of rows per [NDB](#) table. Limits on table size depend on a number of factors, in particular on the amount of RAM available to each data node.

- The [NDBCLUSTER](#) engine does not support foreign key constraints. As with [MyISAM](#) tables, if these are specified in a `CREATE TABLE` or `ALTER TABLE` statement, they are ignored.

For a complete listing of limitations in MySQL Cluster, see [Section 17.1.6, “Known Limitations of MySQL Cluster”](#). See also [Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”](#).

B.10.23: How do I import an existing MySQL database into a MySQL Cluster?

You can import databases into MySQL Cluster much as you would with any other version of MySQL. Other than the limitations mentioned elsewhere in this FAQ, the only other special requirement is that any tables to be included in the cluster must use the [NDB](#) storage engine. This means that the tables must be created with `ENGINE=NDB` or `ENGINE=NDBCLUSTER`.

It is also possible to convert existing tables that use other storage engines to [NDBCLUSTER](#) using one or more `ALTER TABLE` statement. However, the definition of the table must be compatible with the [NDBCLUSTER](#) storage engine prior to making the conversion. In MySQL 5.5, an additional workaround is also required; see [Section 17.1.6, “Known Limitations of MySQL Cluster”](#), for details.

B.10.24: How do MySQL Cluster nodes communicate with one another?

Cluster nodes can communicate through any of three different transport mechanisms: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host; however, this is considered experimental. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See [Section 17.3.5, “Using High-Speed Interconnects with MySQL Cluster”](#), for more about using SCI as a transport mechanism for MySQL Cluster.

B.10.25: What is an *arbiter*?

If one or more data nodes in a cluster fail, it is possible that not all cluster data nodes will be able to “see” one another. In fact, it is possible that two sets of data nodes might become isolated from one another in a network partitioning, also known as a “split-brain” scenario. This type of situation is undesirable because each set of data nodes tries to behave as though it is the entire cluster. An arbiter is required to decide between the competing sets of data nodes.

When all data nodes in at least one node group are alive, network partitioning is not an issue, because no single subset of the cluster can form a functional cluster on its own. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the “split-brain” scenario) becomes possible. Then an arbiter is required. All cluster nodes recognize the same node as the arbiter, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbiter instead. The arbiter accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbiter selection is controlled by the [ArbitrationRank](#) configuration parameter for MySQL Server and management server nodes. In MySQL Cluster NDB 7.0.7 and later, you can also use the [ArbitrationRank](#) configuration parameter to control the arbiter selection process. For more information about these parameters, see [Section 17.3.2.5, “Defining a MySQL Cluster Management Server”](#).

The role of arbiter does not in and of itself impose any heavy demands upon the host so designated, and thus the arbiter host does not need to be particularly fast or to have extra memory especially for this purpose.

B.10.26: What data types are supported by MySQL Cluster?

MySQL Cluster supports all of the usual MySQL data types, including those associated with MySQL's spatial extensions; however, the NDB storage engine does not support spatial indexes. (Spatial indexes are supported only by [MyISAM](#); see [Section 12.17, “Spatial Extensions”](#), for more information.) In addition, there are some differences with regard to indexes when used with NDB tables.

**Note**

MySQL Cluster Disk Data tables (that is, tables created with `TABLESPACE ... STORAGE DISK ENGINE=NDB` or `TABLESPACE ... STORAGE DISK ENGINE=NDBCLUSTER`) have only fixed-width rows. This means that (for example) each Disk Data table record containing a `VARCHAR(255)` column requires space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein.

See [Section 17.1.6, “Known Limitations of MySQL Cluster”](#), for more information about these issues.

B.10.27: How do I start and stop MySQL Cluster?

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node, using the `ndb_mgmd` command.

You must include the `-f [2188]` or `--config-file [2188]` option to tell the management node where its configuration file can be found.

2. Start each data node with the `ndbd` command.

Each data node must be started with the `-c [2241]` or `--ndb-connectstring [2241]` option so that the data node knows how to connect to the management server.

3. Start each MySQL Server (SQL node) using your preferred startup script, such as `mysqld_safe`.

Each MySQL Server must be started with the `--ndbcluster [2142]` and `--ndb-connectstring [2141]` options. These options cause `mysqld` to enable `NDBCLUSTER` storage engine support and how to connect to the management server.

Each of these commands must be run from a system shell on the machine housing the affected node. (You do not have to be physically present at the machine—a remote login shell can be used for this purpose.) You can verify that the cluster is running by starting the `NDB` management client `ndb_mgm` on the machine housing the management node and issuing the `SHOW` or `ALL STATUS` command.

To shut down a running cluster, issue the command `SHUTDOWN` in the management client. Alternatively, you may enter the following command in a system shell:

```
shell> ndb_mgm -e "SHUTDOWN"
```

(The quotation marks in this example are optional, since there are no spaces in the command string following the `-e` option; in addition, the `SHUTDOWN` command, like other management client commands, is not case-sensitive.)

Either of these commands causes the `ndb_mgm`, `ndb_mgm`, and any `ndbd` processes to terminate gracefully. MySQL servers running as SQL nodes can be stopped using `mysqladmin shutdown`.

For more information, see [Section 17.5.2, “Commands in the MySQL Cluster Management Client”](#), and [Section 17.2.6, “Safe Shutdown and Restart of MySQL Cluster”](#).

B.10.28: What happens to MySQL Cluster data when the MySQL Cluster is shut down?

The data that was held in memory by the cluster's data nodes is written to disk, and is reloaded into memory the next time that the cluster is started.

B.10.29: Is it a good idea to have more than one management node for a MySQL Cluster?

It can be helpful as a fail-safe. Only one management node controls the cluster at any given time, but it is possible to configure one management node as primary, and one or more additional management nodes to take over in the event that the primary management node fails.

See [Section 17.3.2, “MySQL Cluster Configuration Files”](#), for information on how to configure MySQL Cluster management nodes.

B.10.30: Can I mix different kinds of hardware and operating systems in one MySQL Cluster?

Yes, as long as all machines and operating systems have the same “endianness” (all big-endian or all little-endian). We are working to overcome this limitation in a future MySQL Cluster release.

It is also possible to use software from different MySQL Cluster releases on different nodes. However, we support this only as part of a rolling upgrade procedure (see [Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”](#)).

B.10.31: Can I run two data nodes on a single host? Two SQL nodes?

Yes, it is possible to do this. In the case of multiple data nodes, it is advisable (but not required) for each node to use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of `mysqld` must use a different TCP/IP port. However, in MySQL 5.5, running more than one cluster node of a given type per machine is generally not encouraged or supported for production use.

We also advise against running data nodes and SQL nodes together on the same host, since the `ndbd` and `mysqld` processes may compete for memory.

B.10.32: Can I use host names with MySQL Cluster?

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your application requires “five nines” availability, you should use fixed (numeric) IP addresses, since making communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional potential points of failure.

B.10.33: Does MySQL Cluster support IPv6?

IPv6 is supported for connections between SQL nodes (MySQL servers), but connections between all other types of MySQL Cluster nodes must use IPv4.

In practical terms, this means that you can use IPv6 for replication between MySQL Clusters, but connections between nodes in the same MySQL Cluster must use IPv4. For more information, see [Section 17.6.3, “Known Issues in MySQL Cluster Replication”](#).

B.10.34: How do I handle MySQL users in a MySQL Cluster having multiple MySQL servers?

MySQL user accounts and privileges are normally not automatically propagated between different MySQL servers accessing the same MySQL Cluster. Beginning with MySQL Cluster NDB 7.2, MySQL Cluster provides support for distributed privileges. While privilege distribution is not enabled automatically, you can activate it by following a procedure provided in the MySQL Cluster documentation. See [Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”](#), for more information.

B.10.35: How do I continue to send queries in the event that one of the SQL nodes fails?

MySQL Cluster does not provide any sort of automatic failover between SQL nodes. Your application must be prepared to handle the loss of SQL nodes and to fail over between them.

B.10.36: How do I back up and restore a MySQL Cluster?

You can use the NDB native backup and restore functionality in the MySQL Cluster management client and the `ndb_restore` program. See [Section 17.5.3, “Online Backup of MySQL Cluster”](#), and [Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”](#).

You can also use the traditional functionality provided for this purpose in `mysqldump` and the MySQL server. See [Section 4.5.4, “mysqldump — A Database Backup Program”](#), for more information.

B.10.37: What is an “angel process”?

This process monitors and, if necessary, attempts to restart the data node process. If you check the list of active processes on your system after starting `ndbd`, you can see that there are actually 2 processes running by that name, as shown here (we omit the output from `ndb_mgmd` and `ndbd` for brevity):

```
shell> ./ndb_mgmd

shell> ps aux | grep ndb
me      23002  0.0  0.0 122948  3104 ?        Ssl  14:14   0:00 ./ndb_mgmd
```

```

me      23025  0.0  0.0  5284   820 pts/2    S+   14:14   0:00 grep ndb

shell> ./ndbd -c 127.0.0.1 --initial

shell> ps aux | grep ndb
me      23002  0.0  0.0 123080  3356 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23096  0.0  0.0  35876  2036 ?        Ss   14:14   0:00 ./ndbd -c 127.0.0.1 --initial
me      23097  1.0  2.4 524116 91096 ?        Sl   14:14   0:00 ./ndbd -c 127.0.0.1 --initial
me      23168  0.0  0.0  5284   812 pts/2    R+   14:15   0:00 grep ndb

```

The `ndbd` process showing 0 memory and CPU usage is the angel process. It actually does use a very small amount of each, of course. It simply checks to see if the main `ndbd` process (the primary data node process that actually handles the data) is running. If permitted to do so (for example, if the `StopOnError` configuration parameter is set to `false`—see [Section 17.3.3.1, “MySQL Cluster Data Node Configuration Parameters”](#)), the angel process tries to restart the primary data node process.

B.11. MySQL 5.5 FAQ: MySQL Chinese, Japanese, and Korean Character Sets

This set of Frequently Asked Questions derives from the experience of MySQL's Support and Development groups in handling many inquiries about CJK (Chinese-Japanese-Korean) issues.

Questions

- [B.11.1: \[3958\]](#) What CJK character sets are available in MySQL?
- [B.11.2: \[3959\]](#) I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters?
- [B.11.3: \[3960\]](#) What problems should I be aware of when working with the Big5 Chinese character set?
- [B.11.4: \[3961\]](#) Why do Japanese character set conversions fail?
- [B.11.5: \[3962\]](#) What should I do if I want to convert SJIS `81CA` to `cp932`?
- [B.11.6: \[3962\]](#) How does MySQL represent the Yen (¥) sign?
- [B.11.7: \[3962\]](#) Does MySQL plan to make a separate character set where `5C` is the Yen sign, as at least one other major DBMS does?
- [B.11.8: \[3962\]](#) Of what issues should I be aware when working with Korean character sets in MySQL?
- [B.11.9: \[3962\]](#) Why do I get `Incorrect string value` error messages?
- [B.11.10: \[3963\]](#) Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or another API?
- [B.11.11: \[3964\]](#) I've upgraded to MySQL 5.5. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?
- [B.11.12: \[3965\]](#) Why do some `LIKE [1111]` and `FULLTEXT` searches with CJK characters fail?
- [B.11.13: \[3966\]](#) How do I know whether character `x` is available in all character sets?
- [B.11.14: \[3967\]](#) Why do CJK strings sort incorrectly in Unicode? (I)
- [B.11.15: \[3968\]](#) Why do CJK strings sort incorrectly in Unicode? (II)
- [B.11.16: \[3968\]](#) Why are my supplementary characters rejected by MySQL?

- [B.11.17: \[3969\]](#) Shouldn't it be “CJKV”?
- [B.11.18: \[3969\]](#) Does MySQL allow CJK characters to be used in database and table names?
- [B.11.19: \[3969\]](#) Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?
- [B.11.20: \[3969\]](#) Where can I get help with CJK and related issues in MySQL?

Questions and Answers

B.11.1: What CJK character sets are available in MySQL?

The list of CJK character sets may vary depending on your MySQL version. For example, the `ucjpm` character set was not supported prior to MySQL 5.0.3. However, since the name of the applicable language appears in the `DESCRIPTION` column for every entry in the `INFORMATION_SCHEMA.CHARACTER_SETS` table, you can obtain a current list of all the non-Unicode CJK character sets using this query:

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
-> FROM INFORMATION_SCHEMA.CHARACTER_SETS
-> WHERE DESCRIPTION LIKE '%Chinese%'
-> OR DESCRIPTION LIKE '%Japanese%'
-> OR DESCRIPTION LIKE '%Korean%'
-> ORDER BY CHARACTER_SET_NAME;
+-----+-----+
| CHARACTER_SET_NAME | DESCRIPTION |
+-----+-----+
big5	Big5 Traditional Chinese
cp932	SJIS for Windows Japanese
ucjpm	UJIS for Windows Japanese
euckr	EUC-KR Korean
gb2312	GB2312 Simplified Chinese
gbk	GBK Simplified Chinese
sjis	Shift-JIS Japanese
ujis	EUC-JP Japanese
+-----+-----+
8 rows in set (0.01 sec)
```

(See [Section 20.1, “The `INFORMATION_SCHEMA.CHARACTER_SETS` Table”](#), for more information.)

MySQL supports the two common variants of the *GB* (*Guojia Biaozhun*, or *National Standard*, or *Simplified Chinese*) character sets which are official in the People's Republic of China: `gb2312` and `gbk`. Sometimes people try to insert `gbk` characters into `gb2312`, and it works most of the time because `gbk` is a superset of `gb2312`—but eventually they try to insert a rarer Chinese character and it doesn't work. (See [Bug #16072](#) for an example).

Here, we try to clarify exactly what characters are legitimate in `gb2312` or `gbk`, with reference to the official documents. Please check these references before reporting `gb2312` or `gbk` bugs.

- For a complete listing of the `gb2312` characters, ordered according to the `gb2312_chinese_ci` collation: [gb2312](#)
- MySQL's `gbk` is in reality “Microsoft code page 936”. This differs from the official `gbk` for characters `A1A4` (middle dot), `A1AA` (em dash), `A6E0–A6F5`, and `A8BB–A8C0`.
- For a listing of `gbk`/Unicode mappings, see <http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT>.
- For MySQL's listing of `gbk` characters, see [gbk](#).

B.11.2: I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters?

This problem is usually due to a setting in MySQL that doesn't match the settings for the application program or the operating system. Here are some common steps for correcting these types of issues:

- *Be certain of what MySQL version you are using.*

Use the statement `SELECT VERSION();` to determine this.

- *Make sure that the database is actually using the desired character set.*

People often think that the client character set is always the same as either the server character set or the character set used for display purposes. However, both of these are false assumptions. You can make sure by checking the result of `SHOW CREATE TABLE tablename` or—better yet—by using this statement:

```
SELECT character_set_name, collation_name
FROM information_schema.columns
WHERE table_schema = your_database_name
AND table_name = your_table_name
AND column_name = your_column_name;
```

- *Determine the hexadecimal value of the character or characters that are not being displayed correctly.*

You can obtain this information for a column `column_name` in the table `table_name` using the following query:

```
SELECT HEX(column_name)
FROM table_name;
```

`3F` is the encoding for the `?` character; this means that `?` is the character actually stored in the column. This most often happens because of a problem converting a particular character from your client character set to the target character set.

- *Make sure that a round trip possible—that is, when you select `literal` (or `_introducer hexadecimal-value`), you obtain `literal` as a result.*

For example, the Japanese *Katakana* character *Pe* (`ペ`) exists in all CJK character sets, and has the code point value (hexadecimal coding) `0x30da`. To test a round trip for this character, use this query:

```
SELECT 'ペ' AS `ペ`;          /* or SELECT _ucs2 0x30da; */
```

If the result is not also `ペ`, then the round trip has failed.

For bug reports regarding such failures, we might ask you to follow up with `SELECT HEX('ペ');`. Then we can determine whether the client encoding is correct.

- *Make sure that the problem is not with the browser or other application, rather than with MySQL.*

Use the `mysql` client program (on Windows: `mysql.exe`) to accomplish this task. If `mysql` displays correctly but your application doesn't, then your problem is probably due to system settings.

To find out what your settings are, use the `SHOW VARIABLES` statement, whose output should resemble what is shown here:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
```

```

| character_set_client      | utf8
| character_set_connection | utf8
| character_set_database   | latin1
| character_set_filesystem | binary
| character_set_results    | utf8
| character_set_server     | latin1
| character_set_system     | utf8
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/
+-----+
8 rows in set (0.03 sec)

```

These are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set and a default for MySQL).

Although Unicode (usually the `utf8` variant on Unix, and the `ucs2` variant on Windows) is preferable to Latin, it is often not what your operating system utilities support best. Many Windows users find that a Microsoft character set, such as `cp932` for Japanese Windows, is suitable.

If you cannot control the server settings, and you have no idea what your underlying computer is, then try changing to a common character set for the country that you're in (`euckr` = Korea; `gb2312` or `gbk` = People's Republic of China; `big5` = Taiwan; `sjis`, `ujis`, `cp932`, or `eucjpm` = Japan; `ucs2` or `utf8` = anywhere). Usually it is necessary to change only the client and connection and results settings. There is a simple statement which changes all three at once: `SET NAMES`. For example:

```
SET NAMES 'big5';
```

Once the setting is correct, you can make it permanent by editing `my.cnf` or `my.ini`. For example you might add lines looking like these:

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

It is also possible that there are issues with the API configuration setting being used in your application; see *Why does my GUI front end or browser not display CJK characters correctly...?* for more information.

B.11.3: What problems should I be aware of when working with the Big5 Chinese character set?

MySQL supports the Big5 character set which is common in Hong Kong and Taiwan (Republic of China). MySQL's `big5` is in reality Microsoft code page 950, which is very similar to the original `big5` character set. We changed to this character set starting with MySQL version 4.1.16 / 5.0.16 (as a result of Bug #12476). For example, the following statements work in current versions of MySQL, but not in old versions:

```
mysql> CREATE TABLE big5 (BIG5 CHAR(1) CHARACTER SET BIG5);
Query OK, 0 rows affected (0.13 sec)

mysql> INSERT INTO big5 VALUES (0xf9dc);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM big5;
+-----+
| big5 |
+-----+
| 嫻  |
+-----+
1 row in set (0.02 sec)
```

A feature request for adding `HKSCS` extensions has been filed. People who need this extension may find the suggested patch for Bug #13577 to be of interest.

B.11.4: Why do Japanese character set conversions fail?

MySQL supports the `sjis`, `ujis`, `cp932`, and `ujis` character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with `sjis` or `ujis`) and a Windows client (typically with `cp932`).

In the following conversion table, the `ucs2` column represents the source, and the `sjis`, `cp932`, `ujis`, and `ujis` columns represent the destinations—that is, the last 4 columns provide the hexadecimal result when we use `CONVERT(ucs2)` [1168] or we assign a `ucs2` column containing the value to an `sjis`, `cp932`, `ujis`, or `ujis` column.

| Character Name | ucs2 | sjis | cp932 | ujis | ujis |
|------------------------|------|------|-------|--------|------|
| BROKEN BAR | 00A6 | 3F | 3F | 8FA2C3 | 3F |
| FULLWIDTH BROKEN BAR | FFE4 | 3F | FA55 | 3F | 8FA2 |
| YEN SIGN | 00A5 | 3F | 3F | 20 | 3F |
| FULLWIDTH YEN SIGN | FFE5 | 818F | 818F | A1EF | 3F |
| TILDE | 007E | 7E | 7E | 7E | 7E |
| OVERLINE | 203E | 3F | 3F | 20 | 3F |
| HORIZONTAL BAR | 2015 | 815C | 815C | A1BD | A1BD |
| EM DASH | 2014 | 3F | 3F | 3F | 3F |
| REVERSE SOLIDUS | 005C | 815F | 5C | 5C | 5C |
| FULLWIDTH "" | FF3C | 3F | 815F | 3F | A1C0 |
| WAVE DASH | 301C | 8160 | 3F | A1C1 | 3F |
| FULLWIDTH TILDE | FF5E | 3F | 8160 | 3F | A1C1 |
| DOUBLE VERTICAL LINE | 2016 | 8161 | 3F | A1C2 | 3F |
| PARALLEL TO | 2225 | 3F | 8161 | 3F | A1C2 |
| MINUS SIGN | 2212 | 817C | 3F | A1DD | 3F |
| FULLWIDTH HYPHEN-MINUS | FF0D | 3F | 817C | 3F | A1DD |
| CENT SIGN | 00A2 | 8191 | 3F | A1F1 | 3F |
| FULLWIDTH CENT SIGN | FFE0 | 3F | 8191 | 3F | A1F1 |
| POUND SIGN | 00A3 | 8192 | 3F | A1F2 | 3F |
| FULLWIDTH POUND SIGN | FFE1 | 3F | 8192 | 3F | A1F2 |
| NOT SIGN | 00AC | 81CA | 3F | A2CC | 3F |
| FULLWIDTH NOT SIGN | FFE2 | 3F | 81CA | 3F | A2CC |

Now consider the following portion of the table.

| | ucs2 | sjis | cp932 |
|--------------------|------|------|-------|
| NOT SIGN | 00AC | 81CA | 3F |
| FULLWIDTH NOT SIGN | FFE2 | 3F | 81CA |

This means that MySQL converts the `NOT SIGN` (Unicode `U+00AC`) to `sjis` code point `0x81CA` and to `cp932` code point `3F`. (`3F` is the question mark (“?”)—this is what is always used when the conversion cannot be performed.)

B.11.5: What should I do if I want to convert SJIS 81CA to cp932?

Our answer is: “?”. There are serious complaints about this: many people would prefer a “loose” conversion, so that 81CA (NOT SIGN) in sjis becomes 81CA (FULLWIDTH NOT SIGN) in cp932. We are considering a change to this behavior.

B.11.6: How does MySQL represent the Yen (¥) sign?

A problem arises because some versions of Japanese character sets (both sjis and euc) treat 5C as a *reverse solidus* (\—also known as a backslash), and others treat it as a yen sign (¥).

MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description. In MySQL, 5C is always the *reverse solidus* (\).

B.11.7: Does MySQL plan to make a separate character set where 5C is the Yen sign, as at least one other major DBMS does?

This is one possible solution to the Yen sign issue; however, this will not happen in MySQL 5.1 or 6.0.

B.11.8: Of what issues should I be aware when working with Korean character sets in MySQL?

In theory, while there have been several versions of the euckr (*Extended Unix Code Korea*) character set, only one problem has been noted.

We use the “ASCII” variant of EUC-KR, in which the code point 0x5c is REVERSE SOLIDUS, that is \, instead of the “KS-Roman” variant of EUC-KR, in which the code point 0x5c is WON SIGN(₩). This means that you cannot convert Unicode U+20A9 to euckr:

```
mysql> SELECT
->     CONVERT('₩' USING euckr) AS euckr,
->     HEX(CONVERT('₩' USING euckr)) AS hexeuckr;
+-----+-----+
| euckr | hexeuckr |
+-----+-----+
| ?     | 3F       |
+-----+-----+
1 row in set (0.00 sec)
```

MySQL's graphic Korean chart is here: [euckr](#).

B.11.9: Why do I get Incorrect string value error messages?

For illustration, we'll create a table with one Unicode (ucs2) column and one Chinese (gb2312) column.

```
mysql> CREATE TABLE ch
->   (ucs2 CHAR(3) CHARACTER SET ucs2,
->   gb2312 CHAR(3) CHARACTER SET gb2312);
Query OK, 0 rows affected (0.05 sec)
```

We'll try to place the rare character ㄸ in both columns.

```
mysql> INSERT INTO ch VALUES ('AㄸB','AㄸB');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

Ah, there is a warning. Use the following statement to see what it is:

```
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1366
Message: Incorrect string value: '\xE6\xB1\x8CB' for column 'gb2312' at row 1
```

```
1 row in set (0.00 sec)
```

So it is a warning about the `gb2312` column only.

```
mysql> SELECT ucs2,HEX(ucs2),gb2312,HEX(gb2312) FROM ch;
+-----+-----+-----+-----+
| ucs2  | HEX(ucs2)  | gb2312 | HEX(gb2312) |
+-----+-----+-----+-----+
| A??B | 00416C4C0042 | A?B    | 413F42      |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Several things need explanation here:

1. The fact that it is a “warning” rather than an “error” is characteristic of MySQL. We like to try to do what we can, to get the best fit, rather than give up.
2. The `??` character is not in the `gb2312` character set. We described that problem earlier.
3. If you are using an old version of MySQL, you will probably see a different message.
4. With `sql_mode=TRADITIONAL` [556], there would be an error message, rather than a warning.

B.11.10: Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or another API?

Obtain a direct connection to the server using the `mysql` client (Windows: `mysql.exe`), and try the same query there. If `mysql` responds correctly, then the trouble may be that your application interface requires initialization. Use `mysql` to tell you what character set or sets it uses with the statement `SHOW VARIABLES LIKE 'char%'`. If you are using Access, then you are most likely connecting with Connector/ODBC. In this case, you should check [Section 22.1.4, “Configuring Connector/ODBC”](#). If, for instance, you use `big5`, you would enter `SET NAMES 'big5'`. (Note that no `;` is required in this case). If you are using ASP, you might need to add `SET NAMES` in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
    & "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

In much the same way, if you are using any character set other than `latin1` with Connector/Net, then you must specify the character set in the connection string. See [Section 22.2.5.1, “Connecting to MySQL Using Connector/Net”](#), for more information.

If you are using PHP, try this:

```
<?php
$link = mysql_connect($host, $usr, $pwd);

mysql_select_db($db);

if( mysql_error() ) { print "Database ERROR: " . mysql_error(); }
mysql_query("SET NAMES 'utf8'", $link);
?>
```

In this case, we used `SET NAMES` to change `character_set_client` [486] and `character_set_connection` [487] and `character_set_results` [488].

We encourage the use of the newer `mysqli` extension, rather than `mysql`. Using `mysqli`, the previous example could be rewritten as shown here:

```
<?php
$link = new mysqli($host, $usr, $pwd, $db);

if( mysqli_connect_errno() )
{
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$link->query("SET NAMES 'utf8'");
?>
```

Another issue often encountered in PHP applications has to do with assumptions made by the browser. Sometimes adding or changing a `<meta>` tag suffices to correct the problem: for example, to insure that the user agent interprets page content as UTF-8, you should include `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` in the `<head>` of the HTML page.

If you are using Connector/J, see [Section 22.3.5.4, “Using Character Sets and Unicode”](#).

B.11.11: I've upgraded to MySQL 5.5. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?

In MySQL Version 4.0, there was a single “global” character set for both server and client, and the decision as to which character to use was made by the server administrator. This changed starting with MySQL Version 4.1. What happens now is a “handshake”, as described in [Section 10.1.4, “Connection Character Sets and Collations”](#):

When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client` [486], `character_set_results` [488], and `character_set_connection` [487] system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

The effect of this is that you cannot control the client character set by starting `mysqld` with `--character-set-server=utf8` [438]. However, some of our Asian customers have said that they prefer the MySQL 4.0 behavior. To make it possible to retain this behavior, we added a `mysqld` switch, `--character-set-client-handshake` [437], which can be turned off with `--skip-character-set-client-handshake` [437]. If you start `mysqld` with `--skip-character-set-client-handshake` [437], then, when a client connects, it sends to the server the name of the character set that it wants to use—however, *the server ignores this request from the client*.

By way of example, suppose that your favorite server character set is `latin1` (unlikely in a CJK area, but this is the default value). Suppose further that the client uses `utf8` because this is what the client's operating system supports. Now, start the server with `latin1` as its default character set:

```
mysqld --character-set-server=latin1
```

And then start the client with the default character set `utf8`:

```
mysql --default-character-set=utf8
```

The current settings can be seen by viewing the output of `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+

```

```

+-----+-----+
character_set_client	utf8
character_set_connection	utf8
character_set_database	latin1
character_set_filesystem	binary
character_set_results	utf8
character_set_server	latin1
character_set_system	utf8
character_sets_dir	/usr/local/mysql/share/mysql/charsets/
+-----+-----+
8 rows in set (0.01 sec)

```

Now stop the client, and then stop the server using `mysqladmin`. Then start the server again, but this time tell it to skip the handshake like so:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

Start the client with `utf8` once again as the default character set, then display the current settings:

```

mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
character_set_client	latin1
character_set_connection	latin1
character_set_database	latin1
character_set_filesystem	binary
character_set_results	latin1
character_set_server	latin1
character_set_system	utf8
character_sets_dir	/usr/local/mysql/share/mysql/charsets/
+-----+-----+
8 rows in set (0.01 sec)

```

As you can see by comparing the differing results from `SHOW VARIABLES`, the server ignores the client's initial settings if the `--skip-character-set-client-handshake` [437] is used.

B.11.12: Why do some `LIKE` [1111] and `FULLTEXT` searches with CJK characters fail?

There is a very simple problem with `LIKE` [1111] searches on `BINARY` and `BLOB` columns: we need to know the end of a character. With multi-byte character sets, different characters might have different octet lengths. For example, in `utf8`, `A` requires one byte but `ㄹ` requires three bytes, as shown here:

```

+-----+-----+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 'ㄹ') |
+-----+-----+
| 1 | 3 |
+-----+-----+
1 row in set (0.00 sec)

```

If we don't know where the first character ends, then we don't know where the second character begins, in which case even very simple searches such as `LIKE '_A%'` [1111] fail. The solution is to use a regular CJK character set in the first place, or to convert to a CJK character set before comparing.

This is one reason why MySQL cannot allow encodings of nonexistent characters. If it is not strict about rejecting bad input, then it has no way of knowing where characters end.

For `FULLTEXT` searches, we need to know where words begin and end. With Western languages, this is rarely a problem because most (if not all) of these use an easy-to-identify word boundary—the space character. However, this is not usually the case with Asian writing. We could use arbitrary halfway measures, like assuming that all Han characters represent words, or (for Japanese) depending on changes from Katakana to Hiragana due to grammatical endings. However, the only sure solution requires a

comprehensive word list, which means that we would have to include a dictionary in the server for each Asian language supported. This is simply not feasible.

B.11.13: How do I know whether character *x* is available in all character sets?

The majority of simplified Chinese and basic nonhalfwidth Japanese *Kana* characters appear in all CJK character sets. This stored procedure accepts a [UCS-2](#) Unicode character, converts it to all other character sets, and displays the results in hexadecimal.

```
DELIMITER //

CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN

CREATE TABLE tj
    (ucs2 CHAR(1) character set ucs2,
    utf8 CHAR(1) character set utf8,
    big5 CHAR(1) character set big5,
    cp932 CHAR(1) character set cp932,
    eucjpms CHAR(1) character set eucjpms,
    euckr CHAR(1) character set euckr,
    gb2312 CHAR(1) character set gb2312,
    gbk CHAR(1) character set gbk,
    sjis CHAR(1) character set sjis,
    ujis CHAR(1) character set ujis);

INSERT INTO tj (ucs2) VALUES (ucs2_char);

UPDATE tj SET utf8=ucs2,
    big5=ucs2,
    cp932=ucs2,
    eucjpms=ucs2,
    euckr=ucs2,
    gb2312=ucs2,
    gbk=ucs2,
    sjis=ucs2,
    ujis=ucs2;

/* If there is a conversion problem, UPDATE will produce a warning. */

SELECT hex(ucs2) AS ucs2,
    hex(utf8) AS utf8,
    hex(big5) AS big5,
    hex(cp932) AS cp932,
    hex(eucjpms) AS eucjpms,
    hex(euckr) AS euckr,
    hex(gb2312) AS gb2312,
    hex(gbk) AS gbk,
    hex(sjis) AS sjis,
    hex(ujis) AS ujis
FROM tj;

DROP TABLE tj;

END//
```

The input can be any single `ucs2` character, or it can be the code point value (hexadecimal representation) of that character. For example, from Unicode's list of `ucs2` encodings and names (<http://www.unicode.org/Public/UNIDATA/UnicodeData.txt>), we know that the *Katakana* character *Pe* appears in all CJK character sets, and that its code point value is `0x30da`. If we use this value as the argument to `p_convert()`, the result is as shown here:

```
mysql> CALL p_convert(0x30da)//
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ucs2 | utf8 | big5 | cp932 | eucjpms | euckr | gb2312 | gbk | sjis | ujis |
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| 30DA | E3839A | C772 | 8379 | A5DA | ABDA | A5DA | A5DA | 8379 | A5DA |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)

```

Since none of the column values is 3F—that is, the question mark character (?)—we know that every conversion worked.

B.11.14: Why do CJK strings sort incorrectly in Unicode? (I)

Sometimes people observe that the result of a `utf8_unicode_ci` or `ucs2_unicode_ci` search, or of an `ORDER BY` sort is not what they think a native would expect. Although we never rule out the possibility that there is a bug, we have found in the past that many people do not read correctly the standard table of weights for the Unicode Collation Algorithm. MySQL uses the table found at <http://www.unicode.org/Public/UC4.0.0/allkeys-4.0.0.txt>. This is not the first table you will find by navigating from the [unicode.org](http://www.unicode.org) home page, because MySQL uses the older 4.0.0 “allkeys” table, rather than the more recent 4.1.0 table. (The newer ‘520’ collations in MySQL 5.6 use the 5.2 “allkeys” table.) This is because we are very wary about changing ordering which affects indexes, lest we bring about situations such as that reported in Bug #16526, illustrated as follows:

```

mysql> CREATE TABLE tj (s1 CHAR(1) CHARACTER SET utf8 COLLATE utf8_unicode_ci);
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO tj VALUES ('が'),('か');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM tj WHERE s1 = 'か';
+-----+
| s1 |
+-----+
| が |
| か |
+-----+
2 rows in set (0.00 sec)

```

The character in the first result row is not the one that we searched for. Why did MySQL retrieve it? First we look for the Unicode code point value, which is possible by reading the hexadecimal number for the `ucs2` version of the characters:

```

mysql> SELECT s1, HEX(CONVERT(s1 USING ucs2)) FROM tj;
+-----+-----+
| s1 | HEX(CONVERT(s1 USING ucs2)) |
+-----+-----+
| が | 304C |
| か | 304B |
+-----+-----+
2 rows in set (0.03 sec)

```

Now we search for 304B and 304C in the 4.0.0 `allkeys` table, and find these lines:

```

304B ; [.1E57.0020.000E.304B] # HIRAGANA LETTER KA
304C ; [.1E57.0020.000E.304B][.0000.0140.0002.3099] # HIRAGANA LETTER GA; QQCM

```

The official Unicode names (following the “#” mark) tell us the Japanese syllabary (Hiragana), the informal classification (letter, digit, or punctuation mark), and the Western identifier (`KA` or `GA`, which happen to be voiced and unvoiced components of the same letter pair). More importantly, the *primary weight* (the first hexadecimal number inside the square brackets) is 1E57 on both lines. For comparisons in both searching and sorting, MySQL pays attention to the primary weight only, ignoring all the other numbers. This means that we are sorting が and か correctly according to the Unicode specification. If we wanted to distinguish them, we’d have to use a non-UCA (Unicode Collation Algorithm) collation (`utf8_bin` or

`utf8_general_ci`), or to compare the `HEX()` [1104] values, or use `ORDER BY CONVERT(s1 USING sjis)`. Being correct “according to Unicode” isn’t enough, of course: the person who submitted the bug was equally correct. We plan to add another collation for Japanese according to the JIS X 4061 standard, in which voiced/unvoiced letter pairs like `KA/GA` are distinguishable for ordering purposes.

B.11.15: Why do CJK strings sort incorrectly in Unicode? (II)

If you are using Unicode (`ucs2` or `utf8`), and you know what the Unicode sort order is (see [Section B.11, “MySQL 5.5 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”](#)), but MySQL still seems to sort your table incorrectly, then you should first verify the table character set:

```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
`s1` char(1) CHARACTER SET ucs2 DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Since the character set appears to be correct, let’s see what information the `INFORMATION_SCHEMA.COLUMNS` table can provide about this column:

```
mysql> SELECT COLUMN_NAME, CHARACTER_SET_NAME, COLLATION_NAME
-> FROM INFORMATION_SCHEMA.COLUMNS
-> WHERE COLUMN_NAME = 's1'
-> AND TABLE_NAME = 't';
+-----+-----+-----+
| COLUMN_NAME | CHARACTER_SET_NAME | COLLATION_NAME |
+-----+-----+-----+
| s1          | ucs2               | ucs2_general_ci |
+-----+-----+-----+
1 row in set (0.01 sec)
```

(See [Section 20.4, “The INFORMATION_SCHEMA COLUMNS Table”](#), for more information.)

You can see that the collation is `ucs2_general_ci` instead of `ucs2_unicode_ci`. The reason why this is so can be found using `SHOW CHARSET`, as shown here:

```
mysql> SHOW CHARSET LIKE 'ucs2%';
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| ucs2    | UCS-2 Unicode | ucs2_general_ci   | 2      |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

For `ucs2` and `utf8`, the default collation is “general”. To specify a Unicode collation, use `COLLATE ucs2_unicode_ci`.

B.11.16: Why are my supplementary characters rejected by MySQL?

Before MySQL 5.5.3, MySQL does not support supplementary characters—that is, characters which need more than 3 bytes—for `UTF-8`. We support only what Unicode calls the *Basic Multilingual Plane / Plane 0*. Only a few very rare Han characters are supplementary; support for them is uncommon. This has led to reports such as that found in Bug #12600, which we rejected as “not a bug”. With `utf8`, we must truncate an input string when we encounter bytes that we don’t understand. Otherwise, we wouldn’t know how long the bad multi-byte character is.

One possible workaround is to use `ucs2` instead of `utf8`, in which case the “bad” characters are changed to question marks; however, no truncation takes place. You can also change the data type to `BLOB` or `BINARY`, which perform no validity checking.

As of MySQL 5.5.3, Unicode support is extended to include supplementary characters by means of additional Unicode character sets: `utf16`, `utf32`, and 4-byte `utf8mb4`. These character sets support supplementary Unicode characters outside the Basic Multilingual Plane (BMP).

B.11.17: Shouldn't it be “CJKV”?

No. The term “CJKV” (*Chinese Japanese Korean Vietnamese*) refers to Vietnamese character sets which contain Han (originally Chinese) characters. MySQL has no plan to support the old Vietnamese script using Han characters. MySQL does of course support the modern Vietnamese script with Western characters.

As of MySQL 5.6, there are Vietnamese collations for Unicode character sets, as described in [Section 10.1.14.1, “Unicode Character Sets”](#).

B.11.18: Does MySQL allow CJK characters to be used in database and table names?

This issue is fixed in MySQL 5.1, by automatically rewriting the names of the corresponding directories and files.

For example, if you create a database named 楮 on a server whose operating system does not support CJK in directory names, MySQL creates a directory named `@0w@00a5@00ae`. which is just a fancy way of encoding `E6A5AE`—that is, the Unicode hexadecimal representation for the 楮 character. However, if you run a `SHOW DATABASES` statement, you can see that the database is listed as 楮.

B.11.19: Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?

A Simplified Chinese version of the Manual, current for MySQL 5.1.12, can be found at <http://dev.mysql.com/doc/>. The Japanese translation of the MySQL 4.1 manual can be downloaded from <http://dev.mysql.com/doc/>.

B.11.20: Where can I get help with CJK and related issues in MySQL?

The following resources are available:

- A listing of MySQL user groups can be found at <http://dev.mysql.com/user-groups/>.
- View feature requests relating to character set issues at <http://tinyurl.com/y6xcuf>.
- Visit the MySQL [Character Sets, Collation, Unicode Forum](#). We are also in the process of adding foreign-language forums at <http://forums.mysql.com/>.

B.12. MySQL 5.5 FAQ: Connectors & APIs

For common questions, issues, and answers relating to the MySQL Connectors and other APIs, see the following areas of the Manual:

- [Section 22.8.11, “Common Questions and Problems When Using the C API”](#)
- [Section 22.9.12, “Common Problems with MySQL and PHP”](#)
- [Section 22.1.7, “Connector/ODBC Notes and Tips”](#)
- [Section 22.2.5, “Connector/Net Programming”](#)
- [Section 22.3, “MySQL Connector/J”](#)

B.13. MySQL 5.5 FAQ: Replication

In the following section, we provide answers to questions that are most frequently asked about MySQL Replication.

Questions

- [B.13.1: \[3970\]](#) Must the slave be connected to the master all the time?
- [B.13.2: \[3970\]](#) Must I enable networking on my master and slave to enable replication?
- [B.13.3: \[3970\]](#) How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?
- [B.13.4: \[3971\]](#) How do I force the master to block updates until the slave catches up?
- [B.13.5: \[3971\]](#) What issues should I be aware of when setting up two-way replication?
- [B.13.6: \[3971\]](#) How can I use replication to improve performance of my system?
- [B.13.7: \[3972\]](#) What should I do to prepare client code in my own applications to use performance-enhancing replication?
- [B.13.8: \[3972\]](#) When and how much can MySQL replication improve the performance of my system?
- [B.13.9: \[3973\]](#) How can I use replication to provide redundancy or high availability?
- [B.13.10: \[3973\]](#) How do I tell whether a master server is using statement-based or row-based binary logging format?
- [B.13.11: \[3973\]](#) How do I tell a slave to use row-based replication?
- [B.13.12: \[3973\]](#) How do I prevent `GRANT` and `REVOKE` statements from replicating to slave machines?
- [B.13.13: \[3973\]](#) Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on Mac OS X and Windows)?
- [B.13.14: \[3973\]](#) Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

Questions and Answers

B.13.1: Must the slave be connected to the master all the time?

No, it does not. The slave can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a master/slave relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the slave is not guaranteed to be in synchrony with the master unless you take some special measures.

To ensure that catchup can occur for a slave that has been disconnected, you must not remove binary log files from the master that contain information that has not yet been replicated to the slaves. Asynchronous replication can work only if the slave is able to continue reading the binary log from the point where it last read events.

B.13.2: Must I enable networking on my master and slave to enable replication?

Yes, networking must be enabled on the master and slave. If networking is not enabled, the slave cannot connect to the master and transfer the binary log. Check that the `skip-networking` [\[461\]](#) option has not been enabled in the configuration file for either server.

B.13.3: How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?

Check the `Seconds_Behind_Master` column in the output from `SHOW SLAVE STATUS`. See [Section 16.1.4.1, “Checking Replication Status”](#).

When the slave SQL thread executes an event read from the master, it modifies its own time to the event timestamp. (This is why `TIMESTAMP` is well replicated.) In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the slave SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. You can use this to determine the date of the last replicated event. Note that if your slave has been disconnected from the master for one hour, and then reconnects, you may immediately see large `Time` values such as 3600 for the slave SQL thread in `SHOW PROCESSLIST`. This is because the slave is executing statements that are one hour old. See [Section 16.2.1, “Replication Implementation Details”](#).

B.13.4: How do I force the master to block updates until the slave catches up?

Use the following procedure:

1. On the master, execute these statements:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

Record the replication coordinates (the current binary log file name and position) from the output of the `SHOW` statement.

2. On the slave, issue the following statement, where the arguments to the `MASTER_POS_WAIT()` [1200] function are the replication coordinate values obtained in the previous step:

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_pos);
```

The `SELECT` statement blocks until the slave reaches the specified log file and position. At that point, the slave is in synchrony with the master and the statement returns.

3. On the master, issue the following statement to enable the master to begin processing updates again:

```
mysql> UNLOCK TABLES;
```

B.13.5: What issues should I be aware of when setting up two-way replication?

MySQL replication currently does not support any locking protocol between master and slave to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-master 1, and in the meantime, before it propagates to co-master 2, client B could make an update to co-master 2 that makes the update of client A work differently than it did on co-master 1. Thus, when the update of client A makes it to co-master 2, it produces tables that are different from what you have on co-master 1, even after all the updates from co-master 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention because the updates originating on another server are serialized in one slave thread. Even this benefit might be offset by network delays.

B.13.6: How can I use replication to improve performance of my system?

Set up one server as the master and direct all writes to it. Then configure as many slaves as you have the budget and rackspace for, and distribute the reads among the master and the slaves. You can also

start the slaves with the `--skip-innodb` [1546], `--low-priority-updates` [452], and `--delay-key-write=ALL` [441] options to get speed improvements on the slave end. In this case, the slave uses nontransactional `MyISAM` tables instead of `InnoDB` tables to get more speed by eliminating transactional overhead.

B.13.7: What should I do to prepare client code in my own applications to use performance-enhancing replication?

See the guide to using replication as a scale-out solution, [Section 16.3.3, "Using Replication for Scale-Out"](#).

B.13.8: When and how much can MySQL replication improve the performance of my system?

MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-master/multiple-slave setup, you can scale the system by adding more slaves until you either run out of network bandwidth, or your update load grows to the point that the master cannot handle it.

To determine how many slaves you can use before the added benefits begin to level out, and how much you can improve performance of your site, you must know your query patterns, and determine empirically by benchmarking the relationship between the throughput for reads and writes on a typical master and a typical slave. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system. Let `reads` and `writes` denote the number of reads and writes per second, respectively.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `reads` is $1200 - 2 * \text{writes}$. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Suppose that the master and each slave have the same capacity, and that we have one master and N slaves. Then we have for each server (master or slave):

$$\text{reads} = 1200 - 2 * \text{writes}$$

$$\text{reads} = 9 * \text{writes} / (N + 1) \text{ (reads are split, but writes replicated to all slaves)}$$

$$9 * \text{writes} / (N + 1) + 2 * \text{writes} = 1200$$

$$\text{writes} = 1200 / (2 + 9/(N + 1))$$

The last equation indicates the maximum number of writes for N slaves, given a maximum possible read rate of 1,200 per second and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If $N = 0$ (which means we have no replication), our system can handle about $1200/11 = 109$ writes per second.
- If $N = 1$, we get up to 184 writes per second.
- If $N = 8$, we get up to 400 writes per second.
- If $N = 17$, we get up to 480 writes per second.
- Eventually, as N approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

Note that these computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what will happen on your system if you add N replication slaves. However, answering the following questions should help you decide whether and by how much replication will improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many slaves do you have bandwidth available on your network?

B.13.9: How can I use replication to provide redundancy or high availability?

How you implement redundancy is entirely dependent on your application and circumstances. High-availability solutions (with automatic failover) require active monitoring and either custom scripts or third party tools to provide the failover support from the original MySQL server to the slave.

To handle the process manually, you should be able to switch from a failed master to a pre-configured slave by altering your application to talk to the new server or by adjusting the DNS for the MySQL server from the failed server to the new server.

For more information and some example solutions, see [Section 16.3.6, “Switching Masters During Failover”](#).

B.13.10: How do I tell whether a master server is using statement-based or row-based binary logging format?

Check the value of the `binlog_format` [1899] system variable:

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

The value shown will be one of `STATEMENT`, `ROW`, or `MIXED`. For `MIXED` mode, row-based logging is preferred but replication switches automatically to statement-based logging under certain conditions; for information about when this may occur, see [Section 5.2.4.3, “Mixed Binary Logging Format”](#).

B.13.11: How do I tell a slave to use row-based replication?

Slaves automatically know which format to use.

B.13.12: How do I prevent `GRANT` and `REVOKE` statements from replicating to slave machines?

Start the server with the `--replicate-wild-ignore-table=mysql.%` [1880] option to ignore replication for tables in the `mysql` database.

B.13.13: Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on Mac OS X and Windows)?

Yes.

B.13.14: Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

Yes.

B.14. MySQL 5.5 FAQ: MySQL Enterprise Scalability Thread Pool

Questions

- [B.14.1: \[3974\]](#) What is the Thread Pool and what problem does it solve?
- [B.14.2: \[3974\]](#) How does the Thread Pool limit and manage concurrent sessions and transactions for optimal performance and throughput?
- [B.14.3: \[3974\]](#) How is the Thread Pool different from the client side Connection Pool?
- [B.14.4: \[3974\]](#) When should I use the Thread Pool?
- [B.14.5: \[3975\]](#) Are there recommended Thread Pool configurations?

Questions and Answers

B.14.1: What is the Thread Pool and what problem does it solve?

The MySQL Thread Pool is a MySQL server plugin that extends the default connection-handling capabilities of the MySQL server to limit the number of concurrently executing statements/queries and transactions to ensure that each has sufficient CPU and memory resources to fulfill its task. As of MySQL 5.5.16, commercial distributions of MySQL include the Thread Pool plugin.

The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. The Thread Pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The Thread Pool plugin increases server performance by efficiently managing statement execution threads for large numbers of client connections, especially on modern multi-CPU/Core systems.

For more information, see [Section 8.11.6, “The Thread Pool Plugin”](#).

B.14.2: How does the Thread Pool limit and manage concurrent sessions and transactions for optimal performance and throughput?

The Thread Pool uses a “divide and conquer” approach to limiting and balancing concurrency. Unlike the default connection handling of the MySQL Server, the Thread Pool separates connections and threads, so there is no fixed relationship between connections and the threads that execute statements received from those connections. The Thread Pool then manages client connections within configurable thread groups, where they are prioritized and queued based on the nature of the work they were submitted to accomplish.

For more information, see [Section 8.11.6.2, “Thread Pool Operation”](#).

B.14.3: How is the Thread Pool different from the client side Connection Pool?

The MySQL Connection Pool operates on the client side to ensure that a MySQL client does not constantly connect to and disconnect from the MySQL server. It is designed to cache idle connections in the MySQL client for use by other users as they are needed. This minimizes the overhead and expense of establishing and tearing down connections as queries are submitted to the MySQL server. The MySQL Connection Pool has no visibility as to the query handling capabilities or load of the backend MySQL server. By contrast, the Thread Pool operates on the MySQL server side and is designed to manage the execution of inbound concurrent connections and queries as they are received from the client connections accessing the backend MySQL database. Because of the separation of duties, the MySQL Connection Pool and Thread Pool are orthogonal and can be used independent of each other.

MySQL Connection Pooling via the MySQL Connectors is covered in [Chapter 22, Connectors and APIs](#).

B.14.4: When should I use the Thread Pool?

There are a few rules of thumb to consider for optimal Thread Pool use cases:

The MySQL [Threads_running \[611\]](#) variable keeps track of the number of concurrent statements currently executing in the MySQL Server. If this variable consistently exceeds a region where the server won't operate optimally (usually going beyond 40 for InnoDB workloads), the Thread Pool will be beneficial, especially in extreme parallel overload situations.

If you are using the [innodb_thread_concurrency \[1573\]](#) to limit the number of concurrently executing statements, you will find the Thread Pool solves the same problem, only better, by assigning connections to thread groups, then queuing executions based on transactional content, user defined designations, and so forth.

Lastly, if your workload comprises mainly short queries, the Thread Pool will be beneficial.

To learn more, see [Section 8.11.6.3, "Thread Pool Tuning"](#).

B.14.5: Are there recommended Thread Pool configurations?

The Thread Pool has a number of user case driven configuration parameters that affect its performance. To learn about these and tips on tuning, see [Section 8.11.6.3, "Thread Pool Tuning"](#).

Appendix C. Errors, Error Codes, and Common Problems

Table of Contents

| | |
|---|------|
| C.1. Sources of Error Information | 3977 |
| C.2. Types of Error Values | 3977 |
| C.3. Server Error Codes and Messages | 3978 |
| C.4. Client Error Codes and Messages | 4029 |
| C.5. Problems and Common Errors | 4033 |
| C.5.1. How to Determine What Is Causing a Problem | 4033 |
| C.5.2. Common Errors When Using MySQL Programs | 4035 |
| C.5.3. Installation-Related Issues | 4048 |
| C.5.4. Administration-Related Issues | 4048 |
| C.5.5. Query-Related Issues | 4055 |
| C.5.6. Optimizer-Related Issues | 4063 |
| C.5.7. Table Definition-Related Issues | 4063 |
| C.5.8. Known Issues in MySQL | 4064 |

This appendix lists common problems and errors that may occur and potential resolutions, in addition to listing the errors that may appear when you call MySQL from any host language. The first section covers problems and resolutions. Detailed information on errors is provided: One list displays server error messages. Another list displays client program messages.

C.1. Sources of Error Information

There are several sources of error information in MySQL:

- Each SQL statement executed results in an error code, an SQLSTATE value, and an error message, as described in [Section C.2, “Types of Error Values”](#). These errors are returned from the server side; see [Section C.3, “Server Error Codes and Messages”](#).
- Errors can occur on the client side, usually involving problems communicating with the server; see [Section C.4, “Client Error Codes and Messages”](#).
- SQL statement warning and error information is available through the `SHOW WARNINGS` and `SHOW ERRORS` statements. The `warning_count` [573] system variable indicates the number of errors, warnings, and notes. The `error_count` [498] system variable indicates the number of errors. Its value excludes warnings and notes.
- `SHOW SLAVE STATUS` statement output includes information about replication errors occurring on the slave side.
- `SHOW ENGINE INNODB STATUS` statement output includes information about the most recent foreign key error if a `CREATE TABLE` statement for an `InnoDB` table fails.
- The `pererror` program provides information from the command line about error numbers. See [Section 4.8.1, “pererror — Explain Error Codes”](#).

Descriptions of server and client errors are provided later in this Appendix. For information about errors related to `InnoDB`, see [Section 14.3.13, “InnoDB Error Handling”](#).

C.2. Types of Error Values

When an error occurs in MySQL, the server returns two types of error values:

- A MySQL-specific error code. This value is numeric. It is not portable to other database systems.
- An SQLSTATE value. The value is a five-character string (for example, '42S02'). The values are taken from ANSI SQL and ODBC and are more standardized.

A message string that provides a textual description of the error is also available.

When an error occurs, the MySQL error code, SQLSTATE value, and message string are available using C API functions:

- MySQL error code: Call `mysql_errno()`
- SQLSTATE value: Call `mysql_sqlstate()`
- Error message: Call `mysql_error()`

For prepared statements, the corresponding error functions are `mysql_stmt_errno()`, `mysql_stmt_sqlstate()`, and `mysql_stmt_error()`. All error functions are described in [Section 22.8, “MySQL C API”](#).

The number of errors, warnings, and notes for the previous statement can be obtained by calling `mysql_warning_count()`. See [Section 22.8.3.72, “mysql_warning_count\(\)”](#).

The first two characters of an SQLSTATE value indicate the error class:

- Class = '00' indicates success.
- Class = '01' indicates a warning.
- Class = '02' indicates “not found.” This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. This condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.
- Class > '02' indicates an exception.

C.3. Server Error Codes and Messages

MySQL programs have access to several types of error information when the server returns an error. For example, the `mysql` client program displays errors using the following format:

```
shell> SELECT * FROM no_such_table;  
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

The message displayed contains three types of information:

- A numeric error code (1146). This number is MySQL-specific and is not portable to other database systems.
- A five-character SQLSTATE value ('42S02'). The values are taken from ANSI SQL and ODBC and are more standardized. Not all MySQL error numbers have corresponding SQLSTATE values. In these cases, 'HY000' (general error) is used.
- A message string that provides a textual description of the error.

For error checking, use error codes, not error messages. Error messages do not change often, but it is possible. Also if the database administrator changes the language setting, that affects the language of error messages.

Error codes are stable across GA releases of a given MySQL series. Before a series reaches GA status, new codes may still be under development and subject to change.

Server error information comes from the following source files. For details about the way that error information is defined, see the [MySQL Internals Manual](#).

- Error message information is listed in the `share/errmsg.txt` file. `%d` and `%s` represent numbers and strings, respectively, that are substituted into the Message values when they are displayed.
- The Error values listed in `share/errmsg.txt` are used to generate the definitions in the `include/mysql_error.h` and `include/mysql_ename.h` MySQL source files.
- The SQLSTATE values listed in `share/errmsg.txt` are used to generate the definitions in the `include/sql_state.h` MySQL source file.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: 1000 SQLSTATE: HY000 ([ER_HASHCHK \[3979\]](#))
Message: hashchk
- Error: 1001 SQLSTATE: HY000 ([ER_NISAMCHK \[3979\]](#))
Message: isamchk
- Error: 1002 SQLSTATE: HY000 ([ER_NO \[3979\]](#))
Message: NO
- Error: 1003 SQLSTATE: HY000 ([ER_YES \[3979\]](#))
Message: YES
- Error: 1004 SQLSTATE: HY000 ([ER_CANT_CREATE_FILE \[3979\]](#))
Message: Can't create file '%s' (errno: %d)
- Error: 1005 SQLSTATE: HY000 ([ER_CANT_CREATE_TABLE \[3979\]](#))
Message: Can't create table '%s' (errno: %d)
- Error: 1006 SQLSTATE: HY000 ([ER_CANT_CREATE_DB \[3979\]](#))
Message: Can't create database '%s' (errno: %d)
- Error: 1007 SQLSTATE: HY000 ([ER_DB_CREATE_EXISTS \[3979\]](#))
Message: Can't create database '%s'; database exists
- Error: 1008 SQLSTATE: HY000 ([ER_DB_DROP_EXISTS \[3979\]](#))
Message: Can't drop database '%s'; database doesn't exist
- Error: 1009 SQLSTATE: HY000 ([ER_DB_DROP_DELETE \[3979\]](#))
Message: Error dropping database (can't delete '%s', errno: %d)
- Error: 1010 SQLSTATE: HY000 ([ER_DB_DROP_RMDIR \[3979\]](#))
Message: Error dropping database (can't rmdir '%s', errno: %d)
- Error: 1011 SQLSTATE: HY000 ([ER_CANT_DELETE_FILE \[3979\]](#))

Message: Error on delete of '%s' (errno: %d)

- Error: 1012 SQLSTATE: HY000 ([ER_CANT_FIND_SYSTEM_REC \[3980\]](#))

Message: Can't read record in system table

- Error: 1013 SQLSTATE: HY000 ([ER_CANT_GET_STAT \[3980\]](#))

Message: Can't get status of '%s' (errno: %d)

- Error: 1014 SQLSTATE: HY000 ([ER_CANT_GET_WD \[3980\]](#))

Message: Can't get working directory (errno: %d)

- Error: 1015 SQLSTATE: HY000 ([ER_CANT_LOCK \[3980\]](#))

Message: Can't lock file (errno: %d)

- Error: 1016 SQLSTATE: HY000 ([ER_CANT_OPEN_FILE \[3980\]](#))

Message: Can't open file: '%s' (errno: %d)

- Error: 1017 SQLSTATE: HY000 ([ER_FILE_NOT_FOUND \[3980\]](#))

Message: Can't find file: '%s' (errno: %d)

- Error: 1018 SQLSTATE: HY000 ([ER_CANT_READ_DIR \[3980\]](#))

Message: Can't read dir of '%s' (errno: %d)

- Error: 1019 SQLSTATE: HY000 ([ER_CANT_SET_WD \[3980\]](#))

Message: Can't change dir to '%s' (errno: %d)

- Error: 1020 SQLSTATE: HY000 ([ER_CHECKREAD \[3980\]](#))

Message: Record has changed since last read in table '%s'

- Error: 1021 SQLSTATE: HY000 ([ER_DISK_FULL \[3980\]](#))

Message: Disk full (%s); waiting for someone to free some space...

- Error: 1022 SQLSTATE: 23000 ([ER_DUP_KEY \[3980\]](#))

Message: Can't write; duplicate key in table '%s'

- Error: 1023 SQLSTATE: HY000 ([ER_ERROR_ON_CLOSE \[3980\]](#))

Message: Error on close of '%s' (errno: %d)

- Error: 1024 SQLSTATE: HY000 ([ER_ERROR_ON_READ \[3980\]](#))

Message: Error reading file '%s' (errno: %d)

- Error: 1025 SQLSTATE: HY000 ([ER_ERROR_ON_RENAME \[3980\]](#))

Message: Error on rename of '%s' to '%s' (errno: %d)

- Error: 1026 SQLSTATE: HY000 ([ER_ERROR_ON_WRITE \[3980\]](#))

Message: Error writing file '%s' (errno: %d)

- Error: 1027 SQLSTATE: HY000 (ER_FILE_USED [3981])

Message: '%s' is locked against change

- Error: 1028 SQLSTATE: HY000 (ER_FILESORT_ABORT [3981])

Message: Sort aborted

- Error: 1029 SQLSTATE: HY000 (ER_FORM_NOT_FOUND [3981])

Message: View '%s' doesn't exist for '%s'

- Error: 1030 SQLSTATE: HY000 (ER_GET_ERRNO [3981])

Message: Got error %d from storage engine

- Error: 1031 SQLSTATE: HY000 (ER_ILLEGAL HA [3981])

Message: Table storage engine for '%s' doesn't have this option

- Error: 1032 SQLSTATE: HY000 (ER_KEY_NOT_FOUND [3981])

Message: Can't find record in '%s'

- Error: 1033 SQLSTATE: HY000 (ER_NOT_FORM_FILE [3981])

Message: Incorrect information in file: '%s'

- Error: 1034 SQLSTATE: HY000 (ER_NOT_KEYFILE [3981])

Message: Incorrect key file for table '%s'; try to repair it

- Error: 1035 SQLSTATE: HY000 (ER_OLD_KEYFILE [3981])

Message: Old key file for table '%s'; repair it!

- Error: 1036 SQLSTATE: HY000 (ER_OPEN_AS_READONLY [3981])

Message: Table '%s' is read only

- Error: 1037 SQLSTATE: HY001 (ER_OUTOFMEMORY [3981])

Message: Out of memory; restart server and try again (needed %d bytes)

- Error: 1038 SQLSTATE: HY001 (ER_OUT_OF_SORTMEMORY [3981])

Message: Out of sort memory, consider increasing server sort buffer size

- Error: 1039 SQLSTATE: HY000 (ER_UNEXPECTED_EOF [3981])

Message: Unexpected EOF found when reading file '%s' (errno: %d)

- Error: 1040 SQLSTATE: 08004 (ER_CON_COUNT_ERROR [3981])

Message: Too many connections

- Error: 1041 SQLSTATE: HY000 (ER_OUT_OF_RESOURCES [3981])

Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space

- Error: 1042 SQLSTATE: 08S01 ([ER_BAD_HOST_ERROR \[3982\]](#))

Message: Can't get hostname for your address

- Error: 1043 SQLSTATE: 08S01 ([ER_HANDSHAKE_ERROR \[3982\]](#))

Message: Bad handshake

- Error: 1044 SQLSTATE: 42000 ([ER_DBACCESS_DENIED_ERROR \[3982\]](#))

Message: Access denied for user '%s'@'%s' to database '%s'

- Error: 1045 SQLSTATE: 28000 ([ER_ACCESS_DENIED_ERROR \[3982\]](#))

Message: Access denied for user '%s'@'%s' (using password: %s)

- Error: 1046 SQLSTATE: 3D000 ([ER_NO_DB_ERROR \[3982\]](#))

Message: No database selected

- Error: 1047 SQLSTATE: 08S01 ([ER_UNKNOWN_COM_ERROR \[3982\]](#))

Message: Unknown command

- Error: 1048 SQLSTATE: 23000 ([ER_BAD_NULL_ERROR \[3982\]](#))

Message: Column '%s' cannot be null

- Error: 1049 SQLSTATE: 42000 ([ER_BAD_DB_ERROR \[3982\]](#))

Message: Unknown database '%s'

- Error: 1050 SQLSTATE: 42S01 ([ER_TABLE_EXISTS_ERROR \[3982\]](#))

Message: Table '%s' already exists

- Error: 1051 SQLSTATE: 42S02 ([ER_BAD_TABLE_ERROR \[3982\]](#))

Message: Unknown table '%s'

- Error: 1052 SQLSTATE: 23000 ([ER_NON_UNIQ_ERROR \[3982\]](#))

Message: Column '%s' in %s is ambiguous

- Error: 1053 SQLSTATE: 08S01 ([ER_SERVER_SHUTDOWN \[3982\]](#))

Message: Server shutdown in progress

- Error: 1054 SQLSTATE: 42S22 ([ER_BAD_FIELD_ERROR \[3982\]](#))

Message: Unknown column '%s' in '%s'

- Error: 1055 SQLSTATE: 42000 ([ER_WRONG_FIELD_WITH_GROUP \[3982\]](#))

Message: '%s' isn't in GROUP BY

- Error: 1056 SQLSTATE: 42000 (ER_WRONG_GROUP_FIELD [3983])
Message: Can't group on '%s'
- Error: 1057 SQLSTATE: 42000 (ER_WRONG_SUM_SELECT [3983])
Message: Statement has sum functions and columns in same statement
- Error: 1058 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT [3983])
Message: Column count doesn't match value count
- Error: 1059 SQLSTATE: 42000 (ER_TOO_LONG_IDENT [3983])
Message: Identifier name '%s' is too long
- Error: 1060 SQLSTATE: 42S21 (ER_DUP_FIELDNAME [3983])
Message: Duplicate column name '%s'
- Error: 1061 SQLSTATE: 42000 (ER_DUP_KEYNAME [3983])
Message: Duplicate key name '%s'
- Error: 1062 SQLSTATE: 23000 (ER_DUP_ENTRY [3983])
Message: Duplicate entry '%s' for key %d
- Error: 1063 SQLSTATE: 42000 (ER_WRONG_FIELD_SPEC [3983])
Message: Incorrect column specifier for column '%s'
- Error: 1064 SQLSTATE: 42000 (ER_PARSE_ERROR [3983])
Message: %s near '%s' at line %d
- Error: 1065 SQLSTATE: 42000 (ER_EMPTY_QUERY [3983])
Message: Query was empty
- Error: 1066 SQLSTATE: 42000 (ER_NONUNIQ_TABLE [3983])
Message: Not unique table/alias: '%s'
- Error: 1067 SQLSTATE: 42000 (ER_INVALID_DEFAULT [3983])
Message: Invalid default value for '%s'
- Error: 1068 SQLSTATE: 42000 (ER_MULTIPLE_PRI_KEY [3983])
Message: Multiple primary key defined
- Error: 1069 SQLSTATE: 42000 (ER_TOO_MANY_KEYS [3983])
Message: Too many keys specified; max %d keys allowed
- Error: 1070 SQLSTATE: 42000 (ER_TOO_MANY_KEY_PARTS [3983])
Message: Too many key parts specified; max %d parts allowed

- Error: 1071 SQLSTATE: 42000 ([ER_TOO_LONG_KEY \[3984\]](#))
Message: Specified key was too long; max key length is %d bytes
- Error: 1072 SQLSTATE: 42000 ([ER_KEY_COLUMN_DOES_NOT_EXISTS \[3984\]](#))
Message: Key column '%s' doesn't exist in table
- Error: 1073 SQLSTATE: 42000 ([ER_BLOB_USED_AS_KEY \[3984\]](#))
Message: BLOB column '%s' can't be used in key specification with the used table type
- Error: 1074 SQLSTATE: 42000 ([ER_TOO_BIG_FIELDLENGTH \[3984\]](#))
Message: Column length too big for column '%s' (max = %lu); use BLOB or TEXT instead
- Error: 1075 SQLSTATE: 42000 ([ER_WRONG_AUTO_KEY \[3984\]](#))
Message: Incorrect table definition; there can be only one auto column and it must be defined as a key
- Error: 1076 SQLSTATE: HY000 ([ER_READY \[3984\]](#))
Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d
- Error: 1077 SQLSTATE: HY000 ([ER_NORMAL_SHUTDOWN \[3984\]](#))
Message: %s: Normal shutdown
- Error: 1078 SQLSTATE: HY000 ([ER_GOT_SIGNAL \[3984\]](#))
Message: %s: Got signal %d. Aborting!
- Error: 1079 SQLSTATE: HY000 ([ER_SHUTDOWN_COMPLETE \[3984\]](#))
Message: %s: Shutdown complete
- Error: 1080 SQLSTATE: 08S01 ([ER_FORCING_CLOSE \[3984\]](#))
Message: %s: Forcing close of thread %ld user: '%s'
- Error: 1081 SQLSTATE: 08S01 ([ER_IPSOCKET_ERROR \[3984\]](#))
Message: Can't create IP socket
- Error: 1082 SQLSTATE: 42S12 ([ER_NO_SUCH_INDEX \[3984\]](#))
Message: Table '%s' has no index like the one used in CREATE INDEX; recreate the table
- Error: 1083 SQLSTATE: 42000 ([ER_WRONG_FIELD_TERMINATORS \[3984\]](#))
Message: Field separator argument is not what is expected; check the manual
- Error: 1084 SQLSTATE: 42000 ([ER_BLOBS_AND_NO_TERMINATED \[3984\]](#))
Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'
- Error: 1085 SQLSTATE: HY000 ([ER_TEXTFILE_NOT_READABLE \[3984\]](#))
Message: The file '%s' must be in the database directory or be readable by all

- Error: 1086 SQLSTATE: HY000 ([ER_FILE_EXISTS_ERROR \[3985\]](#))
Message: File '%s' already exists
- Error: 1087 SQLSTATE: HY000 ([ER_LOAD_INFO \[3985\]](#))
Message: Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld
- Error: 1088 SQLSTATE: HY000 ([ER_ALTER_INFO \[3985\]](#))
Message: Records: %ld Duplicates: %ld
- Error: 1089 SQLSTATE: HY000 ([ER_WRONG_SUB_KEY \[3985\]](#))
Message: Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique prefix keys
- Error: 1090 SQLSTATE: 42000 ([ER_CANT_REMOVE_ALL_FIELDS \[3985\]](#))
Message: You can't delete all columns with ALTER TABLE; use DROP TABLE instead
- Error: 1091 SQLSTATE: 42000 ([ER_CANT_DROP_FIELD_OR_KEY \[3985\]](#))
Message: Can't DROP '%s'; check that column/key exists
- Error: 1092 SQLSTATE: HY000 ([ER_INSERT_INFO \[3985\]](#))
Message: Records: %ld Duplicates: %ld Warnings: %ld
- Error: 1093 SQLSTATE: HY000 ([ER_UPDATE_TABLE_USED \[3985\]](#))
Message: You can't specify target table '%s' for update in FROM clause
- Error: 1094 SQLSTATE: HY000 ([ER_NO_SUCH_THREAD \[3985\]](#))
Message: Unknown thread id: %lu
- Error: 1095 SQLSTATE: HY000 ([ER_KILL_DENIED_ERROR \[3985\]](#))
Message: You are not owner of thread %lu
- Error: 1096 SQLSTATE: HY000 ([ER_NO_TABLES_USED \[3985\]](#))
Message: No tables used
- Error: 1097 SQLSTATE: HY000 ([ER_TOO_BIG_SET \[3985\]](#))
Message: Too many strings for column %s and SET
- Error: 1098 SQLSTATE: HY000 ([ER_NO_UNIQUE_LOGFILE \[3985\]](#))
Message: Can't generate a unique log-filename %s.(1-999)
- Error: 1099 SQLSTATE: HY000 ([ER_TABLE_NOT_LOCKED_FOR_WRITE \[3985\]](#))
Message: Table '%s' was locked with a READ lock and can't be updated
- Error: 1100 SQLSTATE: HY000 ([ER_TABLE_NOT_LOCKED \[3985\]](#))
Message: Table '%s' was not locked with LOCK TABLES

- Error: 1101 SQLSTATE: 42000 ([ER_BLOB_CANT_HAVE_DEFAULT \[3986\]](#))
Message: BLOB/TEXT column '%s' can't have a default value
- Error: 1102 SQLSTATE: 42000 ([ER_WRONG_DB_NAME \[3986\]](#))
Message: Incorrect database name '%s'
- Error: 1103 SQLSTATE: 42000 ([ER_WRONG_TABLE_NAME \[3986\]](#))
Message: Incorrect table name '%s'
- Error: 1104 SQLSTATE: 42000 ([ER_TOO_BIG_SELECT \[3986\]](#))
Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET MAX_JOIN_SIZE=# if the SELECT is okay
- Error: 1105 SQLSTATE: HY000 ([ER_UNKNOWN_ERROR \[3986\]](#))
Message: Unknown error
- Error: 1106 SQLSTATE: 42000 ([ER_UNKNOWN_PROCEDURE \[3986\]](#))
Message: Unknown procedure '%s'
- Error: 1107 SQLSTATE: 42000 ([ER_WRONG_PARAMCOUNT_TO_PROCEDURE \[3986\]](#))
Message: Incorrect parameter count to procedure '%s'
- Error: 1108 SQLSTATE: HY000 ([ER_WRONG_PARAMETERS_TO_PROCEDURE \[3986\]](#))
Message: Incorrect parameters to procedure '%s'
- Error: 1109 SQLSTATE: 42S02 ([ER_UNKNOWN_TABLE \[3986\]](#))
Message: Unknown table '%s' in %s
- Error: 1110 SQLSTATE: 42000 ([ER_FIELD_SPECIFIED_TWICE \[3986\]](#))
Message: Column '%s' specified twice
- Error: 1111 SQLSTATE: HY000 ([ER_INVALID_GROUP_FUNC_USE \[3986\]](#))
Message: Invalid use of group function
- Error: 1112 SQLSTATE: 42000 ([ER_UNSUPPORTED_EXTENSION \[3986\]](#))
Message: Table '%s' uses an extension that doesn't exist in this MySQL version
- Error: 1113 SQLSTATE: 42000 ([ER_TABLE_MUST_HAVE_COLUMNS \[3986\]](#))
Message: A table must have at least 1 column
- Error: 1114 SQLSTATE: HY000 ([ER_RECORD_FILE_FULL \[3986\]](#))
Message: The table '%s' is full
- Error: 1115 SQLSTATE: 42000 ([ER_UNKNOWN_CHARACTER_SET \[3986\]](#))
Message: Unknown character set: '%s'

- Error: [1116](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_TABLES](#) [[3987](#)])
Message: Too many tables; MySQL can only use %d tables in a join
- Error: [1117](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_FIELDS](#) [[3987](#)])
Message: Too many columns
- Error: [1118](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_ROWSIZE](#) [[3987](#)])
Message: Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some columns to TEXT or BLOBs
- Error: [1119](#) SQLSTATE: [HY000](#) ([ER_STACK_OVERRUN](#) [[3987](#)])
Message: Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld --thread_stack=#' to specify a bigger stack if needed
- Error: [1120](#) SQLSTATE: [42000](#) ([ER_WRONG_OUTER_JOIN](#) [[3987](#)])
Message: Cross dependency found in OUTER JOIN; examine your ON conditions
- Error: [1121](#) SQLSTATE: [42000](#) ([ER_NULL_COLUMN_IN_INDEX](#) [[3987](#)])
Message: Table handler doesn't support NULL in given index. Please change column '%s' to be NOT NULL or use another handler
- Error: [1122](#) SQLSTATE: [HY000](#) ([ER_CANT_FIND_UDF](#) [[3987](#)])
Message: Can't load function '%s'
- Error: [1123](#) SQLSTATE: [HY000](#) ([ER_CANT_INITIALIZE_UDF](#) [[3987](#)])
Message: Can't initialize function '%s'; %s
- Error: [1124](#) SQLSTATE: [HY000](#) ([ER_UDF_NO_PATHS](#) [[3987](#)])
Message: No paths allowed for shared library
- Error: [1125](#) SQLSTATE: [HY000](#) ([ER_UDF_EXISTS](#) [[3987](#)])
Message: Function '%s' already exists
- Error: [1126](#) SQLSTATE: [HY000](#) ([ER_CANT_OPEN_LIBRARY](#) [[3987](#)])
Message: Can't open shared library '%s' (errno: %d %s)
- Error: [1127](#) SQLSTATE: [HY000](#) ([ER_CANT_FIND_DL_ENTRY](#) [[3987](#)])
Message: Can't find symbol '%s' in library
- Error: [1128](#) SQLSTATE: [HY000](#) ([ER_FUNCTION_NOT_DEFINED](#) [[3987](#)])
Message: Function '%s' is not defined
- Error: [1129](#) SQLSTATE: [HY000](#) ([ER_HOST_IS_BLOCKED](#) [[3987](#)])
Message: Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

- Error: 1130 SQLSTATE: HY000 ([ER_HOST_NOT_PRIVILEGED \[3988\]](#))
Message: Host '%s' is not allowed to connect to this MySQL server
- Error: 1131 SQLSTATE: 42000 ([ER_PASSWORD_ANONYMOUS_USER \[3988\]](#))
Message: You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords
- Error: 1132 SQLSTATE: 42000 ([ER_PASSWORD_NOT_ALLOWED \[3988\]](#))
Message: You must have privileges to update tables in the mysql database to be able to change passwords for others
- Error: 1133 SQLSTATE: 42000 ([ER_PASSWORD_NO_MATCH \[3988\]](#))
Message: Can't find any matching row in the user table
- Error: 1134 SQLSTATE: HY000 ([ER_UPDATE_INFO \[3988\]](#))
Message: Rows matched: %ld Changed: %ld Warnings: %ld
- Error: 1135 SQLSTATE: HY000 ([ER_CANT_CREATE_THREAD \[3988\]](#))
Message: Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug
- Error: 1136 SQLSTATE: 21S01 ([ER_WRONG_VALUE_COUNT_ON_ROW \[3988\]](#))
Message: Column count doesn't match value count at row %ld
- Error: 1137 SQLSTATE: HY000 ([ER_CANT_REOPEN_TABLE \[3988\]](#))
Message: Can't reopen table: '%s'
- Error: 1138 SQLSTATE: 22004 ([ER_INVALID_USE_OF_NULL \[3988\]](#))
Message: Invalid use of NULL value
- Error: 1139 SQLSTATE: 42000 ([ER_REGEXP_ERROR \[3988\]](#))
Message: Got error '%s' from regexp
- Error: 1140 SQLSTATE: 42000 ([ER_MIX_OF_GROUP_FUNC_AND_FIELDS \[3988\]](#))
Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause
- Error: 1141 SQLSTATE: 42000 ([ER_NONEXISTING_GRANT \[3988\]](#))
Message: There is no such grant defined for user '%s' on host '%s'
- Error: 1142 SQLSTATE: 42000 ([ER_TABLEACCESS_DENIED_ERROR \[3988\]](#))
Message: %s command denied to user '%s'@'%s' for table '%s'
- Error: 1143 SQLSTATE: 42000 ([ER_COLUMNACCESS_DENIED_ERROR \[3988\]](#))
Message: %s command denied to user '%s'@'%s' for column '%s' in table '%s'

- Error: [1144](#) SQLSTATE: [42000](#) ([ER_ILLEGAL_GRANT_FOR_TABLE](#) [[3989](#)])
Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used
- Error: [1145](#) SQLSTATE: [42000](#) ([ER_GRANT_WRONG_HOST_OR_USER](#) [[3989](#)])
Message: The host or user argument to GRANT is too long
- Error: [1146](#) SQLSTATE: [42S02](#) ([ER_NO_SUCH_TABLE](#) [[3989](#)])
Message: Table '%s.%s' doesn't exist
- Error: [1147](#) SQLSTATE: [42000](#) ([ER_NONEXISTING_TABLE_GRANT](#) [[3989](#)])
Message: There is no such grant defined for user '%s' on host '%s' on table '%s'
- Error: [1148](#) SQLSTATE: [42000](#) ([ER_NOT_ALLOWED_COMMAND](#) [[3989](#)])
Message: The used command is not allowed with this MySQL version
- Error: [1149](#) SQLSTATE: [42000](#) ([ER_SYNTAX_ERROR](#) [[3989](#)])
Message: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use
- Error: [1150](#) SQLSTATE: [HY000](#) ([ER_DELAYED_CANT_CHANGE_LOCK](#) [[3989](#)])
Message: Delayed insert thread couldn't get requested lock for table %s
- Error: [1151](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_DELAYED_THREADS](#) [[3989](#)])
Message: Too many delayed threads in use
- Error: [1152](#) SQLSTATE: [08S01](#) ([ER_ABORTING_CONNECTION](#) [[3989](#)])
Message: Aborted connection %ld to db: '%s' user: '%s' (%s)
- Error: [1153](#) SQLSTATE: [08S01](#) ([ER_NET_PACKET_TOO_LARGE](#) [[3989](#)])
Message: Got a packet bigger than 'max_allowed_packet' bytes
- Error: [1154](#) SQLSTATE: [08S01](#) ([ER_NET_READ_ERROR_FROM_PIPE](#) [[3989](#)])
Message: Got a read error from the connection pipe
- Error: [1155](#) SQLSTATE: [08S01](#) ([ER_NET_FCNTL_ERROR](#) [[3989](#)])
Message: Got an error from fcntl()
- Error: [1156](#) SQLSTATE: [08S01](#) ([ER_NET_PACKETS_OUT_OF_ORDER](#) [[3989](#)])
Message: Got packets out of order
- Error: [1157](#) SQLSTATE: [08S01](#) ([ER_NET_UNCOMPRESS_ERROR](#) [[3989](#)])
Message: Couldn't uncompress communication packet
- Error: [1158](#) SQLSTATE: [08S01](#) ([ER_NET_READ_ERROR](#) [[3989](#)])

Message: Got an error reading communication packets

- Error: 1159 SQLSTATE: 08S01 ([ER_NET_READ_INTERRUPTED \[3990\]](#))

Message: Got timeout reading communication packets

- Error: 1160 SQLSTATE: 08S01 ([ER_NET_ERROR_ON_WRITE \[3990\]](#))

Message: Got an error writing communication packets

- Error: 1161 SQLSTATE: 08S01 ([ER_NET_WRITE_INTERRUPTED \[3990\]](#))

Message: Got timeout writing communication packets

- Error: 1162 SQLSTATE: 42000 ([ER_TOO_LONG_STRING \[3990\]](#))

Message: Result string is longer than 'max_allowed_packet' bytes

- Error: 1163 SQLSTATE: 42000 ([ER_TABLE_CANT_HANDLE_BLOB \[3990\]](#))

Message: The used table type doesn't support BLOB/TEXT columns

- Error: 1164 SQLSTATE: 42000 ([ER_TABLE_CANT_HANDLE_AUTO_INCREMENT \[3990\]](#))

Message: The used table type doesn't support AUTO_INCREMENT columns

- Error: 1165 SQLSTATE: HY000 ([ER_DELAYED_INSERT_TABLE_LOCKED \[3990\]](#))

Message: INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES

- Error: 1166 SQLSTATE: 42000 ([ER_WRONG_COLUMN_NAME \[3990\]](#))

Message: Incorrect column name '%s'

- Error: 1167 SQLSTATE: 42000 ([ER_WRONG_KEY_COLUMN \[3990\]](#))

Message: The used storage engine can't index column '%s'

- Error: 1168 SQLSTATE: HY000 ([ER_WRONG_MRG_TABLE \[3990\]](#))

Message: Unable to open underlying table which is differently defined or of non-MyISAM type or doesn't exist

- Error: 1169 SQLSTATE: 23000 ([ER_DUP_UNIQUE \[3990\]](#))

Message: Can't write, because of unique constraint, to table '%s'

- Error: 1170 SQLSTATE: 42000 ([ER_BLOB_KEY_WITHOUT_LENGTH \[3990\]](#))

Message: BLOB/TEXT column '%s' used in key specification without a key length

- Error: 1171 SQLSTATE: 42000 ([ER_PRIMARY_CANT_HAVE_NULL \[3990\]](#))

Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead

- Error: 1172 SQLSTATE: 42000 ([ER_TOO_MANY_ROWS \[3990\]](#))

Message: Result consisted of more than one row

- Error: 1173 SQLSTATE: 42000 ([ER_REQUIRES_PRIMARY_KEY \[3991\]](#))
Message: This table type requires a primary key
- Error: 1174 SQLSTATE: HY000 ([ER_NO_RAID_COMPILED \[3991\]](#))
Message: This version of MySQL is not compiled with RAID support
- Error: 1175 SQLSTATE: HY000 ([ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE \[3991\]](#))
Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column
- Error: 1176 SQLSTATE: 42000 ([ER_KEY_DOES_NOT_EXISTS \[3991\]](#))
Message: Key '%s' doesn't exist in table '%s'
- Error: 1177 SQLSTATE: 42000 ([ER_CHECK_NO_SUCH_TABLE \[3991\]](#))
Message: Can't open table
- Error: 1178 SQLSTATE: 42000 ([ER_CHECK_NOT_IMPLEMENTED \[3991\]](#))
Message: The storage engine for the table doesn't support %s
- Error: 1179 SQLSTATE: 25000 ([ER_CANT_DO_THIS_DURING_AN_TRANSACTION \[3991\]](#))
Message: You are not allowed to execute this command in a transaction
- Error: 1180 SQLSTATE: HY000 ([ER_ERROR_DURING_COMMIT \[3991\]](#))
Message: Got error %d during COMMIT
- Error: 1181 SQLSTATE: HY000 ([ER_ERROR_DURING_ROLLBACK \[3991\]](#))
Message: Got error %d during ROLLBACK
- Error: 1182 SQLSTATE: HY000 ([ER_ERROR_DURING_FLUSH_LOGS \[3991\]](#))
Message: Got error %d during FLUSH_LOGS
- Error: 1183 SQLSTATE: HY000 ([ER_ERROR_DURING_CHECKPOINT \[3991\]](#))
Message: Got error %d during CHECKPOINT
- Error: 1184 SQLSTATE: 08S01 ([ER_NEW_ABORTING_CONNECTION \[3991\]](#))
Message: Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s)
- Error: 1185 SQLSTATE: HY000 ([ER_DUMP_NOT_IMPLEMENTED \[3991\]](#))
Message: The storage engine for the table does not support binary table dump
- Error: 1186 SQLSTATE: HY000 ([ER_FLUSH_MASTER_BINLOG_CLOSED \[3991\]](#))
Message: Binlog closed, cannot RESET MASTER
- Error: 1187 SQLSTATE: HY000 ([ER_INDEX_REBUILD \[3991\]](#))
Message: Failed rebuilding the index of dumped table '%s'

- Error: 1188 SQLSTATE: HY000 ([ER_MASTER \[3992\]](#))
Message: Error from master: '%s'
- Error: 1189 SQLSTATE: 08S01 ([ER_MASTER_NET_READ \[3992\]](#))
Message: Net error reading from master
- Error: 1190 SQLSTATE: 08S01 ([ER_MASTER_NET_WRITE \[3992\]](#))
Message: Net error writing to master
- Error: 1191 SQLSTATE: HY000 ([ER_FT_MATCHING_KEY_NOT_FOUND \[3992\]](#))
Message: Can't find FULLTEXT index matching the column list
- Error: 1192 SQLSTATE: HY000 ([ER_LOCK_OR_ACTIVE_TRANSACTION \[3992\]](#))
Message: Can't execute the given command because you have active locked tables or an active transaction
- Error: 1193 SQLSTATE: HY000 ([ER_UNKNOWN_SYSTEM_VARIABLE \[3992\]](#))
Message: Unknown system variable '%s'
- Error: 1194 SQLSTATE: HY000 ([ER_CRASHED_ON_USAGE \[3992\]](#))
Message: Table '%s' is marked as crashed and should be repaired
- Error: 1195 SQLSTATE: HY000 ([ER_CRASHED_ON_REPAIR \[3992\]](#))
Message: Table '%s' is marked as crashed and last (automatic?) repair failed
- Error: 1196 SQLSTATE: HY000 ([ER_WARNING_NOT_COMPLETE_ROLLBACK \[3992\]](#))
Message: Some non-transactional changed tables couldn't be rolled back
- Error: 1197 SQLSTATE: HY000 ([ER_TRANS_CACHE_FULL \[3992\]](#))
Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again
- Error: 1198 SQLSTATE: HY000 ([ER_SLAVE_MUST_STOP \[3992\]](#))
Message: This operation cannot be performed with a running slave; run STOP SLAVE first
- Error: 1199 SQLSTATE: HY000 ([ER_SLAVE_NOT_RUNNING \[3992\]](#))
Message: This operation requires a running slave; configure slave and do START SLAVE
- Error: 1200 SQLSTATE: HY000 ([ER_BAD_SLAVE \[3992\]](#))
Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO
- Error: 1201 SQLSTATE: HY000 ([ER_MASTER_INFO \[3992\]](#))
Message: Could not initialize master info structure; more error messages can be found in the MySQL error log
- Error: 1202 SQLSTATE: HY000 ([ER_SLAVE_THREAD \[3992\]](#))

Message: Could not create slave thread; check system resources

- Error: 1203 SQLSTATE: 42000 ([ER_TOO_MANY_USER_CONNECTIONS \[3993\]](#))

Message: User %s already has more than 'max_user_connections' active connections

- Error: 1204 SQLSTATE: HY000 ([ER_SET_CONSTANTS_ONLY \[3993\]](#))

Message: You may only use constant expressions with SET

- Error: 1205 SQLSTATE: HY000 ([ER_LOCK_WAIT_TIMEOUT \[3993\]](#))

Message: Lock wait timeout exceeded; try restarting transaction

- Error: 1206 SQLSTATE: HY000 ([ER_LOCK_TABLE_FULL \[3993\]](#))

Message: The total number of locks exceeds the lock table size

- Error: 1207 SQLSTATE: 25000 ([ER_READ_ONLY_TRANSACTION \[3993\]](#))

Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error: 1208 SQLSTATE: HY000 ([ER_DROP_DB_WITH_READ_LOCK \[3993\]](#))

Message: DROP DATABASE not allowed while thread is holding global read lock

- Error: 1209 SQLSTATE: HY000 ([ER_CREATE_DB_WITH_READ_LOCK \[3993\]](#))

Message: CREATE DATABASE not allowed while thread is holding global read lock

- Error: 1210 SQLSTATE: HY000 ([ER_WRONG_ARGUMENTS \[3993\]](#))

Message: Incorrect arguments to %s

- Error: 1211 SQLSTATE: 42000 ([ER_NO_PERMISSION_TO_CREATE_USER \[3993\]](#))

Message: '%s'@'%s' is not allowed to create new users

- Error: 1212 SQLSTATE: HY000 ([ER_UNION_TABLES_IN_DIFFERENT_DIR \[3993\]](#))

Message: Incorrect table definition; all MERGE tables must be in the same database

- Error: 1213 SQLSTATE: 40001 ([ER_LOCK_DEADLOCK \[3993\]](#))

Message: Deadlock found when trying to get lock; try restarting transaction

- Error: 1214 SQLSTATE: HY000 ([ER_TABLE_CANT_HANDLE_FT \[3993\]](#))

Message: The used table type doesn't support FULLTEXT indexes

- Error: 1215 SQLSTATE: HY000 ([ER_CANNOT_ADD_FOREIGN \[3993\]](#))

Message: Cannot add foreign key constraint

- Error: 1216 SQLSTATE: 23000 ([ER_NO_REFERENCED_ROW \[3993\]](#))

Message: Cannot add or update a child row: a foreign key constraint fails

- Error: 1217 SQLSTATE: 23000 ([ER_ROW_IS_REFERENCED \[3993\]](#))

Message: Cannot delete or update a parent row: a foreign key constraint fails

- Error: 1218 SQLSTATE: 08S01 ([ER_CONNECT_TO_MASTER \[3994\]](#))

Message: Error connecting to master: %s

- Error: 1219 SQLSTATE: HY000 ([ER_QUERY_ON_MASTER \[3994\]](#))

Message: Error running query on master: %s

- Error: 1220 SQLSTATE: HY000 ([ER_ERROR_WHEN_EXECUTING_COMMAND \[3994\]](#))

Message: Error when executing command %s: %s

- Error: 1221 SQLSTATE: HY000 ([ER_WRONG_USAGE \[3994\]](#))

Message: Incorrect usage of %s and %s

- Error: 1222 SQLSTATE: 21000 ([ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT \[3994\]](#))

Message: The used SELECT statements have a different number of columns

- Error: 1223 SQLSTATE: HY000 ([ER_CANT_UPDATE_WITH_READLOCK \[3994\]](#))

Message: Can't execute the query because you have a conflicting read lock

- Error: 1224 SQLSTATE: HY000 ([ER_MIXING_NOT_ALLOWED \[3994\]](#))

Message: Mixing of transactional and non-transactional tables is disabled

- Error: 1225 SQLSTATE: HY000 ([ER_DUP_ARGUMENT \[3994\]](#))

Message: Option '%s' used twice in statement

- Error: 1226 SQLSTATE: 42000 ([ER_USER_LIMIT_REACHED \[3994\]](#))

Message: User '%s' has exceeded the '%s' resource (current value: %ld)

- Error: 1227 SQLSTATE: 42000 ([ER_SPECIFIC_ACCESS_DENIED_ERROR \[3994\]](#))

Message: Access denied; you need (at least one of) the %s privilege(s) for this operation

- Error: 1228 SQLSTATE: HY000 ([ER_LOCAL_VARIABLE \[3994\]](#))

Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL

- Error: 1229 SQLSTATE: HY000 ([ER_GLOBAL_VARIABLE \[3994\]](#))

Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL

- Error: 1230 SQLSTATE: 42000 ([ER_NO_DEFAULT \[3994\]](#))

Message: Variable '%s' doesn't have a default value

- Error: 1231 SQLSTATE: 42000 ([ER_WRONG_VALUE_FOR_VAR \[3994\]](#))

Message: Variable '%s' can't be set to the value of '%s'

- Error: 1232 SQLSTATE: 42000 ([ER_WRONG_TYPE_FOR_VAR \[3994\]](#))

Message: Incorrect argument type to variable '%s'

- Error: 1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ [3995])

Message: Variable '%s' can only be set, not read

- Error: 1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE [3995])

Message: Incorrect usage/placement of '%s'

- Error: 1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET [3995])

Message: This version of MySQL doesn't yet support '%s'

- Error: 1236 SQLSTATE: HY000 (ER_MASTER_FATAL_ERROR_READING_BINLOG [3995])

Message: Got fatal error %d from master when reading data from binary log: '%s'

- Error: 1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE [3995])

Message: Slave SQL thread ignored the query because of replicate-*table rules

- Error: 1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR [3995])

Message: Variable '%s' is a %s variable

- Error: 1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF [3995])

Message: Incorrect foreign key definition for '%s': %s

- Error: 1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF [3995])

Message: Key reference and table reference don't match

- Error: 1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS [3995])

Message: Operand should contain %d column(s)

- Error: 1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW [3995])

Message: Subquery returns more than 1 row

- Error: 1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER [3995])

Message: Unknown prepared statement handler (%.*s) given to %s

- Error: 1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB [3995])

Message: Help database is corrupt or does not exist

- Error: 1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE [3995])

Message: Cyclic reference on subqueries

- Error: 1246 SQLSTATE: HY000 (ER_AUTO_CONVERT [3995])

Message: Converting column '%s' from %s to %s

- Error: 1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE [3995])

Message: Reference '%s' not supported (%s)

- Error: 1248 SQLSTATE: 42000 ([ER_DERIVED_MUST_HAVE_ALIAS \[3996\]](#))

Message: Every derived table must have its own alias

- Error: 1249 SQLSTATE: 01000 ([ER_SELECT_REDUCED \[3996\]](#))

Message: Select %u was reduced during optimization

- Error: 1250 SQLSTATE: 42000 ([ER_TABLENAME_NOT_ALLOWED_HERE \[3996\]](#))

Message: Table '%s' from one of the SELECTs cannot be used in %s

- Error: 1251 SQLSTATE: 08004 ([ER_NOT_SUPPORTED_AUTH_MODE \[3996\]](#))

Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client

- Error: 1252 SQLSTATE: 42000 ([ER_SPATIAL_CANT_HAVE_NULL \[3996\]](#))

Message: All parts of a SPATIAL index must be NOT NULL

- Error: 1253 SQLSTATE: 42000 ([ER_COLLATION_CHARSET_MISMATCH \[3996\]](#))

Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

- Error: 1254 SQLSTATE: HY000 ([ER_SLAVE_WAS_RUNNING \[3996\]](#))

Message: Slave is already running

- Error: 1255 SQLSTATE: HY000 ([ER_SLAVE_WAS_NOT_RUNNING \[3996\]](#))

Message: Slave already has been stopped

- Error: 1256 SQLSTATE: HY000 ([ER_TOO_BIG_FOR_UNCOMPRESS \[3996\]](#))

Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)

- Error: 1257 SQLSTATE: HY000 ([ER_ZLIB_Z_MEM_ERROR \[3996\]](#))

Message: ZLIB: Not enough memory

- Error: 1258 SQLSTATE: HY000 ([ER_ZLIB_Z_BUF_ERROR \[3996\]](#))

Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error: 1259 SQLSTATE: HY000 ([ER_ZLIB_Z_DATA_ERROR \[3996\]](#))

Message: ZLIB: Input data corrupted

- Error: 1260 SQLSTATE: HY000 ([ER_CUT_VALUE_GROUP_CONCAT \[3996\]](#))

Message: Row %u was cut by GROUP_CONCAT()

- Error: 1261 SQLSTATE: 01000 ([ER_WARN_TOO_FEW_RECORDS \[3996\]](#))

Message: Row %ld doesn't contain data for all columns

- Error: 1262 SQLSTATE: 01000 (ER_WARN_TOO_MANY_RECORDS [3997])

Message: Row %ld was truncated; it contained more data than there were input columns

- Error: 1263 SQLSTATE: 22004 (ER_WARN_NULL_TO_NOTNULL [3997])

Message: Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld

- Error: 1264 SQLSTATE: 22003 (ER_WARN_DATA_OUT_OF_RANGE [3997])

Message: Out of range value for column '%s' at row %ld

- Error: 1265 SQLSTATE: 01000 (WARN_DATA_TRUNCATED [3997])

Message: Data truncated for column '%s' at row %ld

- Error: 1266 SQLSTATE: HY000 (ER_WARN_USING_OTHER_HANDLER [3997])

Message: Using storage engine %s for table '%s'

- Error: 1267 SQLSTATE: HY000 (ER_CANT_AGGREGATE_2COLLATIONS [3997])

Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'

- Error: 1268 SQLSTATE: HY000 (ER_DROP_USER [3997])

Message: Cannot drop one or more of the requested users

- Error: 1269 SQLSTATE: HY000 (ER_REVOKE_GRANTS [3997])

Message: Can't revoke all privileges for one or more of the requested users

- Error: 1270 SQLSTATE: HY000 (ER_CANT_AGGREGATE_3COLLATIONS [3997])

Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'

- Error: 1271 SQLSTATE: HY000 (ER_CANT_AGGREGATE_NCOLLATIONS [3997])

Message: Illegal mix of collations for operation '%s'

- Error: 1272 SQLSTATE: HY000 (ER_VARIABLE_IS_NOT_STRUCT [3997])

Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error: 1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION [3997])

Message: Unknown collation: '%s'

- Error: 1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS [3997])

Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE [3997])

Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format

- Error: 1276 SQLSTATE: HY000 ([ER_WARN_FIELD_RESOLVED \[3998\]](#))
Message: Field or reference '%s%s%s%s%' of SELECT #%d was resolved in SELECT #%d
- Error: 1277 SQLSTATE: HY000 ([ER_BAD_SLAVE_UNTIL_COND \[3998\]](#))
Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL
- Error: 1278 SQLSTATE: HY000 ([ER_MISSING_SKIP_SLAVE \[3998\]](#))
Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart
- Error: 1279 SQLSTATE: HY000 ([ER_UNTIL_COND_IGNORED \[3998\]](#))
Message: SQL thread is not to be started so UNTIL options are ignored
- Error: 1280 SQLSTATE: 42000 ([ER_WRONG_NAME_FOR_INDEX \[3998\]](#))
Message: Incorrect index name '%s'
- Error: 1281 SQLSTATE: 42000 ([ER_WRONG_NAME_FOR_CATALOG \[3998\]](#))
Message: Incorrect catalog name '%s'
- Error: 1282 SQLSTATE: HY000 ([ER_WARN_QC_RESIZE \[3998\]](#))
Message: Query cache failed to set size %lu; new query cache size is %lu
- Error: 1283 SQLSTATE: HY000 ([ER_BAD_FT_COLUMN \[3998\]](#))
Message: Column '%s' cannot be part of FULLTEXT index
- Error: 1284 SQLSTATE: HY000 ([ER_UNKNOWN_KEY_CACHE \[3998\]](#))
Message: Unknown key cache '%s'
- Error: 1285 SQLSTATE: HY000 ([ER_WARN_HOSTNAME_WONT_WORK \[3998\]](#))
Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work
- Error: 1286 SQLSTATE: 42000 ([ER_UNKNOWN_STORAGE_ENGINE \[3998\]](#))
Message: Unknown storage engine '%s'
- Error: 1287 SQLSTATE: HY000 ([ER_WARN_DEPRECATED_SYNTAX \[3998\]](#))
Message: '%s' is deprecated and will be removed in a future release. Please use %s instead
- Error: 1288 SQLSTATE: HY000 ([ER_NON_UPDATABLE_TABLE \[3998\]](#))
Message: The target table %s of the %s is not updatable
- Error: 1289 SQLSTATE: HY000 ([ER_FEATURE_DISABLED \[3998\]](#))
Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working
- Error: 1290 SQLSTATE: HY000 ([ER_OPTION_PREVENTS_STATEMENT \[3998\]](#))

Message: The MySQL server is running with the %s option so it cannot execute this statement

- Error: 1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE [3999])

Message: Column '%s' has duplicated value '%s' in %s

- Error: 1292 SQLSTATE: 22007 (ER_TRUNCATED_WRONG_VALUE [3999])

Message: Truncated incorrect %s value: '%s'

- Error: 1293 SQLSTATE: HY000 (ER_TOO_MUCH_AUTO_TIMESTAMP_COLS [3999])

Message: Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

- Error: 1294 SQLSTATE: HY000 (ER_INVALID_ON_UPDATE [3999])

Message: Invalid ON UPDATE clause for '%s' column

- Error: 1295 SQLSTATE: HY000 (ER_UNSUPPORTED_PS [3999])

Message: This command is not supported in the prepared statement protocol yet

- Error: 1296 SQLSTATE: HY000 (ER_GET_ERRMSG [3999])

Message: Got error %d '%s' from %s

- Error: 1297 SQLSTATE: HY000 (ER_GET_TEMPORARY_ERRMSG [3999])

Message: Got temporary error %d '%s' from %s

- Error: 1298 SQLSTATE: HY000 (ER_UNKNOWN_TIME_ZONE [3999])

Message: Unknown or incorrect time zone: '%s'

- Error: 1299 SQLSTATE: HY000 (ER_WARN_INVALID_TIMESTAMP [3999])

Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: 1300 SQLSTATE: HY000 (ER_INVALID_CHARACTER_STRING [3999])

Message: Invalid %s character string: '%s'

- Error: 1301 SQLSTATE: HY000 (ER_WARN_ALLOWED_PACKET_OVERFLOWED [3999])

Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: 1302 SQLSTATE: HY000 (ER_CONFLICTING_DECLARATIONS [3999])

Message: Conflicting declarations: '%s%s' and '%s%s'

- Error: 1303 SQLSTATE: 2F003 (ER_SP_NO_RECURSIVE_CREATE [3999])

Message: Can't create a %s from within another stored routine

- Error: 1304 SQLSTATE: 42000 (ER_SP_ALREADY_EXISTS [3999])

Message: %s %s already exists

- Error: 1305 SQLSTATE: 42000 ([ER_SP_DOES_NOT_EXIST \[4000\]](#))
Message: %s %s does not exist
- Error: 1306 SQLSTATE: HY000 ([ER_SP_DROP_FAILED \[4000\]](#))
Message: Failed to DROP %s %s
- Error: 1307 SQLSTATE: HY000 ([ER_SP_STORE_FAILED \[4000\]](#))
Message: Failed to CREATE %s %s
- Error: 1308 SQLSTATE: 42000 ([ER_SP_LILABEL_MISMATCH \[4000\]](#))
Message: %s with no matching label: %s
- Error: 1309 SQLSTATE: 42000 ([ER_SP_LABEL_REDEFINE \[4000\]](#))
Message: Redefining label %s
- Error: 1310 SQLSTATE: 42000 ([ER_SP_LABEL_MISMATCH \[4000\]](#))
Message: End-label %s without match
- Error: 1311 SQLSTATE: 01000 ([ER_SP_UNINIT_VAR \[4000\]](#))
Message: Referring to uninitialized variable %s
- Error: 1312 SQLSTATE: 0A000 ([ER_SP_BADSELECT \[4000\]](#))
Message: PROCEDURE %s can't return a result set in the given context
- Error: 1313 SQLSTATE: 42000 ([ER_SP_BADRETURN \[4000\]](#))
Message: RETURN is only allowed in a FUNCTION
- Error: 1314 SQLSTATE: 0A000 ([ER_SP_BADSTATEMENT \[4000\]](#))
Message: %s is not allowed in stored procedures
- Error: 1315 SQLSTATE: 42000 ([ER_UPDATE_LOG_DEPRECATED_IGNORED \[4000\]](#))
Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored. This option will be removed in MySQL 5.6.
- Error: 1316 SQLSTATE: 42000 ([ER_UPDATE_LOG_DEPRECATED_TRANSLATED \[4000\]](#))
Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN. This option will be removed in MySQL 5.6.
- Error: 1317 SQLSTATE: 70100 ([ER_QUERY_INTERRUPTED \[4000\]](#))
Message: Query execution was interrupted
- Error: 1318 SQLSTATE: 42000 ([ER_SP_WRONG_NO_OF_ARGS \[4000\]](#))
Message: Incorrect number of arguments for %s %s; expected %u, got %u
- Error: 1319 SQLSTATE: 42000 ([ER_SP_COND_MISMATCH \[4000\]](#))

Message: Undefined CONDITION: %s

- Error: 1320 SQLSTATE: 42000 (ER_SP_NORETURN [4001])

Message: No RETURN found in FUNCTION %s

- Error: 1321 SQLSTATE: 2F005 (ER_SP_NORETURNEND [4001])

Message: FUNCTION %s ended without RETURN

- Error: 1322 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_QUERY [4001])

Message: Cursor statement must be a SELECT

- Error: 1323 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_SELECT [4001])

Message: Cursor SELECT must not have INTO

- Error: 1324 SQLSTATE: 42000 (ER_SP_CURSOR_MISMATCH [4001])

Message: Undefined CURSOR: %s

- Error: 1325 SQLSTATE: 24000 (ER_SP_CURSOR_ALREADY_OPEN [4001])

Message: Cursor is already open

- Error: 1326 SQLSTATE: 24000 (ER_SP_CURSOR_NOT_OPEN [4001])

Message: Cursor is not open

- Error: 1327 SQLSTATE: 42000 (ER_SP_UNDECLARED_VAR [4001])

Message: Undeclared variable: %s

- Error: 1328 SQLSTATE: HY000 (ER_SP_WRONG_NO_OF_FETCH_ARGS [4001])

Message: Incorrect number of FETCH variables

- Error: 1329 SQLSTATE: 02000 (ER_SP_FETCH_NO_DATA [4001])

Message: No data - zero rows fetched, selected, or processed

- Error: 1330 SQLSTATE: 42000 (ER_SP_DUP_PARAM [4001])

Message: Duplicate parameter: %s

- Error: 1331 SQLSTATE: 42000 (ER_SP_DUP_VAR [4001])

Message: Duplicate variable: %s

- Error: 1332 SQLSTATE: 42000 (ER_SP_DUP_COND [4001])

Message: Duplicate condition: %s

- Error: 1333 SQLSTATE: 42000 (ER_SP_DUP_CURS [4001])

Message: Duplicate cursor: %s

- Error: 1334 SQLSTATE: HY000 (ER_SP_CANT_ALTER [4001])

Message: Failed to ALTER %s %s

- Error: 1335 SQLSTATE: 0A000 ([ER_SP_SUBSELECT_NYI \[4002\]](#))

Message: Subquery value not supported

- Error: 1336 SQLSTATE: 0A000 ([ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG \[4002\]](#))

Message: %s is not allowed in stored function or trigger

- Error: 1337 SQLSTATE: 42000 ([ER_SP_VARCOND_AFTER_CURSHNDLR \[4002\]](#))

Message: Variable or condition declaration after cursor or handler declaration

- Error: 1338 SQLSTATE: 42000 ([ER_SP_CURSOR_AFTER_HANDLER \[4002\]](#))

Message: Cursor declaration after handler declaration

- Error: 1339 SQLSTATE: 20000 ([ER_SP_CASE_NOT_FOUND \[4002\]](#))

Message: Case not found for CASE statement

- Error: 1340 SQLSTATE: HY000 ([ER_FPARSER_TOO_BIG_FILE \[4002\]](#))

Message: Configuration file '%s' is too big

- Error: 1341 SQLSTATE: HY000 ([ER_FPARSER_BAD_HEADER \[4002\]](#))

Message: Malformed file type header in file '%s'

- Error: 1342 SQLSTATE: HY000 ([ER_FPARSER_EOF_IN_COMMENT \[4002\]](#))

Message: Unexpected end of file while parsing comment '%s'

- Error: 1343 SQLSTATE: HY000 ([ER_FPARSER_ERROR_IN_PARAMETER \[4002\]](#))

Message: Error while parsing parameter '%s' (line: '%s')

- Error: 1344 SQLSTATE: HY000 ([ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER \[4002\]](#))

Message: Unexpected end of file while skipping unknown parameter '%s'

- Error: 1345 SQLSTATE: HY000 ([ER_VIEW_NO_EXPLAIN \[4002\]](#))

Message: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table

- Error: 1346 SQLSTATE: HY000 ([ER_FRM_UNKNOWN_TYPE \[4002\]](#))

Message: File '%s' has unknown type '%s' in its header

- Error: 1347 SQLSTATE: HY000 ([ER_WRONG_OBJECT \[4002\]](#))

Message: '%s.%s' is not %s

- Error: 1348 SQLSTATE: HY000 ([ER_NONUPDATEABLE_COLUMN \[4002\]](#))

Message: Column '%s' is not updatable

- Error: 1349 SQLSTATE: HY000 ([ER_VIEW_SELECT_DERIVED \[4002\]](#))

Message: View's SELECT contains a subquery in the FROM clause

- Error: 1350 SQLSTATE: HY000 (ER_VIEW_SELECT_CLAUSE [4003])

Message: View's SELECT contains a '%s' clause

- Error: 1351 SQLSTATE: HY000 (ER_VIEW_SELECT_VARIABLE [4003])

Message: View's SELECT contains a variable or parameter

- Error: 1352 SQLSTATE: HY000 (ER_VIEW_SELECT_TMPTABLE [4003])

Message: View's SELECT refers to a temporary table '%s'

- Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST [4003])

Message: View's SELECT and view's field list have different column counts

- Error: 1354 SQLSTATE: HY000 (ER_WARN_VIEW_MERGE [4003])

Message: View merge algorithm can't be used here for now (assumed undefined algorithm)

- Error: 1355 SQLSTATE: HY000 (ER_WARN_VIEW_WITHOUT_KEY [4003])

Message: View being updated does not have complete key of underlying table in it

- Error: 1356 SQLSTATE: HY000 (ER_VIEW_INVALID [4003])

Message: View '%s.%s' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them

- Error: 1357 SQLSTATE: HY000 (ER_SP_NO_DROP_SP [4003])

Message: Can't drop or alter a %s from within another stored routine

- Error: 1358 SQLSTATE: HY000 (ER_SP_GOTO_IN_HNDLR [4003])

Message: GOTO is not allowed in a stored procedure handler

- Error: 1359 SQLSTATE: HY000 (ER_TRG_ALREADY_EXISTS [4003])

Message: Trigger already exists

- Error: 1360 SQLSTATE: HY000 (ER_TRG_DOES_NOT_EXIST [4003])

Message: Trigger does not exist

- Error: 1361 SQLSTATE: HY000 (ER_TRG_ON_VIEW_OR_TEMP_TABLE [4003])

Message: Trigger's '%s' is view or temporary table

- Error: 1362 SQLSTATE: HY000 (ER_TRG_CANT_CHANGE_ROW [4003])

Message: Updating of %s row is not allowed in %strigger

- Error: 1363 SQLSTATE: HY000 (ER_TRG_NO_SUCH_ROW_IN_TRG [4003])

Message: There is no %s row in %s trigger

- Error: 1364 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_FIELD [4004])
Message: Field '%s' doesn't have a default value
- Error: 1365 SQLSTATE: 22012 (ER_DIVISION_BY_ZERO [4004])
Message: Division by 0
- Error: 1366 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE_FOR_FIELD [4004])
Message: Incorrect %s value: '%s' for column '%s' at row %ld
- Error: 1367 SQLSTATE: 22007 (ER_ILLEGAL_VALUE_FOR_TYPE [4004])
Message: Illegal %s '%s' value found during parsing
- Error: 1368 SQLSTATE: HY000 (ER_VIEW_NONUPD_CHECK [4004])
Message: CHECK OPTION on non-updatable view '%s.%s'
- Error: 1369 SQLSTATE: HY000 (ER_VIEW_CHECK_FAILED [4004])
Message: CHECK OPTION failed '%s.%s'
- Error: 1370 SQLSTATE: 42000 (ER_PROCACCESS_DENIED_ERROR [4004])
Message: %s command denied to user '%s'@'%s' for routine '%s'
- Error: 1371 SQLSTATE: HY000 (ER_RELAY_LOG_FAIL [4004])
Message: Failed purging old relay logs: %s
- Error: 1372 SQLSTATE: HY000 (ER_PASSWD_LENGTH [4004])
Message: Password hash should be a %d-digit hexadecimal number
- Error: 1373 SQLSTATE: HY000 (ER_UNKNOWN_TARGET_BINLOG [4004])
Message: Target log not found in binlog index
- Error: 1374 SQLSTATE: HY000 (ER_IO_ERR_LOG_INDEX_READ [4004])
Message: I/O error reading log index file
- Error: 1375 SQLSTATE: HY000 (ER_BINLOG_PURGE_PROHIBITED [4004])
Message: Server configuration does not permit binlog purge
- Error: 1376 SQLSTATE: HY000 (ER_FSEEK_FAIL [4004])
Message: Failed on fseek()
- Error: 1377 SQLSTATE: HY000 (ER_BINLOG_PURGE_FATAL_ERR [4004])
Message: Fatal error during log purge
- Error: 1378 SQLSTATE: HY000 (ER_LOG_IN_USE [4004])
Message: A purgeable log is in use, will not purge

- Error: [1379](#) SQLSTATE: [HY000](#) ([ER_LOG_PURGE_UNKNOWN_ERR \[4005\]](#))
Message: Unknown error during log purge
- Error: [1380](#) SQLSTATE: [HY000](#) ([ER_RELAY_LOG_INIT \[4005\]](#))
Message: Failed initializing relay log position: %s
- Error: [1381](#) SQLSTATE: [HY000](#) ([ER_NO_BINARY_LOGGING \[4005\]](#))
Message: You are not using binary logging
- Error: [1382](#) SQLSTATE: [HY000](#) ([ER_RESERVED_SYNTAX \[4005\]](#))
Message: The '%s' syntax is reserved for purposes internal to the MySQL server
- Error: [1383](#) SQLSTATE: [HY000](#) ([ER_WSAS_FAILED \[4005\]](#))
Message: WSASStartup Failed
- Error: [1384](#) SQLSTATE: [HY000](#) ([ER_DIFF_GROUPS_PROC \[4005\]](#))
Message: Can't handle procedures with different groups yet
- Error: [1385](#) SQLSTATE: [HY000](#) ([ER_NO_GROUP_FOR_PROC \[4005\]](#))
Message: Select must have a group with this procedure
- Error: [1386](#) SQLSTATE: [HY000](#) ([ER_ORDER_WITH_PROC \[4005\]](#))
Message: Can't use ORDER clause with this procedure
- Error: [1387](#) SQLSTATE: [HY000](#) ([ER_LOGGING_PROHIBIT_CHANGING_OF \[4005\]](#))
Message: Binary logging and replication forbid changing the global server %s
- Error: [1388](#) SQLSTATE: [HY000](#) ([ER_NO_FILE_MAPPING \[4005\]](#))
Message: Can't map file: %s, errno: %d
- Error: [1389](#) SQLSTATE: [HY000](#) ([ER_WRONG_MAGIC \[4005\]](#))
Message: Wrong magic in %s
- Error: [1390](#) SQLSTATE: [HY000](#) ([ER_PS_MANY_PARAM \[4005\]](#))
Message: Prepared statement contains too many placeholders
- Error: [1391](#) SQLSTATE: [HY000](#) ([ER_KEY_PART_0 \[4005\]](#))
Message: Key part '%s' length cannot be 0
- Error: [1392](#) SQLSTATE: [HY000](#) ([ER_VIEW_CHECKSUM \[4005\]](#))
Message: View text checksum failed
- Error: [1393](#) SQLSTATE: [HY000](#) ([ER_VIEW_MULTIUPDATE \[4005\]](#))
Message: Can not modify more than one base table through a join view '%s.%s'

- Error: 1394 SQLSTATE: HY000 ([ER_VIEW_NO_INSERT_FIELD_LIST \[4006\]](#))
Message: Can not insert into join view '%s.%s' without fields list
- Error: 1395 SQLSTATE: HY000 ([ER_VIEW_DELETE_MERGE_VIEW \[4006\]](#))
Message: Can not delete from join view '%s.%s'
- Error: 1396 SQLSTATE: HY000 ([ER_CANNOT_USER \[4006\]](#))
Message: Operation %s failed for %s
- Error: 1397 SQLSTATE: XAE04 ([ER_XAER_NOTA \[4006\]](#))
Message: XAER_NOTA: Unknown XID
- Error: 1398 SQLSTATE: XAE05 ([ER_XAER_INVAL \[4006\]](#))
Message: XAER_INVAL: Invalid arguments (or unsupported command)
- Error: 1399 SQLSTATE: XAE07 ([ER_XAER_RMFAIL \[4006\]](#))
Message: XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state
- Error: 1400 SQLSTATE: XAE09 ([ER_XAER_OUTSIDE \[4006\]](#))
Message: XAER_OUTSIDE: Some work is done outside global transaction
- Error: 1401 SQLSTATE: XAE03 ([ER_XAER_RMERR \[4006\]](#))
Message: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency
- Error: 1402 SQLSTATE: XA100 ([ER_XA_RBROLLBACK \[4006\]](#))
Message: XA_RBROLLBACK: Transaction branch was rolled back
- Error: 1403 SQLSTATE: 42000 ([ER_NONEXISTING_PROC_GRANT \[4006\]](#))
Message: There is no such grant defined for user '%s' on host '%s' on routine '%s'
- Error: 1404 SQLSTATE: HY000 ([ER_PROC_AUTO_GRANT_FAIL \[4006\]](#))
Message: Failed to grant EXECUTE and ALTER ROUTINE privileges
- Error: 1405 SQLSTATE: HY000 ([ER_PROC_AUTO_REVOKE_FAIL \[4006\]](#))
Message: Failed to revoke all privileges to dropped routine
- Error: 1406 SQLSTATE: 22001 ([ER_DATA_TOO_LONG \[4006\]](#))
Message: Data too long for column '%s' at row %ld
- Error: 1407 SQLSTATE: 42000 ([ER_SP_BAD_SQLSTATE \[4006\]](#))
Message: Bad SQLSTATE: '%s'
- Error: 1408 SQLSTATE: HY000 ([ER_STARTUP \[4006\]](#))
Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s

- Error: 1409 SQLSTATE: HY000 (ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR [4007])
Message: Can't load value from file with fixed size rows to variable
- Error: 1410 SQLSTATE: 42000 (ER_CANT_CREATE_USER_WITH_GRANT [4007])
Message: You are not allowed to create a user with GRANT
- Error: 1411 SQLSTATE: HY000 (ER_WRONG_VALUE_FOR_TYPE [4007])
Message: Incorrect %s value: '%s' for function %s
- Error: 1412 SQLSTATE: HY000 (ER_TABLE_DEF_CHANGED [4007])
Message: Table definition has changed, please retry transaction
- Error: 1413 SQLSTATE: 42000 (ER_SP_DUP_HANDLER [4007])
Message: Duplicate handler declared in the same block
- Error: 1414 SQLSTATE: 42000 (ER_SP_NOT_VAR_ARG [4007])
Message: OUT or INOUT argument %d for routine %s is not a variable or NEW pseudo-variable in BEFORE trigger
- Error: 1415 SQLSTATE: 0A000 (ER_SP_NO_RESET [4007])
Message: Not allowed to return a result set from a %s
- Error: 1416 SQLSTATE: 22003 (ER_CANT_CREATE_GEOMETRY_OBJECT [4007])
Message: Cannot get geometry object from data you send to the GEOMETRY field
- Error: 1417 SQLSTATE: HY000 (ER_FAILED_ROUTINE_BREAK_BINLOG [4007])
Message: A routine failed and has neither NO SQL nor READS SQL DATA in its declaration and binary logging is enabled; if non-transactional tables were updated, the binary log will miss their changes
- Error: 1418 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_ROUTINE [4007])
Message: This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
- Error: 1419 SQLSTATE: HY000 (ER_BINLOG_CREATE_ROUTINE_NEED_SUPER [4007])
Message: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
- Error: 1420 SQLSTATE: HY000 (ER_EXEC_STMT_WITH_OPEN_CURSOR [4007])
Message: You can't execute a prepared statement which has an open cursor associated with it. Reset the statement to re-execute it.
- Error: 1421 SQLSTATE: HY000 (ER_STMT_HAS_NO_OPEN_CURSOR [4007])
Message: The statement (%lu) has no open cursor.
- Error: 1422 SQLSTATE: HY000 (ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG [4007])

Message: Explicit or implicit commit is not allowed in stored function or trigger.

- Error: 1423 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_VIEW_FIELD [4008])

Message: Field of view '%s.%s' underlying table doesn't have a default value

- Error: 1424 SQLSTATE: HY000 (ER_SP_NO_RECURSION [4008])

Message: Recursive stored functions and triggers are not allowed.

- Error: 1425 SQLSTATE: 42000 (ER_TOO_BIG_SCALE [4008])

Message: Too big scale %d specified for column '%s'. Maximum is %lu.

- Error: 1426 SQLSTATE: 42000 (ER_TOO_BIG_PRECISION [4008])

Message: Too big precision %d specified for column '%s'. Maximum is %lu.

- Error: 1427 SQLSTATE: 42000 (ER_M_BIGGER_THAN_D [4008])

Message: For float(M,D), double(M,D) or decimal(M,D), M must be >= D (column '%s').

- Error: 1428 SQLSTATE: HY000 (ER_WRONG_LOCK_OF_SYSTEM_TABLE [4008])

Message: You can't combine write-locking of system tables with other tables or lock types

- Error: 1429 SQLSTATE: HY000 (ER_CONNECT_TO_FOREIGN_DATA_SOURCE [4008])

Message: Unable to connect to foreign data source: %s

- Error: 1430 SQLSTATE: HY000 (ER_QUERY_ON_FOREIGN_DATA_SOURCE [4008])

Message: There was a problem processing the query on the foreign data source. Data source error: %s

- Error: 1431 SQLSTATE: HY000 (ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST [4008])

Message: The foreign data source you are trying to reference does not exist. Data source error: %s

- Error: 1432 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE [4008])

Message: Can't create federated table. The data source connection string '%s' is not in the correct format

- Error: 1433 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID [4008])

Message: The data source connection string '%s' is not in the correct format

- Error: 1434 SQLSTATE: HY000 (ER_CANT_CREATE_FEDERATED_TABLE [4008])

Message: Can't create federated table. Foreign data src error: %s

- Error: 1435 SQLSTATE: HY000 (ER_TRG_IN_WRONG_SCHEMA [4008])

Message: Trigger in wrong schema

- Error: 1436 SQLSTATE: HY000 (ER_STACK_OVERRUN_NEED_MORE [4008])

Message: Thread stack overrun: %ld bytes used of a %ld byte stack, and %ld bytes needed. Use 'mysqld --thread_stack=#' to specify a bigger stack.

- Error: 1437 SQLSTATE: 42000 (ER_TOO_LONG_BODY [4009])
Message: Routine body for '%s' is too long
- Error: 1438 SQLSTATE: HY000 (ER_WARN_CANT_DROP_DEFAULT_KEYCACHE [4009])
Message: Cannot drop default keycache
- Error: 1439 SQLSTATE: 42000 (ER_TOO_BIG_DISPLAYWIDTH [4009])
Message: Display width out of range for column '%s' (max = %lu)
- Error: 1440 SQLSTATE: XAE08 (ER_XAER_DUPID [4009])
Message: XAER_DUPID: The XID already exists
- Error: 1441 SQLSTATE: 22008 (ER_DATETIME_FUNCTION_OVERFLOW [4009])
Message: Datetime function: %s field overflow
- Error: 1442 SQLSTATE: HY000 (ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG [4009])
Message: Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.
- Error: 1443 SQLSTATE: HY000 (ER_VIEW_PREVENT_UPDATE [4009])
Message: The definition of table '%s' prevents operation %s on table '%s'.
- Error: 1444 SQLSTATE: HY000 (ER_PS_NO_RECURSION [4009])
Message: The prepared statement contains a stored routine call that refers to that same statement. It's not allowed to execute a prepared statement in such a recursive manner
- Error: 1445 SQLSTATE: HY000 (ER_SP_CANT_SET_AUTOCOMMIT [4009])
Message: Not allowed to set autocommit from a stored function or trigger
- Error: 1446 SQLSTATE: HY000 (ER_MALFORMED_DEFINER [4009])
Message: Definer is not fully qualified
- Error: 1447 SQLSTATE: HY000 (ER_VIEW_FRM_NO_USER [4009])
Message: View '%s'. '%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!
- Error: 1448 SQLSTATE: HY000 (ER_VIEW_OTHER_USER [4009])
Message: You need the SUPER privilege for creation view with '%s'@'%s' definer
- Error: 1449 SQLSTATE: HY000 (ER_NO_SUCH_USER [4009])
Message: The user specified as a definer ('%s'@'%s') does not exist
- Error: 1450 SQLSTATE: HY000 (ER_FORBID_SCHEMA_CHANGE [4009])
Message: Changing schema from '%s' to '%s' is not allowed.
- Error: 1451 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED_2 [4009])

Message: Cannot delete or update a parent row: a foreign key constraint fails (%s)

- Error: 1452 SQLSTATE: 23000 ([ER_NO_REFERENCED_ROW_2 \[4010\]](#))

Message: Cannot add or update a child row: a foreign key constraint fails (%s)

- Error: 1453 SQLSTATE: 42000 ([ER_SP_BAD_VAR_SHADOW \[4010\]](#))

Message: Variable '%s' must be quoted with `...`, or renamed

- Error: 1454 SQLSTATE: HY000 ([ER_TRG_NO_DEFINER \[4010\]](#))

Message: No definer attribute for trigger '%s'.%s'. The trigger will be activated under the authorization of the invoker, which may have insufficient privileges. Please recreate the trigger.

- Error: 1455 SQLSTATE: HY000 ([ER_OLD_FILE_FORMAT \[4010\]](#))

Message: '%s' has an old format, you should re-create the '%s' object(s)

- Error: 1456 SQLSTATE: HY000 ([ER_SP_RECURSION_LIMIT \[4010\]](#))

Message: Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s

- Error: 1457 SQLSTATE: HY000 ([ER_SP_PROC_TABLE_CORRUPT \[4010\]](#))

Message: Failed to load routine %s. The table mysql.proc is missing, corrupt, or contains bad data (internal code %d)

- Error: 1458 SQLSTATE: 42000 ([ER_SP_WRONG_NAME \[4010\]](#))

Message: Incorrect routine name '%s'

- Error: 1459 SQLSTATE: HY000 ([ER_TABLE_NEEDS_UPGRADE \[4010\]](#))

Message: Table upgrade required. Please do "REPAIR TABLE `s`" or dump/reload to fix it!

- Error: 1460 SQLSTATE: 42000 ([ER_SP_NO_AGGREGATE \[4010\]](#))

Message: AGGREGATE is not supported for stored functions

- Error: 1461 SQLSTATE: 42000 ([ER_MAX_PREPARED_STMT_COUNT_REACHED \[4010\]](#))

Message: Can't create more than max_prepared_stmt_count statements (current value: %lu)

- Error: 1462 SQLSTATE: HY000 ([ER_VIEW_RECURSIVE \[4010\]](#))

Message: `s`.`s` contains view recursion

- Error: 1463 SQLSTATE: 42000 ([ER_NON_GROUPING_FIELD_USED \[4010\]](#))

Message: Non-grouping field '%s' is used in %s clause

- Error: 1464 SQLSTATE: HY000 ([ER_TABLE_CANT_HANDLE_SPKEYS \[4010\]](#))

Message: The used table type doesn't support SPATIAL indexes

- Error: 1465 SQLSTATE: HY000 ([ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA \[4010\]](#))

Message: Triggers can not be created on system tables

- Error: 1466 SQLSTATE: HY000 (ER_REMOVED_SPACES [4011])

Message: Leading spaces are removed from name '%s'

- Error: 1467 SQLSTATE: HY000 (ER_AUTOINC_READ_FAILED [4011])

Message: Failed to read auto-increment value from storage engine

- Error: 1468 SQLSTATE: HY000 (ER_USERNAME [4011])

Message: user name

- Error: 1469 SQLSTATE: HY000 (ER_HOSTNAME [4011])

Message: host name

- Error: 1470 SQLSTATE: HY000 (ER_WRONG_STRING_LENGTH [4011])

Message: String '%s' is too long for %s (should be no longer than %d)

- Error: 1471 SQLSTATE: HY000 (ER_NON_INSERTABLE_TABLE [4011])

Message: The target table %s of the %s is not insertable-into

- Error: 1472 SQLSTATE: HY000 (ER_ADMIN_WRONG_MRG_TABLE [4011])

Message: Table '%s' is differently defined or of non-MyISAM type or doesn't exist

- Error: 1473 SQLSTATE: HY000 (ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT [4011])

Message: Too high level of nesting for select

- Error: 1474 SQLSTATE: HY000 (ER_NAME_BECOMES_EMPTY [4011])

Message: Name '%s' has become "

- Error: 1475 SQLSTATE: HY000 (ER_AMBIGUOUS_FIELD_TERM [4011])

Message: First character of the FIELDS TERMINATED string is ambiguous; please use non-optional and non-empty FIELDS ENCLOSED BY

- Error: 1476 SQLSTATE: HY000 (ER_FOREIGN_SERVER_EXISTS [4011])

Message: The foreign server, %s, you are trying to create already exists.

- Error: 1477 SQLSTATE: HY000 (ER_FOREIGN_SERVER_DOESNT_EXIST [4011])

Message: The foreign server name you are trying to reference does not exist. Data source error: %s

- Error: 1478 SQLSTATE: HY000 (ER_ILLEGAL_HA_CREATE_OPTION [4011])

Message: Table storage engine '%s' does not support the create option '%s'

- Error: 1479 SQLSTATE: HY000 (ER_PARTITION_REQUIRES_VALUES_ERROR [4011])

Message: Syntax error: %s PARTITIONING requires definition of VALUES %s for each partition

- Error: 1480 SQLSTATE: HY000 ([ER_PARTITION_WRONG_VALUES_ERROR \[4012\]](#))
Message: Only %s PARTITIONING can use VALUES %s in partition definition
- Error: 1481 SQLSTATE: HY000 ([ER_PARTITION_MAXVALUE_ERROR \[4012\]](#))
Message: MAXVALUE can only be used in last partition definition
- Error: 1482 SQLSTATE: HY000 ([ER_PARTITION_SUBPARTITION_ERROR \[4012\]](#))
Message: Subpartitions can only be hash partitions and by key
- Error: 1483 SQLSTATE: HY000 ([ER_PARTITION_SUBPART_MIX_ERROR \[4012\]](#))
Message: Must define subpartitions on all partitions if on one partition
- Error: 1484 SQLSTATE: HY000 ([ER_PARTITION_WRONG_NO_PART_ERROR \[4012\]](#))
Message: Wrong number of partitions defined, mismatch with previous setting
- Error: 1485 SQLSTATE: HY000 ([ER_PARTITION_WRONG_NO_SUBPART_ERROR \[4012\]](#))
Message: Wrong number of subpartitions defined, mismatch with previous setting
- Error: 1486 SQLSTATE: HY000 ([ER_WRONG_EXPR_IN_PARTITION_FUNC_ERROR \[4012\]](#))
Message: Constant, random or timezone-dependent expressions in (sub)partitioning function are not allowed
- Error: 1487 SQLSTATE: HY000 ([ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR \[4012\]](#))
Message: Expression in RANGE/LIST VALUES must be constant
- Error: 1488 SQLSTATE: HY000 ([ER_FIELD_NOT_FOUND_PART_ERROR \[4012\]](#))
Message: Field in list of fields for partition function not found in table
- Error: 1489 SQLSTATE: HY000 ([ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR \[4012\]](#))
Message: List of fields is only allowed in KEY partitions
- Error: 1490 SQLSTATE: HY000 ([ER_INCONSISTENT_PARTITION_INFO_ERROR \[4012\]](#))
Message: The partition info in the frm file is not consistent with what can be written into the frm file
- Error: 1491 SQLSTATE: HY000 ([ER_PARTITION_FUNC_NOT_ALLOWED_ERROR \[4012\]](#))
Message: The %s function returns the wrong type
- Error: 1492 SQLSTATE: HY000 ([ER_PARTITIONS_MUST_BE_DEFINED_ERROR \[4012\]](#))
Message: For %s partitions each partition must be defined
- Error: 1493 SQLSTATE: HY000 ([ER_RANGE_NOT_INCREASING_ERROR \[4012\]](#))
Message: VALUES LESS THAN value must be strictly increasing for each partition
- Error: 1494 SQLSTATE: HY000 ([ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR \[4012\]](#))
Message: VALUES value must be of same type as partition function

- Error: 1495 SQLSTATE: HY000 ([ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR \[4013\]](#))
Message: Multiple definition of same constant in list partitioning
- Error: 1496 SQLSTATE: HY000 ([ER_PARTITION_ENTRY_ERROR \[4013\]](#))
Message: Partitioning can not be used stand-alone in query
- Error: 1497 SQLSTATE: HY000 ([ER_MIX_HANDLER_ERROR \[4013\]](#))
Message: The mix of handlers in the partitions is not allowed in this version of MySQL
- Error: 1498 SQLSTATE: HY000 ([ER_PARTITION_NOT_DEFINED_ERROR \[4013\]](#))
Message: For the partitioned engine it is necessary to define all %s
- Error: 1499 SQLSTATE: HY000 ([ER_TOO_MANY_PARTITIONS_ERROR \[4013\]](#))
Message: Too many partitions (including subpartitions) were defined
- Error: 1500 SQLSTATE: HY000 ([ER_SUBPARTITION_ERROR \[4013\]](#))
Message: It is only possible to mix RANGE/LIST partitioning with HASH/KEY partitioning for subpartitioning
- Error: 1501 SQLSTATE: HY000 ([ER_CANT_CREATE_HANDLER_FILE \[4013\]](#))
Message: Failed to create specific handler file
- Error: 1502 SQLSTATE: HY000 ([ER_BLOB_FIELD_IN_PART_FUNC_ERROR \[4013\]](#))
Message: A BLOB field is not allowed in partition function
- Error: 1503 SQLSTATE: HY000 ([ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF \[4013\]](#))
Message: A %s must include all columns in the table's partitioning function
- Error: 1504 SQLSTATE: HY000 ([ER_NO_PARTS_ERROR \[4013\]](#))
Message: Number of %s = 0 is not an allowed value
- Error: 1505 SQLSTATE: HY000 ([ER_PARTITION_MGMT_ON_NONPARTITIONED \[4013\]](#))
Message: Partition management on a not partitioned table is not possible
- Error: 1506 SQLSTATE: HY000 ([ER_FOREIGN_KEY_ON_PARTITIONED \[4013\]](#))
Message: Foreign key clause is not yet supported in conjunction with partitioning
- Error: 1507 SQLSTATE: HY000 ([ER_DROP_PARTITION_NON_EXISTENT \[4013\]](#))
Message: Error in list of partitions to %s
- Error: 1508 SQLSTATE: HY000 ([ER_DROP_LAST_PARTITION \[4013\]](#))
Message: Cannot remove all partitions, use DROP TABLE instead
- Error: 1509 SQLSTATE: HY000 ([ER_COALESCE_ONLY_ON_HASH_PARTITION \[4013\]](#))
Message: COALESCE PARTITION can only be used on HASH/KEY partitions

- Error: 1510 SQLSTATE: HY000 (ER_REORG_HASH_ONLY_ON_SAME_NO [4014])
Message: REORGANIZE PARTITION can only be used to reorganize partitions not to change their numbers
- Error: 1511 SQLSTATE: HY000 (ER_REORG_NO_PARAM_ERROR [4014])
Message: REORGANIZE PARTITION without parameters can only be used on auto-partitioned tables using HASH PARTITIONS
- Error: 1512 SQLSTATE: HY000 (ER_ONLY_ON_RANGE_LIST_PARTITION [4014])
Message: %s PARTITION can only be used on RANGE/LIST partitions
- Error: 1513 SQLSTATE: HY000 (ER_ADD_PARTITION_SUBPART_ERROR [4014])
Message: Trying to Add partition(s) with wrong number of subpartitions
- Error: 1514 SQLSTATE: HY000 (ER_ADD_PARTITION_NO_NEW_PARTITION [4014])
Message: At least one partition must be added
- Error: 1515 SQLSTATE: HY000 (ER_COALESCE_PARTITION_NO_PARTITION [4014])
Message: At least one partition must be coalesced
- Error: 1516 SQLSTATE: HY000 (ER_REORG_PARTITION_NOT_EXIST [4014])
Message: More partitions to reorganize than there are partitions
- Error: 1517 SQLSTATE: HY000 (ER_SAME_NAME_PARTITION [4014])
Message: Duplicate partition name %s
- Error: 1518 SQLSTATE: HY000 (ER_NO_BINLOG_ERROR [4014])
Message: It is not allowed to shut off binlog on this command
- Error: 1519 SQLSTATE: HY000 (ER_CONSECUTIVE_REORG_PARTITIONS [4014])
Message: When reorganizing a set of partitions they must be in consecutive order
- Error: 1520 SQLSTATE: HY000 (ER_REORG_OUTSIDE_RANGE [4014])
Message: Reorganize of range partitions cannot change total ranges except for last partition where it can extend the range
- Error: 1521 SQLSTATE: HY000 (ER_PARTITION_FUNCTION_FAILURE [4014])
Message: Partition function not supported in this version for this handler
- Error: 1522 SQLSTATE: HY000 (ER_PART_STATE_ERROR [4014])
Message: Partition state cannot be defined from CREATE/ALTER TABLE
- Error: 1523 SQLSTATE: HY000 (ER_LIMITED_PART_RANGE [4014])
Message: The %s handler only supports 32 bit integers in VALUES
- Error: 1524 SQLSTATE: HY000 (ER_PLUGIN_IS_NOT_LOADED [4014])

Message: Plugin '%s' is not loaded

- Error: 1525 SQLSTATE: HY000 (ER_WRONG_VALUE [4015])

Message: Incorrect %s value: '%s'

- Error: 1526 SQLSTATE: HY000 (ER_NO_PARTITION_FOR_GIVEN_VALUE [4015])

Message: Table has no partition for value %s

- Error: 1527 SQLSTATE: HY000 (ER_FILEGROUP_OPTION_ONLY_ONCE [4015])

Message: It is not allowed to specify %s more than once

- Error: 1528 SQLSTATE: HY000 (ER_CREATE_FILEGROUP_FAILED [4015])

Message: Failed to create %s

- Error: 1529 SQLSTATE: HY000 (ER_DROP_FILEGROUP_FAILED [4015])

Message: Failed to drop %s

- Error: 1530 SQLSTATE: HY000 (ER_TABLESPACE_AUTO_EXTEND_ERROR [4015])

Message: The handler doesn't support autoextend of tablespaces

- Error: 1531 SQLSTATE: HY000 (ER_WRONG_SIZE_NUMBER [4015])

Message: A size parameter was incorrectly specified, either number or on the form 10M

- Error: 1532 SQLSTATE: HY000 (ER_SIZE_OVERFLOW_ERROR [4015])

Message: The size number was correct but we don't allow the digit part to be more than 2 billion

- Error: 1533 SQLSTATE: HY000 (ER_ALTER_FILEGROUP_FAILED [4015])

Message: Failed to alter: %s

- Error: 1534 SQLSTATE: HY000 (ER_BINLOG_ROW_LOGGING_FAILED [4015])

Message: Writing one row to the row-based binary log failed

- Error: 1535 SQLSTATE: HY000 (ER_BINLOG_ROW_WRONG_TABLE_DEF [4015])

Message: Table definition on master and slave does not match: %s

- Error: 1536 SQLSTATE: HY000 (ER_BINLOG_ROW_RBR_TO_SBR [4015])

Message: Slave running with --log-slave-updates must use row-based binary logging to be able to replicate row-based binary log events

- Error: 1537 SQLSTATE: HY000 (ER_EVENT_ALREADY_EXISTS [4015])

Message: Event '%s' already exists

- Error: 1538 SQLSTATE: HY000 (ER_EVENT_STORE_FAILED [4015])

Message: Failed to store event %s. Error code %d from storage engine.

- Error: 1539 SQLSTATE: HY000 (ER_EVENT_DOES_NOT_EXIST [4016])
Message: Unknown event '%s'
- Error: 1540 SQLSTATE: HY000 (ER_EVENT_CANT_ALTER [4016])
Message: Failed to alter event '%s'
- Error: 1541 SQLSTATE: HY000 (ER_EVENT_DROP_FAILED [4016])
Message: Failed to drop %s
- Error: 1542 SQLSTATE: HY000 (ER_EVENT_INTERVAL_NOT_POSITIVE_OR_TOO_BIG [4016])
Message: INTERVAL is either not positive or too big
- Error: 1543 SQLSTATE: HY000 (ER_EVENT_ENDS_BEFORE_STARTS [4016])
Message: ENDS is either invalid or before STARTS
- Error: 1544 SQLSTATE: HY000 (ER_EVENT_EXEC_TIME_IN_THE_PAST [4016])
Message: Event execution time is in the past. Event has been disabled
- Error: 1545 SQLSTATE: HY000 (ER_EVENT_OPEN_TABLE_FAILED [4016])
Message: Failed to open mysql.event
- Error: 1546 SQLSTATE: HY000 (ER_EVENT_NEITHER_M_EXPR_NOR_M_AT [4016])
Message: No datetime expression provided
- Error: 1547 SQLSTATE: HY000 (ER_COL_COUNT_DOESNT_MATCH_CORRUPTED [4016])
Message: Column count of mysql.%s is wrong. Expected %d, found %d. The table is probably corrupted
- Error: 1548 SQLSTATE: HY000 (ER_CANNOT_LOAD_FROM_TABLE [4016])
Message: Cannot load from mysql.%s. The table is probably corrupted
- Error: 1549 SQLSTATE: HY000 (ER_EVENT_CANNOT_DELETE [4016])
Message: Failed to delete the event from mysql.event
- Error: 1550 SQLSTATE: HY000 (ER_EVENT_COMPILE_ERROR [4016])
Message: Error during compilation of event's body
- Error: 1551 SQLSTATE: HY000 (ER_EVENT_SAME_NAME [4016])
Message: Same old and new event name
- Error: 1552 SQLSTATE: HY000 (ER_EVENT_DATA_TOO_LONG [4016])
Message: Data for column '%s' too long
- Error: 1553 SQLSTATE: HY000 (ER_DROP_INDEX_FK [4016])
Message: Cannot drop index '%s': needed in a foreign key constraint

- Error: 1554 SQLSTATE: HY000 ([ER_WARN_DEPRECATED_SYNTAX_WITH_VER \[4017\]](#))
Message: The syntax '%s' is deprecated and will be removed in MySQL %s. Please use %s instead
- Error: 1555 SQLSTATE: HY000 ([ER_CANT_WRITE_LOCK_LOG_TABLE \[4017\]](#))
Message: You can't write-lock a log table. Only read access is possible
- Error: 1556 SQLSTATE: HY000 ([ER_CANT_LOCK_LOG_TABLE \[4017\]](#))
Message: You can't use locks with log tables.
- Error: 1557 SQLSTATE: 23000 ([ER_FOREIGN_DUPLICATE_KEY \[4017\]](#))
Message: Upholding foreign key constraints for table '%s', entry '%s', key %d would lead to a duplicate entry
- Error: 1558 SQLSTATE: HY000 ([ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE \[4017\]](#))
Message: Column count of mysql.%s is wrong. Expected %d, found %d. Created with MySQL %d, now running %d. Please use mysql_upgrade to fix this error.
- Error: 1559 SQLSTATE: HY000 ([ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR \[4017\]](#))
Message: Cannot switch out of the row-based binary log format when the session has open temporary tables
- Error: 1560 SQLSTATE: HY000 ([ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_FORMAT \[4017\]](#))
Message: Cannot change the binary logging format inside a stored function or trigger
- Error: 1561 SQLSTATE: HY000 ([ER_NDB_CANT_SWITCH_BINLOG_FORMAT \[4017\]](#))
Message: The NDB cluster engine does not support changing the binlog format on the fly yet
- Error: 1562 SQLSTATE: HY000 ([ER_PARTITION_NO_TEMPORARY \[4017\]](#))
Message: Cannot create temporary table with partitions
- Error: 1563 SQLSTATE: HY000 ([ER_PARTITION_CONST_DOMAIN_ERROR \[4017\]](#))
Message: Partition constant is out of partition function domain
- Error: 1564 SQLSTATE: HY000 ([ER_PARTITION_FUNCTION_IS_NOT_ALLOWED \[4017\]](#))
Message: This partition function is not allowed
- Error: 1565 SQLSTATE: HY000 ([ER_DDL_LOG_ERROR \[4017\]](#))
Message: Error in DDL log
- Error: 1566 SQLSTATE: HY000 ([ER_NULL_IN_VALUES_LESS_THAN \[4017\]](#))
Message: Not allowed to use NULL value in VALUES LESS THAN
- Error: 1567 SQLSTATE: HY000 ([ER_WRONG_PARTITION_NAME \[4017\]](#))
Message: Incorrect partition name

- Error: 1568 SQLSTATE: 25001 ([ER_CANT_CHANGE_TX_ISOLATION \[4018\]](#))
Message: Transaction isolation level can't be changed while a transaction is in progress
- Error: 1569 SQLSTATE: HY000 ([ER_DUP_ENTRY_AUTOINCREMENT_CASE \[4018\]](#))
Message: ALTER TABLE causes auto_increment resequencing, resulting in duplicate entry '%s' for key '%s'
- Error: 1570 SQLSTATE: HY000 ([ER_EVENT_MODIFY_QUEUE_ERROR \[4018\]](#))
Message: Internal scheduler error %d
- Error: 1571 SQLSTATE: HY000 ([ER_EVENT_SET_VAR_ERROR \[4018\]](#))
Message: Error during starting/stopping of the scheduler. Error code %u
- Error: 1572 SQLSTATE: HY000 ([ER_PARTITION_MERGE_ERROR \[4018\]](#))
Message: Engine cannot be used in partitioned tables
- Error: 1573 SQLSTATE: HY000 ([ER_CANT_ACTIVATE_LOG \[4018\]](#))
Message: Cannot activate '%s' log
- Error: 1574 SQLSTATE: HY000 ([ER_RBR_NOT_AVAILABLE \[4018\]](#))
Message: The server was not built with row-based replication
- Error: 1575 SQLSTATE: HY000 ([ER_BASE64_DECODE_ERROR \[4018\]](#))
Message: Decoding of base64 string failed
- Error: 1576 SQLSTATE: HY000 ([ER_EVENT_RECURSION_FORBIDDEN \[4018\]](#))
Message: Recursion of EVENT DDL statements is forbidden when body is present
- Error: 1577 SQLSTATE: HY000 ([ER_EVENTS_DB_ERROR \[4018\]](#))
Message: Cannot proceed because system tables used by Event Scheduler were found damaged at server start
- Error: 1578 SQLSTATE: HY000 ([ER_ONLY_INTEGERS_ALLOWED \[4018\]](#))
Message: Only integers allowed as number here
- Error: 1579 SQLSTATE: HY000 ([ER_UNSUPPORTED_LOG_ENGINE \[4018\]](#))
Message: This storage engine cannot be used for log tables"
- Error: 1580 SQLSTATE: HY000 ([ER_BAD_LOG_STATEMENT \[4018\]](#))
Message: You cannot '%s' a log table if logging is enabled
- Error: 1581 SQLSTATE: HY000 ([ER_CANT_RENAME_LOG_TABLE \[4018\]](#))
Message: Cannot rename '%s'. When logging enabled, rename to/from log table must rename two tables: the log table to an archive table and another table back to '%s'
- Error: 1582 SQLSTATE: 42000 ([ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT \[4018\]](#))

Message: Incorrect parameter count in the call to native function '%s'

- Error: 1583 SQLSTATE: 42000 (ER_WRONG_PARAMETERS_TO_NATIVE_FCT [4019])

Message: Incorrect parameters in the call to native function '%s'

- Error: 1584 SQLSTATE: 42000 (ER_WRONG_PARAMETERS_TO_STORED_FCT [4019])

Message: Incorrect parameters in the call to stored function '%s'

- Error: 1585 SQLSTATE: HY000 (ER_NATIVE_FCT_NAME_COLLISION [4019])

Message: This function '%s' has the same name as a native function

- Error: 1586 SQLSTATE: 23000 (ER_DUP_ENTRY_WITH_KEY_NAME [4019])

Message: Duplicate entry '%s' for key '%s'

- Error: 1587 SQLSTATE: HY000 (ER_BINLOG_PURGE_EMFILE [4019])

Message: Too many files opened, please execute the command again

- Error: 1588 SQLSTATE: HY000 (ER_EVENT_CANNOT_CREATE_IN_THE_PAST [4019])

Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.

- Error: 1589 SQLSTATE: HY000 (ER_EVENT_CANNOT_ALTER_IN_THE_PAST [4019])

Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.

- Error: 1590 SQLSTATE: HY000 (ER_SLAVE_INCIDENT [4019])

Message: The incident %s occurred on the master. Message: %s

- Error: 1591 SQLSTATE: HY000 (ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT [4019])

Message: Table has no partition for some existing values

- Error: 1592 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_STATEMENT [4019])

Message: Unsafe statement written to the binary log using statement format since BINLOG_FORMAT = STATEMENT. %s

- Error: 1593 SQLSTATE: HY000 (ER_SLAVE_FATAL_ERROR [4019])

Message: Fatal error: %s

- Error: 1594 SQLSTATE: HY000 (ER_SLAVE_RELAY_LOG_READ_FAILURE [4019])

Message: Relay log read failure: %s

- Error: 1595 SQLSTATE: HY000 (ER_SLAVE_RELAY_LOG_WRITE_FAILURE [4019])

Message: Relay log write failure: %s

- Error: 1596 SQLSTATE: HY000 (ER_SLAVE_CREATE_EVENT_FAILURE [4019])

Message: Failed to create %s

- Error: 1597 SQLSTATE: HY000 (ER_SLAVE_MASTER_COM_FAILURE [4020])

Message: Master command %s failed: %s

- Error: 1598 SQLSTATE: HY000 (ER_BINLOG_LOGGING_IMPOSSIBLE [4020])

Message: Binary logging not possible. Message: %s

- Error: 1599 SQLSTATE: HY000 (ER_VIEW_NO_CREATION_CTX [4020])

Message: View `%s`.`%s` has no creation context

- Error: 1600 SQLSTATE: HY000 (ER_VIEW_INVALID_CREATION_CTX [4020])

Message: Creation context of view `%s`.`%s` is invalid

- Error: 1601 SQLSTATE: HY000 (ER_SR_INVALID_CREATION_CTX [4020])

Message: Creation context of stored routine `%s`.`%s` is invalid

- Error: 1602 SQLSTATE: HY000 (ER_TRG_CORRUPTED_FILE [4020])

Message: Corrupted TRG file for table `%s`.`%s`

- Error: 1603 SQLSTATE: HY000 (ER_TRG_NO_CREATION_CTX [4020])

Message: Triggers for table `%s`.`%s` have no creation context

- Error: 1604 SQLSTATE: HY000 (ER_TRG_INVALID_CREATION_CTX [4020])

Message: Trigger creation context of table `%s`.`%s` is invalid

- Error: 1605 SQLSTATE: HY000 (ER_EVENT_INVALID_CREATION_CTX [4020])

Message: Creation context of event `%s`.`%s` is invalid

- Error: 1606 SQLSTATE: HY000 (ER_TRG_CANT_OPEN_TABLE [4020])

Message: Cannot open table for trigger `%s`.`%s`

- Error: 1607 SQLSTATE: HY000 (ER_CANT_CREATE_SROUTINE [4020])

Message: Cannot create stored routine `%s`. Check warnings

- Error: 1608 SQLSTATE: HY000 (ER_NEVER_USED [4020])

Message: Ambiguous slave modes combination. %s

- Error: 1609 SQLSTATE: HY000 (ER_NO_FORMAT_DESCRIPTION_EVENT_BEFORE_BINLOG_STATEMENT [4020])

Message: The BINLOG statement of type `%s` was not preceded by a format description BINLOG statement.

- Error: 1610 SQLSTATE: HY000 (ER_SLAVE_CORRUPT_EVENT [4020])

Message: Corrupted replication event was detected

- Error: 1611 SQLSTATE: HY000 (ER_LOAD_DATA_INVALID_COLUMN [4021])
Message: Invalid column reference (%s) in LOAD DATA
- Error: 1612 SQLSTATE: HY000 (ER_LOG_PURGE_NO_FILE [4021])
Message: Being purged log %s was not found
- Error: 1613 SQLSTATE: XA106 (ER_XA_RBTIMEOUT [4021])
Message: XA_RBTIMEOUT: Transaction branch was rolled back: took too long
- Error: 1614 SQLSTATE: XA102 (ER_XA_RBDEADLOCK [4021])
Message: XA_RBDEADLOCK: Transaction branch was rolled back: deadlock was detected
- Error: 1615 SQLSTATE: HY000 (ER_NEED_REPREPARE [4021])
Message: Prepared statement needs to be re-prepared
- Error: 1616 SQLSTATE: HY000 (ER_DELAYED_NOT_SUPPORTED [4021])
Message: DELAYED option not supported for table '%s'
- Error: 1617 SQLSTATE: HY000 (WARN_NO_MASTER_INFO [4021])
Message: The master info structure does not exist
- Error: 1618 SQLSTATE: HY000 (WARN_OPTION_IGNORED [4021])
Message: <%s> option ignored
- Error: 1619 SQLSTATE: HY000 (WARN_PLUGIN_DELETE_BUILTIN [4021])
Message: Built-in plugins cannot be deleted
- Error: 1620 SQLSTATE: HY000 (WARN_PLUGIN_BUSY [4021])
Message: Plugin is busy and will be uninstalled on shutdown
- Error: 1621 SQLSTATE: HY000 (ER_VARIABLE_IS_READONLY [4021])
Message: %s variable '%s' is read-only. Use SET %s to assign the value
- Error: 1622 SQLSTATE: HY000 (ER_WARN_ENGINE_TRANSACTION_ROLLBACK [4021])
Message: Storage engine %s does not support rollback for this statement. Transaction rolled back and must be restarted
- Error: 1623 SQLSTATE: HY000 (ER_SLAVE_HEARTBEAT_FAILURE [4021])
Message: Unexpected master's heartbeat data: %s
- Error: 1624 SQLSTATE: HY000 (ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE [4021])
Message: The requested value for the heartbeat period is either negative or exceeds the maximum allowed (%s seconds).
- Error: 1625 SQLSTATE: HY000 (ER_NDB_REPLICATION_SCHEMA_ERROR [4021])

- Message: Bad schema for mysql.ndb_replication table. Message: %s
- Error: 1626 SQLSTATE: HY000 ([ER_CONFLICT_FN_PARSE_ERROR \[4022\]](#))
Message: Error in parsing conflict function. Message: %s
 - Error: 1627 SQLSTATE: HY000 ([ER_EXCEPTIONS_WRITE_ERROR \[4022\]](#))
Message: Write to exceptions table failed. Message: %s"
 - Error: 1628 SQLSTATE: HY000 ([ER_TOO_LONG_TABLE_COMMENT \[4022\]](#))
Message: Comment for table '%s' is too long (max = %lu)
 - Error: 1629 SQLSTATE: HY000 ([ER_TOO_LONG_FIELD_COMMENT \[4022\]](#))
Message: Comment for field '%s' is too long (max = %lu)
 - Error: 1630 SQLSTATE: 42000 ([ER_FUNC_INEXISTENT_NAME_COLLISION \[4022\]](#))
Message: FUNCTION %s does not exist. Check the 'Function Name Parsing and Resolution' section in the Reference Manual
 - Error: 1631 SQLSTATE: HY000 ([ER_DATABASE_NAME \[4022\]](#))
Message: Database
 - Error: 1632 SQLSTATE: HY000 ([ER_TABLE_NAME \[4022\]](#))
Message: Table
 - Error: 1633 SQLSTATE: HY000 ([ER_PARTITION_NAME \[4022\]](#))
Message: Partition
 - Error: 1634 SQLSTATE: HY000 ([ER_SUBPARTITION_NAME \[4022\]](#))
Message: Subpartition
 - Error: 1635 SQLSTATE: HY000 ([ER_TEMPORARY_NAME \[4022\]](#))
Message: Temporary
 - Error: 1636 SQLSTATE: HY000 ([ER_RENAMED_NAME \[4022\]](#))
Message: Renamed
 - Error: 1637 SQLSTATE: HY000 ([ER_TOO_MANY_CONCURRENT_TRXS \[4022\]](#))
Message: Too many active concurrent transactions
 - Error: 1638 SQLSTATE: HY000 ([WARN_NON_ASCII_SEPARATOR_NOT_IMPLEMENTED \[4022\]](#))
Message: Non-ASCII separator arguments are not fully supported
 - Error: 1639 SQLSTATE: HY000 ([ER_DEBUG_SYNC_TIMEOUT \[4022\]](#))
Message: debug sync point wait timed out

- Error: 1640 SQLSTATE: HY000 ([ER_DEBUG_SYNC_HIT_LIMIT \[4023\]](#))
Message: debug sync point hit limit reached
- Error: 1641 SQLSTATE: 42000 ([ER_DUP_SIGNAL_SET \[4023\]](#))
Message: Duplicate condition information item '%s'
- Error: 1642 SQLSTATE: 01000 ([ER_SIGNAL_WARN \[4023\]](#))
Message: Unhandled user-defined warning condition
- Error: 1643 SQLSTATE: 02000 ([ER_SIGNAL_NOT_FOUND \[4023\]](#))
Message: Unhandled user-defined not found condition
- Error: 1644 SQLSTATE: HY000 ([ER_SIGNAL_EXCEPTION \[4023\]](#))
Message: Unhandled user-defined exception condition
- Error: 1645 SQLSTATE: 0K000 ([ER_RESIGNAL_WITHOUT_ACTIVE_HANDLER \[4023\]](#))
Message: RESIGNAL when handler not active
- Error: 1646 SQLSTATE: HY000 ([ER_SIGNAL_BAD_CONDITION_TYPE \[4023\]](#))
Message: SIGNAL/RESIGNAL can only use a CONDITION defined with SQLSTATE
- Error: 1647 SQLSTATE: HY000 ([WARN_COND_ITEM_TRUNCATED \[4023\]](#))
Message: Data truncated for condition item '%s'
- Error: 1648 SQLSTATE: HY000 ([ER_COND_ITEM_TOO_LONG \[4023\]](#))
Message: Data too long for condition item '%s'
- Error: 1649 SQLSTATE: HY000 ([ER_UNKNOWN_LOCALE \[4023\]](#))
Message: Unknown locale: '%s'
- Error: 1650 SQLSTATE: HY000 ([ER_SLAVE_IGNORE_SERVER_IDS \[4023\]](#))
Message: The requested server id %d clashes with the slave startup option --replicate-same-server-id
- Error: 1651 SQLSTATE: HY000 ([ER_QUERY_CACHE_DISABLED \[4023\]](#))
Message: Query cache is disabled; restart the server with query_cache_type=1 to enable it
- Error: 1652 SQLSTATE: HY000 ([ER_SAME_NAME_PARTITION_FIELD \[4023\]](#))
Message: Duplicate partition field name '%s'
- Error: 1653 SQLSTATE: HY000 ([ER_PARTITION_COLUMN_LIST_ERROR \[4023\]](#))
Message: Inconsistency in usage of column lists for partitioning
- Error: 1654 SQLSTATE: HY000 ([ER_WRONG_TYPE_COLUMN_VALUE_ERROR \[4023\]](#))
Message: Partition column values of incorrect type
- Error: 1655 SQLSTATE: HY000 ([ER_TOO_MANY_PARTITION_FUNC_FIELDS_ERROR \[4023\]](#))

Message: Too many fields in '%s'

- Error: 1656 SQLSTATE: HY000 ([ER_MAXVALUE_IN_VALUES_IN \[4024\]](#))

Message: Cannot use MAXVALUE as value in VALUES IN

- Error: 1657 SQLSTATE: HY000 ([ER_TOO_MANY_VALUES_ERROR \[4024\]](#))

Message: Cannot have more than one value for this type of %s partitioning

- Error: 1658 SQLSTATE: HY000 ([ER_ROW_SINGLE_PARTITION_FIELD_ERROR \[4024\]](#))

Message: Row expressions in VALUES IN only allowed for multi-field column partitioning

- Error: 1659 SQLSTATE: HY000 ([ER_FIELD_TYPE_NOT_ALLOWED_AS_PARTITION_FIELD \[4024\]](#))

Message: Field '%s' is of a not allowed type for this type of partitioning

- Error: 1660 SQLSTATE: HY000 ([ER_PARTITION_FIELDS_TOO_LONG \[4024\]](#))

Message: The total length of the partitioning fields is too large

- Error: 1661 SQLSTATE: HY000 ([ER_BINLOG_ROW_ENGINE_AND_STMT_ENGINE \[4024\]](#))

Message: Cannot execute statement: impossible to write to binary log since both row-incapable engines and statement-incapable engines are involved.

- Error: 1662 SQLSTATE: HY000 ([ER_BINLOG_ROW_MODE_AND_STMT_ENGINE \[4024\]](#))

Message: Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = ROW and at least one table uses a storage engine limited to statement-based logging.

- Error: 1663 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_AND_STMT_ENGINE \[4024\]](#))

Message: Cannot execute statement: impossible to write to binary log since statement is unsafe, storage engine is limited to statement-based logging, and BINLOG_FORMAT = MIXED. %s

- Error: 1664 SQLSTATE: HY000 ([ER_BINLOG_ROW_INJECTION_AND_STMT_ENGINE \[4024\]](#))

Message: Cannot execute statement: impossible to write to binary log since statement is in row format and at least one table uses a storage engine limited to statement-based logging.

- Error: 1665 SQLSTATE: HY000 ([ER_BINLOG_STMT_MODE_AND_ROW_ENGINE \[4024\]](#))

Message: Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine limited to row-based logging.%s

- Error: 1666 SQLSTATE: HY000 ([ER_BINLOG_ROW_INJECTION_AND_STMT_MODE \[4024\]](#))

Message: Cannot execute statement: impossible to write to binary log since statement is in row format and BINLOG_FORMAT = STATEMENT.

- Error: 1667 SQLSTATE: HY000 ([ER_BINLOG_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE \[4024\]](#))

Message: Cannot execute statement: impossible to write to binary log since more than one engine is involved and at least one engine is self-logging.

- Error: 1668 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_LIMIT \[4024\]](#))

Message: The statement is unsafe because it uses a LIMIT clause. This is unsafe because the set of rows included cannot be predicted.

- Error: 1669 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_INSERT_DELAYED \[4025\]](#))

Message: The statement is unsafe because it uses INSERT DELAYED. This is unsafe because the times when rows are inserted cannot be predicted.

- Error: 1670 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_SYSTEM_TABLE \[4025\]](#))

Message: The statement is unsafe because it uses the general log, slow query log, or performance_schema table(s). This is unsafe because system tables may differ on slaves.

- Error: 1671 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_AUTOINC_COLUMNS \[4025\]](#))

Message: Statement is unsafe because it invokes a trigger or a stored function that inserts into an AUTO_INCREMENT column. Inserted values cannot be logged correctly.

- Error: 1672 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_UDF \[4025\]](#))

Message: Statement is unsafe because it uses a UDF which may not return the same value on the slave.

- Error: 1673 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_SYSTEM_VARIABLE \[4025\]](#))

Message: Statement is unsafe because it uses a system variable that may have a different value on the slave.

- Error: 1674 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_SYSTEM_FUNCTION \[4025\]](#))

Message: Statement is unsafe because it uses a system function that may return a different value on the slave.

- Error: 1675 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_NONTRANS_AFTER_TRANS \[4025\]](#))

Message: Statement is unsafe because it accesses a non-transactional table after accessing a transactional table within the same transaction.

- Error: 1676 SQLSTATE: HY000 ([ER_MESSAGE_AND_STATEMENT \[4025\]](#))

Message: %s Statement: %s

- Error: 1677 SQLSTATE: HY000 ([ER_SLAVE_CONVERSION_FAILED \[4025\]](#))

Message: Column %d of table '%s.%s' cannot be converted from type '%s' to type '%s'

- Error: 1678 SQLSTATE: HY000 ([ER_SLAVE_CANT_CREATE_CONVERSION \[4025\]](#))

Message: Can't create conversion table for table '%s.%s'

- Error: 1679 SQLSTATE: HY000 ([ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_FORMAT \[4025\]](#))

Message: Cannot modify @@session.binlog_format inside a transaction

- Error: 1680 SQLSTATE: HY000 ([ER_PATH_LENGTH \[4025\]](#))

Message: The path specified for %s is too long.

- Error: 1681 SQLSTATE: HY000 ([ER_WARN_DEPRECATED_SYNTAX_NO_REPLACEMENT \[4026\]](#))
Message: '%s' is deprecated and will be removed in a future release.
- Error: 1682 SQLSTATE: HY000 ([ER_WRONG_NATIVE_TABLE_STRUCTURE \[4026\]](#))
Message: Native table '%s'.'%s' has the wrong structure
- Error: 1683 SQLSTATE: HY000 ([ER_WRONG_PERFSCHEMA_USAGE \[4026\]](#))
Message: Invalid performance_schema usage.
- Error: 1684 SQLSTATE: HY000 ([ER_WARN_I_S_SKIPPED_TABLE \[4026\]](#))
Message: Table '%s'.'%s' was skipped since its definition is being modified by concurrent DDL statement
- Error: 1685 SQLSTATE: HY000
([ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_DIRECT \[4026\]](#))
Message: Cannot modify @@session.binlog_direct_non_transactional_updates inside a transaction
- Error: 1686 SQLSTATE: HY000
([ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_DIRECT \[4026\]](#))
Message: Cannot change the binlog direct flag inside a stored function or trigger
- Error: 1687 SQLSTATE: 42000 ([ER_SPATIAL_MUST_HAVE_GEOM_COL \[4026\]](#))
Message: A SPATIAL index may only contain a geometrical type column
- Error: 1688 SQLSTATE: HY000 ([ER_TOO_LONG_INDEX_COMMENT \[4026\]](#))
Message: Comment for index '%s' is too long (max = %lu)
- Error: 1689 SQLSTATE: HY000 ([ER_LOCK_ABORTED \[4026\]](#))
Message: Wait on a lock was aborted due to a pending exclusive lock
- Error: 1690 SQLSTATE: 22003 ([ER_DATA_OUT_OF_RANGE \[4026\]](#))
Message: %s value is out of range in '%s'
- Error: 1691 SQLSTATE: HY000 ([ER_WRONG_SPVAR_TYPE_IN_LIMIT \[4026\]](#))
Message: A variable of a non-integer based type in LIMIT clause
- Error: 1692 SQLSTATE: HY000
([ER_BINLOG_UNSAFE_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE \[4026\]](#))
Message: Mixing self-logging and non-self-logging engines in a statement is unsafe.
- Error: 1693 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_MIXED_STATEMENT \[4026\]](#))
Message: Statement accesses nontransactional table as well as transactional or temporary table, and writes to any of them.
- Error: 1694 SQLSTATE: HY000
([ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_SQL_LOG_BIN \[4026\]](#))
Message: Cannot modify @@session.sql_log_bin inside a transaction

- Error: 1695 SQLSTATE: HY000 ([ER_STORED_FUNCTION_PREVENTS_SWITCH_SQL_LOG_BIN \[4027\]](#))
Message: Cannot change the sql_log_bin inside a stored function or trigger
- Error: 1696 SQLSTATE: HY000 ([ER_FAILED_READ_FROM_PAR_FILE \[4027\]](#))
Message: Failed to read from the .par file
- Error: 1697 SQLSTATE: HY000 ([ER_VALUES_IS_NOT_INT_TYPE_ERROR \[4027\]](#))
Message: VALUES value for partition '%s' must have type INT
- Error: 1698 SQLSTATE: 28000 ([ER_ACCESS_DENIED_NO_PASSWORD_ERROR \[4027\]](#))
Message: Access denied for user '%s'@'%s'
- Error: 1699 SQLSTATE: HY000 ([ER_SET_PASSWORD_AUTH_PLUGIN \[4027\]](#))
Message: SET PASSWORD has no significance for users authenticating via plugins
- Error: 1700 SQLSTATE: HY000 ([ER_GRANT_PLUGIN_USER_EXISTS \[4027\]](#))
Message: GRANT with IDENTIFIED WITH is illegal because the user %-.*s already exists
- Error: 1701 SQLSTATE: 42000 ([ER_TRUNCATE_ILLEGAL_FK \[4027\]](#))
Message: Cannot truncate a table referenced in a foreign key constraint (%s)
- Error: 1702 SQLSTATE: HY000 ([ER_PLUGIN_IS_PERMANENT \[4027\]](#))
Message: Plugin '%s' is force_plus_permanent and can not be unloaded
- Error: 1703 SQLSTATE: HY000 ([ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MIN \[4027\]](#))
Message: The requested value for the heartbeat period is less than 1 millisecond. The value is reset to 0, meaning that heartbeating will effectively be disabled.
- Error: 1704 SQLSTATE: HY000 ([ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MAX \[4027\]](#))
Message: The requested value for the heartbeat period exceeds the value of `slave_net_timeout` seconds. A sensible value for the period should be less than the timeout.
- Error: 1705 SQLSTATE: HY000 ([ER_STMT_CACHE_FULL \[4027\]](#))
Message: Multi-row statements required more than 'max_binlog_stmt_cache_size' bytes of storage; increase this mysqld variable and try again
- Error: 1706 SQLSTATE: HY000 ([ER_MULTI_UPDATE_KEY_CONFLICT \[4027\]](#))
Message: Primary key/partition key update is not allowed since the table is updated both as '%s' and '%s'.
- Error: 1707 SQLSTATE: HY000 ([ER_TABLE_NEEDS_REBUILD \[4027\]](#))
Message: Table rebuild required. Please do "ALTER TABLE `%s` FORCE" or dump/reload to fix it!
- Error: 1708 SQLSTATE: HY000 ([WARN_OPTION_BELOW_LIMIT \[4027\]](#))
Message: The value of '%s' should be no less than the value of '%s'

- Error: 1709 SQLSTATE: HY000 ([ER_INDEX_COLUMN_TOO_LONG \[4028\]](#))
Message: Index column size too large. The maximum column size is %lu bytes.
- Error: 1710 SQLSTATE: HY000 ([ER_ERROR_IN_TRIGGER_BODY \[4028\]](#))
Message: Trigger '%s' has an error in its body: '%s'
- Error: 1711 SQLSTATE: HY000 ([ER_ERROR_IN_UNKNOWN_TRIGGER_BODY \[4028\]](#))
Message: Unknown trigger has an error in its body: '%s'
- Error: 1712 SQLSTATE: HY000 ([ER_INDEX_CORRUPT \[4028\]](#))
Message: Index %s is corrupted
- Error: 1713 SQLSTATE: HY000 ([ER_UNDO_RECORD_TOO_BIG \[4028\]](#))
Message: Undo log record is too big.
- Error: 1714 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_INSERT_IGNORE_SELECT \[4028\]](#))
Message: INSERT IGNORE... SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.
- Error: 1715 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_INSERT_SELECT_UPDATE \[4028\]](#))
Message: INSERT... SELECT... ON DUPLICATE KEY UPDATE is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are updated. This order cannot be predicted and may differ on master and the slave.
- Error: 1716 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_REPLACE_SELECT \[4028\]](#))
Message: REPLACE... SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are replaced. This order cannot be predicted and may differ on master and the slave.
- Error: 1717 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_CREATE_IGNORE_SELECT \[4028\]](#))
Message: CREATE... IGNORE SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.
- Error: 1718 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_CREATE_REPLACE_SELECT \[4028\]](#))
Message: CREATE... REPLACE SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are replaced. This order cannot be predicted and may differ on master and the slave.
- Error: 1719 SQLSTATE: HY000 ([ER_BINLOG_UNSAFE_UPDATE_IGNORE \[4028\]](#))
Message: UPDATE IGNORE is unsafe because the order in which rows are updated determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.
- Error: 1720 SQLSTATE: HY000 ([ER_PLUGIN_NO_UNINSTALL \[4028\]](#))
Message: Plugin '%s' is marked as not dynamically uninstalleable. You have to stop the server to uninstall it.

- Error: [1721](#) SQLSTATE: [HY000](#) ([ER_PLUGIN_NO_INSTALL](#) [\[4029\]](#))
Message: Plugin '%s' is marked as not dynamically installable. You have to stop the server to install it.
- Error: [1722](#) SQLSTATE: [HY000](#) ([ER_BINLOG_UNSAFE_WRITE_AUTOINC_SELECT](#) [\[4029\]](#))
Message: Statements writing to a table with an auto-increment column after selecting from another table are unsafe because the order in which rows are retrieved determines what (if any) rows will be written. This order cannot be predicted and may differ on master and the slave.
- Error: [1723](#) SQLSTATE: [HY000](#) ([ER_BINLOG_UNSAFE_CREATE_SELECT_AUTOINC](#) [\[4029\]](#))
Message: CREATE TABLE... SELECT... on a table with an auto-increment column is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are inserted. This order cannot be predicted and may differ on master and the slave.
- Error: [1724](#) SQLSTATE: [HY000](#) ([ER_BINLOG_UNSAFE_INSERT_TWO_KEYS](#) [\[4029\]](#))
Message: INSERT... ON DUPLICATE KEY UPDATE on a table with more than one UNIQUE KEY is unsafe
- Error: [1725](#) SQLSTATE: [HY000](#) ([ER_TABLE_IN_FK_CHECK](#) [\[4029\]](#))
Message: Table is being used in foreign key check.
- Error: [1726](#) SQLSTATE: [HY000](#) ([ER_UNSUPPORTED_ENGINE](#) [\[4029\]](#))
Message: Storage engine '%s' does not support system tables. [%s.%s]
- Error: [1727](#) SQLSTATE: [HY000](#) ([ER_BINLOG_UNSAFE_AUTOINC_NOT_FIRST](#) [\[4029\]](#))
Message: INSERT into autoincrement field which is not the first part in the composed primary key is unsafe.

C.4. Client Error Codes and Messages

Client error information comes from the following source files:

- The Error values and the symbols in parentheses correspond to definitions in the `include/errmsg.h` MySQL source file.
- The Message values correspond to the error messages that are listed in the `libmysql/errmsg.c` file. `%d` and `%s` represent numbers and strings, respectively, that are substituted into the messages when they are displayed.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: [2000](#) ([CR_UNKNOWN_ERROR](#) [\[4029\]](#))
Message: Unknown MySQL error
- Error: [2001](#) ([CR_SOCKET_CREATE_ERROR](#) [\[4029\]](#))
Message: Can't create UNIX socket (%d)
- Error: [2002](#) ([CR_CONNECTION_ERROR](#) [\[4029\]](#))
Message: Can't connect to local MySQL server through socket '%s' (%d)

- Error: 2003 ([CR_CONN_HOST_ERROR \[4030\]](#))
Message: Can't connect to MySQL server on '%s' (%d)
- Error: 2004 ([CR_IPSOCK_ERROR \[4030\]](#))
Message: Can't create TCP/IP socket (%d)
- Error: 2005 ([CR_UNKNOWN_HOST \[4030\]](#))
Message: Unknown MySQL server host '%s' (%d)
- Error: 2006 ([CR_SERVER_GONE_ERROR \[4030\]](#))
Message: MySQL server has gone away
- Error: 2007 ([CR_VERSION_ERROR \[4030\]](#))
Message: Protocol mismatch; server version = %d, client version = %d
- Error: 2008 ([CR_OUT_OF_MEMORY \[4030\]](#))
Message: MySQL client ran out of memory
- Error: 2009 ([CR_WRONG_HOST_INFO \[4030\]](#))
Message: Wrong host info
- Error: 2010 ([CR_LOCALHOST_CONNECTION \[4030\]](#))
Message: Localhost via UNIX socket
- Error: 2011 ([CR_TCP_CONNECTION \[4030\]](#))
Message: %s via TCP/IP
- Error: 2012 ([CR_SERVER_HANDSHAKE_ERR \[4030\]](#))
Message: Error in server handshake
- Error: 2013 ([CR_SERVER_LOST \[4030\]](#))
Message: Lost connection to MySQL server during query
- Error: 2014 ([CR_COMMANDS_OUT_OF_SYNC \[4030\]](#))
Message: Commands out of sync; you can't run this command now
- Error: 2015 ([CR_NAMEDPIPE_CONNECTION \[4030\]](#))
Message: Named pipe: %s
- Error: 2016 ([CR_NAMEDPIPEWAIT_ERROR \[4030\]](#))
Message: Can't wait for named pipe to host: %s pipe: %s (%lu)
- Error: 2017 ([CR_NAMEDPIPEOPEN_ERROR \[4030\]](#))
Message: Can't open named pipe to host: %s pipe: %s (%lu)

- Error: 2018 ([CR_NAMEDPIPESETSTATE_ERROR \[4031\]](#))
Message: Can't set state of named pipe to host: %s pipe: %s (%lu)
- Error: 2019 ([CR_CANT_READ_CHARSET \[4031\]](#))
Message: Can't initialize character set %s (path: %s)
- Error: 2020 ([CR_NET_PACKET_TOO_LARGE \[4031\]](#))
Message: Got packet bigger than 'max_allowed_packet' bytes
- Error: 2021 ([CR_EMBEDDED_CONNECTION \[4031\]](#))
Message: Embedded server
- Error: 2022 ([CR_PROBE_SLAVE_STATUS \[4031\]](#))
Message: Error on SHOW SLAVE STATUS:
- Error: 2023 ([CR_PROBE_SLAVE_HOSTS \[4031\]](#))
Message: Error on SHOW SLAVE HOSTS:
- Error: 2024 ([CR_PROBE_SLAVE_CONNECT \[4031\]](#))
Message: Error connecting to slave:
- Error: 2025 ([CR_PROBE_MASTER_CONNECT \[4031\]](#))
Message: Error connecting to master:
- Error: 2026 ([CR_SSL_CONNECTION_ERROR \[4031\]](#))
Message: SSL connection error: %s
- Error: 2027 ([CR_MALFORMED_PACKET \[4031\]](#))
Message: Malformed packet
- Error: 2028 ([CR_WRONG_LICENSE \[4031\]](#))
Message: This client library is licensed only for use with MySQL servers having '%s' license
- Error: 2029 ([CR_NULL_POINTER \[4031\]](#))
Message: Invalid use of null pointer
- Error: 2030 ([CR_NO_PREPARE_STMT \[4031\]](#))
Message: Statement not prepared
- Error: 2031 ([CR_PARAMS_NOT_BOUND \[4031\]](#))
Message: No data supplied for parameters in prepared statement
- Error: 2032 ([CR_DATA_TRUNCATED \[4031\]](#))
Message: Data truncated

- Error: 2033 ([CR_NO_PARAMETERS_EXISTS \[4032\]](#))
Message: No parameters exist in the statement
- Error: 2034 ([CR_INVALID_PARAMETER_NO \[4032\]](#))
Message: Invalid parameter number
- Error: 2035 ([CR_INVALID_BUFFER_USE \[4032\]](#))
Message: Can't send long data for non-string/non-binary data types (parameter: %d)
- Error: 2036 ([CR_UNSUPPORTED_PARAM_TYPE \[4032\]](#))
Message: Using unsupported buffer type: %d (parameter: %d)
- Error: 2037 ([CR_SHARED_MEMORY_CONNECTION \[4032\]](#))
Message: Shared memory: %s
- Error: 2038 ([CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR \[4032\]](#))
Message: Can't open shared memory; client could not create request event (%lu)
- Error: 2039 ([CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR \[4032\]](#))
Message: Can't open shared memory; no answer event received from server (%lu)
- Error: 2040 ([CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR \[4032\]](#))
Message: Can't open shared memory; server could not allocate file mapping (%lu)
- Error: 2041 ([CR_SHARED_MEMORY_CONNECT_MAP_ERROR \[4032\]](#))
Message: Can't open shared memory; server could not get pointer to file mapping (%lu)
- Error: 2042 ([CR_SHARED_MEMORY_FILE_MAP_ERROR \[4032\]](#))
Message: Can't open shared memory; client could not allocate file mapping (%lu)
- Error: 2043 ([CR_SHARED_MEMORY_MAP_ERROR \[4032\]](#))
Message: Can't open shared memory; client could not get pointer to file mapping (%lu)
- Error: 2044 ([CR_SHARED_MEMORY_EVENT_ERROR \[4032\]](#))
Message: Can't open shared memory; client could not create %s event (%lu)
- Error: 2045 ([CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR \[4032\]](#))
Message: Can't open shared memory; no answer from server (%lu)
- Error: 2046 ([CR_SHARED_MEMORY_CONNECT_SET_ERROR \[4032\]](#))
Message: Can't open shared memory; cannot send request event to server (%lu)
- Error: 2047 ([CR_CONN_UNKNOW_PROTOCOL \[4032\]](#))
Message: Wrong or unknown protocol

- Error: 2048 ([CR_INVALID_CONN_HANDLE \[4033\]](#))
Message: Invalid connection handle
- Error: 2049 ([CR_SECURE_AUTH \[4033\]](#))
Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)
- Error: 2050 ([CR_FETCH_CANCELED \[4033\]](#))
Message: Row retrieval was canceled by mysql_stmt_close() call
- Error: 2051 ([CR_NO_DATA \[4033\]](#))
Message: Attempt to read column without prior row fetch
- Error: 2052 ([CR_NO_STMT_METADATA \[4033\]](#))
Message: Prepared statement contains no metadata
- Error: 2053 ([CR_NO_RESULT_SET \[4033\]](#))
Message: Attempt to read a row while there is no result set associated with the statement
- Error: 2054 ([CR_NOT_IMPLEMENTED \[4033\]](#))
Message: This feature is not implemented yet
- Error: 2055 ([CR_SERVER_LOST_EXTENDED \[4033\]](#))
Message: Lost connection to MySQL server at '%s', system error: %d
- Error: 2056 ([CR_STMT_CLOSED \[4033\]](#))
Message: Statement closed indirectly because of a preceding %s() call
- Error: 2057 ([CR_NEW_STMT_METADATA \[4033\]](#))
Message: The number of columns in the result set differs from the number of bound buffers. You must reset the statement, rebind the result set columns, and execute the statement again
- Error: 2058 ([CR_ALREADY_CONNECTED \[4033\]](#))
Message: This handle is already connected. Use a separate handle for each connection.
- Error: 2059 ([CR_AUTH_PLUGIN_CANNOT_LOAD \[4033\]](#))
Message: Authentication plugin '%s' cannot be loaded: %s

C.5. Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

C.5.1. How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:
 - The keyboard does not work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light does not change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
 - The mouse pointer does not move.
 - The machine does not answer to a remote machine's pings.
 - Other programs that are not related to MySQL do not behave correctly.
 - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as [glibc](#)) are up to date.

It is always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See [Section 5.2](#), "MySQL Server Logs".
- If you do not think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use `top`, `df`, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it does not want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that the MySQL server or a MySQL client is causing the problem, it is time to create a bug report for our mailing list or our support team. In the bug report, try to give a very detailed description of how the system is behaving and what you think is happening. You should also state why you think that MySQL is causing the problem. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the "copy and paste" method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only "the system does not work." This provides us with no information about what could be the problem.

If a program fails, it is always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with `top`. Let the program run for a while, it may simply be evaluating something computationally intensive.

- If the `mysqld` server is causing problems, can you get any response from it with `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say when you try to connect to the MySQL server? (Try with `mysql`, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in [Section 1.7, “How to Report Bugs or Problems”](#).

C.5.2. Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

C.5.2.1. Access denied

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server permits client programs to use when connecting. See [Section 6.2, “The MySQL Access Privilege System”](#), and [Section 6.2.7, “Causes of Access-Denied Errors”](#).

C.5.2.2. Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the `mysqld` server in two different ways: By using a Unix socket file to connect through a file in the file system (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you do not specify a host name or if you specify the special host name `localhost`.

If the MySQL server is running on Windows, you can connect using TCP/IP. If the server is started with the `--enable-named-pipe [442]` option, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you do not give a host name when connecting to `mysqld`, a MySQL client first tries to connect to the named pipe. If that does not work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the host name.

The error (2002) `Can't connect to ...` normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket file name or TCP/IP port number when trying to connect to the server. You should also check that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

The error (2003) `Can't connect to MySQL server on 'server' (10061)` indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, and that the network port you specified is the one configured on the server.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps xa | grep mysqld` on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See [Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”](#).

If a `mysqld` process is running, you can check it by trying the following commands. The port number or Unix socket file name might be different in your setup. `host_ip` represents the IP address of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
```

```
shell> mysqladmin --protocol=SOCKET --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotation marks with the `hostname` command; these cause the output of `hostname` (that is, the current host name) to be substituted into the `mysqladmin` command. If you have no `hostname` command or are running on Windows, you can manually type the host name of your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with `--skip-networking` [461], it will not accept TCP/IP connections at all. If the server was started with `--bind-address=127.0.0.1` [435], it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or the Windows XP personal firewall may need to be configured not to block the MySQL port.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running on the local host. Check your operating system's process list to ensure the `mysqld` process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: [Section C.5.2.2.1, "Connection to MySQL Server Failing on Windows"](#).
- Someone has removed the Unix socket file that `mysqld` uses (`/tmp/mysql.sock` by default). For example, you might have a `cron` job that removes old files from the `/tmp` directory. You can always run `mysqladmin version` to check whether the Unix socket file that `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `mysql.sock` or to place the socket file somewhere else. See [Section C.5.4.5, "How to Protect or Change the MySQL Unix Socket File"](#).
- You have started the `mysqld` server with the `--socket=/path/to/socket` [464] option, but forgotten to tell client programs the new name of the socket file. If you change the socket path name for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` [240] option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

```
shell> netstat -ln | grep mysql
```

See [Section C.5.4.5, "How to Protect or Change the MySQL Unix Socket File"](#).

- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other `mysqld` threads (for example, with `kill` or with the `mysql_zap` script) before you can restart the MySQL server. See [Section C.5.4.2, "What to Do If MySQL Keeps Crashing"](#).
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart `mysqld` with a `--socket` [464] option that specifies a socket file name in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port [538]` variable.)
- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, make sure you have disabled SELinux protection for the `mysqld` process.

C.5.2.2.1. Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows does not allow for enough ephemeral (short-lived) ports to serve those connections.

The purpose of `TIME_WAIT` is to keep a connection accepting packets even after the connection has been closed. This is because Internet routing can cause a packet to take a slow route to its destination and it may arrive after both sides have agreed to close. If the port is in use for a new connection, that packet from the old connection could break the protocol or compromise personal information from the original connection. The `TIME_WAIT` delay prevents this by ensuring that the port cannot be reused until after some time has been permitted for those delayed packets to arrive.

It is safe to reduce `TIME_WAIT` greatly on LAN connections because there is little chance of packets arriving at very long delays, as they could through the Internet with its comparatively large distances and latencies.

Windows permits ephemeral (short-lived) TCP ports to the user. After any port is closed it will remain in a `TIME_WAIT` status for 120 seconds. The port will not be available again until this time expires. The default range of port numbers depends on the version of Windows, with a more limited number of ports in older versions:

- Windows through Server 2003: Ports in range 1025–5000
- Windows Vista, Server 2008, and newer: Ports in range 49152–65535

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the `TIME_WAIT` status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

IMPORTANT: The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore the registry if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: <http://support.microsoft.com/kb/256986/EN-US/>.

1. Start Registry Editor (`Regedt32.exe`).
2. Locate the following key in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

3. On the **Edit** menu, click **Add Value**, and then add the following registry value:

```
Value Name: MaxUserPort  
Data Type: REG_DWORD  
Value: 65534
```

This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the **Edit** menu, click **Add Value**, and then add the following registry value:

```
Value Name: TcpTimedWaitDelay  
Data Type: REG_DWORD  
Value: 30
```

This sets the number of seconds to hold a TCP port connection in **TIME_WAIT** state before closing. The valid range is between 0 (zero) and 300 (decimal). The default value is 0x78 (120 decimal).

5. Quit Registry Editor.
6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

C.5.2.3. Lost connection to MySQL server

There are three likely causes for this error message.

Usually it indicates network connectivity trouble and you should check the condition of your network if this error occurs frequently. If the error message includes “during query,” this is probably the case you are experiencing.

Sometimes the “during query” form happens when millions of rows are being sent as part of one or more queries. If you know that this is happening, you should try increasing `net_read_timeout` [532] from its default of 30 seconds to 60 seconds or longer, sufficient for the data transfer to complete.

More rarely, it can happen when the client is attempting the initial connection to the server. In this case, if your `connect_timeout` [492] value is set to only a few seconds, you may be able to resolve the problem by increasing it to ten seconds, perhaps more if you have a very long distance or slow connection. You can determine whether you are experiencing this more uncommon cause by using `SHOW GLOBAL STATUS LIKE 'Aborted_connects'`. It will increase by one for each initial connection attempt that the server aborts. You may see “reading authorization packet” as part of the error message; if so, that also suggests that this is the solution that you need.

If the cause is none of those just described, you may be experiencing a problem with **BLOB** values that are larger than `max_allowed_packet` [519], which can cause this error with some clients. Sometime you may see “packet too large” as part of the error message, and that confirms that you need to increase `max_allowed_packet` [519].

C.5.2.4. Client does not support authentication protocol

The current implementation of the authentication protocol uses a password hashing algorithm that is incompatible with that used by older (pre-4.1) clients. Attempts to connect to a 4.1 or newer server with an older client may fail with the following message:

```
shell> mysql  
Client does not support authentication protocol requested
```

```
by server; consider upgrading MySQL client
```

To deal with this problem, the preferred solution is to upgrade all client programs to use a 4.1.1 or newer client library. If that is not possible, use one of the following approaches:

- To connect to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the `SET PASSWORD` statement and the `OLD_PASSWORD()` [1187] function:

```
mysql> SET PASSWORD FOR
-> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
```

Substitute the password you want to use for “*newpwd*” in the preceding example. MySQL cannot tell you what the original password was, so you’ll need to pick a new one.

- Tell the server to use the older password hashing algorithm by default:
 1. Start `mysqld` with the `old_passwords` [534] system variable set to 1.
 2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

```
mysql> SELECT Host, User, Password FROM mysql.user
-> WHERE LENGTH>Password) > 16;
```

For each account record displayed by the query, use the `Host` and `User` values and assign a password using one of the methods described previously.

The `Client does not support authentication protocol` error also can occur if multiple versions of MySQL are installed but client programs are dynamically linked and link to an older library. Make sure that clients use the most recent library version with which they are compatible. The procedure to do this will depend on your system.



Note

The `mysql` extension does not support the authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the `mysql` extension with MySQL 4.1 or newer, you may need to follow one of the options discussed above for configuring MySQL to work with old clients. The `mysqli` extension (stands for “MySQL, Improved”; added in PHP 5) is compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done to use this MySQL client library. For more information about the `mysqli` extension, see <http://php.net/mysqli>.

For additional background on password hashing and authentication, see [Section 6.1.2.4, “Password Hashing in MySQL”](#).

C.5.2.5. Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` [239] or `-p` option that has no following password value:

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs

when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

C.5.2.6. Host '*host_name*' is blocked

If the following error occurs, it means that `mysqld` has received many connection requests from the given host that were interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.  
Unblock with 'mysqladmin flush-hosts'
```

The value of the `max_connect_errors` [519] system variable determines how many successive interrupted connection requests are permitted. (See [Section 5.1.4, "Server System Variables"](#).) After `max_connect_errors` [519] failed requests without a successful connection, `mysqld` assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you issue a `FLUSH HOSTS` statement or execute a `mysqladmin flush-hosts` command.

By default, `mysqld` blocks a host after 10 connection errors. You can adjust the value by setting `max_connect_errors` [519] at server startup:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

The value can also be set at runtime:

```
mysql> SET GLOBAL max_connect_errors=10000;
```

If you get the `Host 'host_name' is blocked` error message for a given host, you should first verify that there is nothing wrong with TCP/IP connections from that host. If you are having network problems, it does you no good to increase the value of the `max_connect_errors` [519] variable.

C.5.2.7. Too many connections

If you get a `Too many connections` error when you try to connect to the `mysqld` server, this means that all available connections are in use by other clients.

The number of connections permitted is controlled by the `max_connections` [519] system variable. The default value is 151 to improve performance when MySQL is used with the Apache Web server. (Previously, the default was 100.) If you need to support more connections, you should set a larger value for this variable.

`mysqld` actually permits `max_connections+1` [519] clients to connect. The extra connection is reserved for use by accounts that have the `SUPER` [697] privilege. By granting the `SUPER` [697] privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See [Section 13.7.5.30, "SHOW PROCESSLIST Syntax"](#).

The maximum number of connections MySQL can support depends on the quality of the thread library on a given platform, the amount of RAM available, how much RAM is used for each connection, the workload from each connection, and the desired response time. Linux or Solaris should be able to support at 500 to 1000 simultaneous connections routinely and as many as 10,000 connections if you have many gigabytes of RAM available and the workload from each is low or the response time target undemanding. Windows is limited to (open tables × 2 + open connections) < 2048 due to the Posix compatibility layer used on that platform.

Increasing `open-files-limit` [456] may be necessary. Also see [Section 2.5, “Installing MySQL on Linux”](#), for how to raise the operating system limit on how many handles can be used by MySQL.

C.5.2.8. Out of memory

If you issue a query using the `mysql` client program and receive an error like the following one, it means that `mysql` does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke `mysql` with the `--quick` [280] option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

C.5.2.9. MySQL server has gone away

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent).

| Error Code | Description |
|--|---|
| <code>CR_SERVER_GONE_ERROR</code> [4030] | The client couldn't send a question to the server. |
| <code>CR_SERVER_LOST</code> [4030] | The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question. |

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` [572] variable when you start `mysqld`. See [Section 5.1.4, “Server System Variables”](#).

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the `MySQL server has gone away` error are:

- You (or the db administrator) has killed the running thread with a `KILL` statement or a `mysqladmin kill` command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL` structure is equal to 0).

- You are using a Windows client and the server had dropped the connection (probably because `wait_timeout` [572] expired) before the command was issued.

The problem on Windows is that in some cases MySQL does not get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

The solution to this is to either do a `mysql_ping()` on the connection if there has been a long time since the last query (this is what Connector/ODBC does) or set `wait_timeout` [572] on the `mysqld` server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by setting the server's `max_allowed_packet` [519] variable, which has a default value of 1MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in [Section C.5.2.10, "Packet too large"](#).

An `INSERT` or `REPLACE` statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per `INSERT` or `REPLACE`.

- You also get a lost connection if you are sending a packet 16MB or larger if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.
- It is also possible to see this error if host name lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working—from MySQL's point of view the problem is indistinguishable from any other network timeout.

You may also see the `MySQL server has gone away` error if MySQL is started with the `--skip-networking` [461] option.

Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.
- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing `mysqladmin version` and examining the server's uptime. If the client connection was broken because `mysqld` crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See [Section C.5.4.2, "What to Do If MySQL Keeps Crashing"](#).

You can get more information about the lost connections by starting `mysqld` with the `--log-warnings=2` [451] option. This logs some of the disconnected errors in the `hostname.err` file. See [Section 5.2.2, "The Error Log"](#).

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See [Section C.5.4.2, "What to Do If MySQL Keeps Crashing"](#).

- If a specific query kills `mysqld` and the tables involved were checked with `CHECK TABLE` before you ran the query, can you provide a reproducible test case? See [MySQL Internals: Porting to Other Systems](#).
- What is the value of the `wait_timeout` [572] system variable in the MySQL server? (`mysqladmin variables` gives you the value of this variable.)
- Have you tried to run `mysqld` with the general query log enabled to determine whether the problem query appears in the log? (See [Section 5.2.3, “The General Query Log”](#).)

See also [Section C.5.2.11, “Communication Errors and Aborted Connections”](#), and [Section 1.7, “How to Report Bugs or Problems”](#).

C.5.2.10. Packet too large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a master replication server to a slave.

The largest possible packet that can be transmitted to or from a MySQL 5.5 server or client is 1GB.

When a MySQL client or the `mysqld` server receives a packet bigger than `max_allowed_packet` [519] bytes, it issues a `Packet too large` error and closes the connection. With some clients, you may also get a `Lost connection to MySQL server during query` error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` [519] variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the `mysql` client program, its default `max_allowed_packet` [519] variable is 16MB. To set a larger value, start `mysql` like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` [519] value is 1MB. You can increase this if the server needs to handle big queries (for example, if you are working with big `BLOB` columns). For example, to set the variable to 16MB, start the server like this:

```
shell> mysqld --max_allowed_packet=16M
```

You can also use an option file to set `max_allowed_packet` [519]. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=16M
```

It is safe to increase the value of this variable because the extra memory is allocated only when needed. For example, `mysqld` allocates more memory only when you issue a long query or when `mysqld` must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

C.5.2.11. Communication Errors and Aborted Connections

If connection problems occur such as communication errors or aborted connections, use these sources of information to diagnose problems:

- The error log. See [Section 5.2.2, “The Error Log”](#).
- The general query log. See [Section 5.2.3, “The General Query Log”](#).
- The `Aborted_xxx` status variables. See [Section 5.1.6, “Server Status Variables”](#).

If you start the server with the `--log-warnings [451]` option, you might find messages like this in your error log:

```
010301 14:38:23 Aborted connection 854 to db:
'users' user: 'josh'
```

If a client successfully connects but later disconnects improperly or is terminated, the server increments the `Aborted_clients [596]` status variable, and logs an `Aborted connection` message to the error log. The cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout [572]` or `interactive_timeout [506]` seconds without issuing any requests to the server. See [Section 5.1.4, “Server System Variables”](#).
- The client program ended abruptly in the middle of a data transfer.

If a client is unable even to connect, the server increments the `Aborted_connects [596]` status variable. Unsuccessful connection attempts can occur for the following reasons:

- A client does not have privileges to connect to a database.
- A client uses an incorrect password.
- A connection packet does not contain the right information.
- It takes more than `connect_timeout [492]` seconds to get a connect packet. See [Section 5.1.4, “Server System Variables”](#).

If these kinds of things happen, it might indicate that someone is trying to break into your server! Messages for these types of problems are logged to the general query log if it is enabled.

Other reasons for problems with aborted clients or aborted connections:

- The `max_allowed_packet [519]` variable value is too small or queries require more memory than you have allocated for `mysqld`. See [Section C.5.2.10, “Packet too large”](#).
- Use of Ethernet protocol with Linux, both half and full duplex. Many Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file using FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. Switch the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and test the results to determine the best setting.
- A problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.

See also [Section C.5.2.9, “MySQL server has gone away”](#).

C.5.2.12. The table is full

If a table-full error occurs, it may be that the disk is full or that the table has reached its maximum size. The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. See [Section E.10.3, “Limits on Table Size”](#).

This error can occur sometimes for MySQL Cluster tables even when there appears to be more than sufficient data memory available. See the documentation for the [DataMemory \[2043\]](#) MySQL Cluster data node configuration parameter, as well as [Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#), for more information.

C.5.2.13. Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start `mysqld` with the `--tmpdir [466]` option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See [Section 4.2.3.3, “Using Option Files”](#).

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir [568]` directory.

Check also the error code that you get with `perror`. One reason the server cannot write to a table is that the file system is full:

```
shell> perror 28
OS error code 28: No space left on device
```

If you get an error of the following type during startup, it indicates that the file system or directory used for storing data files is write protected. Providing the write error is to a test file, This error is not serious and can be safely ignored.

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

C.5.2.14. Commands out of sync

If you get `Commands out of sync; you can't run this command now` in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

C.5.2.15. Ignoring user

If you get the following error, it means that when `mysqld` was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

```
Found wrong password for user 'some_user'@'some_host'; ignoring user
```

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of `mysqld` with an old `user` table. You can check this by executing `mysqlshow mysql user` to see whether the `Password` column is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.
- The account has an old password (eight characters long). Update the account in the `user` table to have a new password.
- You have specified a password in the `user` table without using the `PASSWORD()` [1188] function. Use `mysql` to update the account in the `user` table with a new password, making sure to use the `PASSWORD()` [1188] function:

```
mysql> UPDATE user SET Password=PASSWORD('newpwd')
-> WHERE User='some_user' AND Host='some_host';
```

C.5.2.16. Table '`tbl_name`' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a file system that has case-sensitive file names.
- Even for file systems that are not case sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See [Section 13.7.5, “SHOW Syntax”](#).

C.5.2.17. Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multi-byte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running `CMake` with the `-DDEFAULT_CHARSET=charset_name` [154] or `-DWITH_EXTRA_CHARSETS=charset_name` [155] option. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

All standard MySQL binaries are compiled with `-DWITH_EXTRA_CHARSETS=all` [155], which enables support for all multi-byte character sets. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

- The character set is a simple character set that is not compiled into `mysqld`, and the character set definition files are not in the place where the client expects to find them.

In this case, you need to use one of the following methods to solve the problem:

- Recompile the client with support for the character set. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

- Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
- Copy the character definition files to the path where the client expects them to be.

C.5.2.18. 'File' Not Found and Similar Errors

If you get `ERROR '...' not found (errno: 23), Can't open file: ... (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you haven't allocated enough file descriptors for the MySQL server. You can use the `pererror` utility to get a description of what the error number means:

```
shell> pererror 23
OS error code 23: File table overflow
shell> pererror 24
OS error code 24: Too many open files
shell> pererror 11
OS error code 11: Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_open_cache` [561] system variable (the default value is 64). This may not entirely prevent running out of file descriptors because in some circumstances the server may attempt to extend the cache size temporarily, as described in [Section 8.4.3.1, "How MySQL Opens and Closes Tables"](#). Reducing the value of `max_connections` [519] also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to `mysqld`, you can use the `--open-files-limit` [257] option to `mysqld_safe` or set the `open_files_limit` [535] system variable. See [Section 5.1.4, "Server System Variables"](#). The easiest way to set these values is to add an option to your option file. See [Section 4.2.3.3, "Using Option Files"](#). If you have an old version of `mysqld` that does not support setting the open files limit, you can edit the `mysqld_safe` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the `"#"` character to uncomment this line, and change the number 256 to set the number of file descriptors to be made available to `mysqld`.

`--open-files-limit` [257] and `ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a "hard" limit that can be overridden only if you start `mysqld_safe` or `mysqld` as `root` (just remember that you also need to start the server with the `--user` [466] option in this case so that it does not continue to run as `root` after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.



Note

If you run the `tcsh` shell, `ulimit` does not work! `tcsh` also reports incorrect values when you ask for the current limits. In this case, you should start `mysqld_safe` using `sh`.

C.5.2.19. Table-Corruption Issues

If you have started `mysqld` with `--myisam-recover-options` [454], MySQL automatically checks and tries to repair MyISAM tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without

`mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further.

As of MySQL 5.5.3, when the server detects `MyISAM` table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: `Got an error from thread_id=1, mi_dynrec.c:368`. This is useful information to include in bug reports.

See also [Section 5.1.3, “Server Command Options”](#), and [Section 23.4.1.7, “Making a Test Case If You Experience Table Corruption”](#).

C.5.3. Installation-Related Issues

C.5.3.1. Problems with File Permissions

If you have problems with file permissions, the `UMASK` environment variable might be set incorrectly when `mysqld` starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default `UMASK` value is `0660`. You can change this behavior by starting `mysqld_safe` as follows:

```
shell> UMASK=384 # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

By default, MySQL creates database directories with an access permission value of `0700`. You can modify this behavior by setting the `UMASK_DIR` variable. If you set its value, new directories are created with the combined `UMASK` and `UMASK_DIR` values. For example, if you want to give group access to all new directories, you can do this:

```
shell> UMASK_DIR=504 # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

MySQL assumes that the value for `UMASK` or `UMASK_DIR` is in octal if it starts with a zero.

See [Section 2.13, “Environment Variables”](#).

C.5.4. Administration-Related Issues

C.5.4.1. How to Reset the Root Password

If you have never set a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, this is insecure. For instructions on assigning passwords, see [Section 2.11.2, “Securing the Initial MySQL Accounts”](#).

If you know the `root` password, but want to change it, see [Section 13.7.1.6, “SET PASSWORD Syntax”](#).

If you set a `root` password previously, but have forgotten it, you can set a new password. The following sections provide instructions for Windows and Unix systems, as well as generic instructions that apply to any system.

C.5.4.1.1. Resetting the Root Password: Windows Systems

On Windows, use the following procedure to reset the password for all MySQL `root` accounts:

1. Log on to your system as Administrator.

2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager: From the [Start](#) menu, select Control Panel, then Administrative Tools, then Services. Find the MySQL service in the list and stop it.

If your server is not running as a service, you may need to use the Task Manager to force it to stop.

3. Create a text file containing the following statements. Replace the password with the password that you want to use.

```
UPDATE mysql.user SET Password=PASSWORD('MyNewPass') WHERE User='root';
FLUSH PRIVILEGES;
```

Write the `UPDATE` and `FLUSH` statements each on a single line. The `UPDATE` statement resets the password for all `root` accounts, and the `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.

4. Save the file. For this example, the file will be named `C:\mysql-init.txt`.
5. Open a console window to get to the command prompt: From the [Start](#) menu, select Run, then enter `cmd` as the command to be run.
6. Start the MySQL server with the special `--init-file` [\[445\]](#) option (notice that the backslash in the option value is doubled):

```
C:\> C:\mysql\bin\mysqld --init-file=C:\mysql-init.txt
```

If you installed MySQL to a location other than `C:\mysql`, adjust the command accordingly.

The server executes the contents of the file named by the `--init-file` [\[445\]](#) option at startup, changing each `root` account password.

You can also add the `--console` [\[438\]](#) option to the command if you want server output to appear in the console window rather than in a log file.

If you installed MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` [\[248\]](#) option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld.exe"
--defaults-file="C:\Program Files\MySQL\MySQL Server 5.5\my.ini"
--init-file=C:\mysql-init.txt
```

The appropriate `--defaults-file` [\[248\]](#) setting can be found using the Services Manager: From the [Start](#) menu, select Control Panel, then Administrative Tools, then Services. Find the MySQL service in the list, right-click it, and choose the `Properties` option. The `Path to executable` field contains the `--defaults-file` [\[248\]](#) setting.

7. After the server has started successfully, delete `C:\mysql-init.txt`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the MySQL server, then restart it in normal mode again. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.

C.5.4.1.2. Resetting the Root Password: Unix Systems

On Unix, use the following procedure to reset the password for all MySQL `root` accounts. The instructions assume that you will start the server so that it runs using the Unix login account that you normally use for running the server. For example, if you run the server using the `mysql` login account, you should log in as `mysql` before using the instructions. Alternatively, you can log in as `root`, but in this case you *must* start `mysqld` with the `--user=mysql` [\[466\]](#) option. If you start the server as `root` without using `--`

`user=mysql` [466], the server may create `root`-owned files in the data directory, such as log files, and these may cause permission-related problems for future server startups. If that happens, you will need to either change the ownership of the files to `mysql` or remove them.

1. Log on to your system as the Unix user that the `mysqld` server runs as (for example, `mysql`).
2. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, host name, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the file name has an extension of `.pid` and begins with either `mysqld` or your system's host name.

You can stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process, using the path name of the `.pid` file in the following command:

```
shell> kill `cat /mysql-data-directory/host_name.pid`
```

Use backticks (not forward quotation marks) with the `cat` command. These cause the output of `cat` to be substituted into the `kill` command.

3. Create a text file containing the following statements. Replace the password with the password that you want to use.

```
UPDATE mysql.user SET Password=PASSWORD('MyNewPass') WHERE User='root';
FLUSH PRIVILEGES;
```

Write the `UPDATE` and `FLUSH` statements each on a single line. The `UPDATE` statement resets the password for all `root` accounts, and the `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.

4. Save the file. For this example, the file will be named `/home/me/mysql-init`. The file contains the password, so it should not be saved where it can be read by other users. If you are not logged in as `mysql` (the user the server runs as), make sure that the file has permissions that permit `mysql` to read it.
5. Start the MySQL server with the special `--init-file` [445] option:

```
shell> mysqld_safe --init-file=/home/me/mysql-init &
```

The server executes the contents of the file named by the `--init-file` [445] option at startup, changing each `root` account password.

6. After the server has started successfully, delete `/home/me/mysql-init`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally.

C.5.4.1.3. Resetting the Root Password: Generic Instructions

The preceding sections provide password-resetting instructions for Windows and Unix systems. Alternatively, on any platform, you can set the new password using the `mysql` client (but this approach is less secure):

1. Stop `mysqld` and restart it with the `--skip-grant-tables` [461] option. This enables anyone to connect without a password and with all privileges. Because this is insecure, you might want to use `--skip-grant-tables` [461] in conjunction with `--skip-networking` [461] to prevent remote clients from connecting.
2. Connect to the `mysqld` server with this command:

```
shell> mysql
```

3. Issue the following statements in the `mysql` client. Replace the password with the password that you want to use.

```
mysql> UPDATE mysql.user SET Password=PASSWORD('MyNewPass')
-> WHERE User='root';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server, then restart it normally (without the `--skip-grant-tables` [461] and `--skip-networking` [461] options).

C.5.4.2. What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This does not mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See [Section 5.2.2, "The Error Log"](#).

On some systems, you can find in the error log a stack trace of where `mysqld` died that you can resolve with the `resolve_stack_dump` program. See [MySQL Internals: Porting to Other Systems](#). Note that the variable values written in the error log may not always be 100% correct.

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with `--delay-key-write` [441], in which case data files are written but not index files.) This means that data file contents are safe even if `mysqld` crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting `mysqld` with the `--flush` [444] option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as `mysqld` without locking the table properly.
- You are running many `mysqld` servers using the same data directory on a system that does not support good file system locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused `mysqld`.
- You have found a bug in the data storage code. This isn't likely, but it is at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Please try the following things:

- Stop the `mysqld` server with `mysqladmin shutdown`, run `myisamchk --silent --force */*.MYI` from the data directory to check all MyISAM tables, and restart `mysqld`. This ensures that you are running from a clean state. See [Chapter 5, MySQL Server Administration](#).
- Start `mysqld` with the general query log enabled (see [Section 5.2.3, “The General Query Log”](#)). Then try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See [Section 5.2.3, “The General Query Log”](#). If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have been able to locate the bug and should submit a bug report for it. See [Section 1.7, “How to Report Bugs or Problems”](#).
- Try to make a test case that we can use to repeat the problem. See [MySQL Internals: Porting to Other Systems](#).
- Try running the tests in the `mysql-test` directory and the MySQL benchmarks. See [Section 23.1.2, “The MySQL Test Suite”](#). They should test MySQL rather well. You can also add code to the benchmarks that simulates your application. The benchmarks can be found in the `sql-bench` directory in a source distribution or, for a binary distribution, in the `sql-bench` directory under your MySQL installation directory.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- If you configure MySQL for debugging, it is much easier to gather information about possible errors if something goes wrong. Reconfigure MySQL with the `-DWITH_DEBUG=1 [155]` option to `CMake` and then recompile. See [MySQL Internals: Porting to Other Systems](#).
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking [460]` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking [460]` option tells `mysqld` not to use external locking. (This means that you cannot run two `mysqld` servers on the same data directory and that you must be careful if you use `myisamchk`. Nevertheless, it may be instructive to try the option as a test.)
- Have you tried `mysqladmin -u root processlist` when `mysqld` appears to be running but not responding? Sometimes `mysqld` is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin -u root processlist` usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.
- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` in a separate window to produce statistics while you run your other queries.
- Try the following:
 1. Start `mysqld` from `gdb` (or another debugger). See [MySQL Internals: Porting to Other Systems](#).
 2. Run your test scripts.
 3. Print the backtrace and the local variables at the three lowest levels. In `gdb`, you can do this with the following commands when `mysqld` has crashed inside `gdb`:

```
backtrace
```

```
info local
up
info local
up
info local
```

With `gdb`, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where `N` is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See [Section 1.7, “How to Report Bugs or Problems”](#). Be even more detailed than usual. Because MySQL works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

The current dynamic row code has been in use for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.

- Do not rule out your server hardware when diagnosing problems. Defective hardware can be the cause of data corruption. Particular attention should be paid to your memory and disk subsystems when troubleshooting hardware.

C.5.4.3. How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This section is relevant for writes to `MyISAM` tables. It also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition enables the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA INFILE` or after an `ALTER TABLE` statement. All of these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes

the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table is left unchanged.

C.5.4.4. Where MySQL Stores Temporary Files

On Unix, MySQL uses the value of the `TMPDIR` environment variable as the path name of the directory in which to store temporary files. If `TMPDIR` is not set, MySQL uses the system default, which is usually `/tmp`, `/var/tmp`, or `/usr/tmp`.

On Windows, MySQL checks in order the values of the `TMPDIR`, `TEMP`, and `TMP` environment variables. For the first one found to be set, MySQL uses it and does not check those remaining. If none of `TMPDIR`, `TEMP`, or `TMP` are set, MySQL uses the Windows system default, which is usually `C:\windows\temp\`.

If the file system containing your temporary file directory is too small, you can use the `--tmpdir [466]` option to `mysqld` to specify a directory in a file system where you have enough space.

In MySQL 5.5, the `--tmpdir [466]` option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (":") on Unix and semicolon characters(";") on Windows.



Note

To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you should not set `--tmpdir [466]` to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

MySQL creates all temporary files as hidden files. This ensures that the temporary files are removed if `mysqld` is terminated. The disadvantage of using hidden files is that you do not see a big temporary file that fills up the file system in which the temporary file directory is located.

When sorting (`ORDER BY` or `GROUP BY`), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some `SELECT` queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form `SQL_*`.

`ALTER TABLE` creates a temporary table in the same directory as the original table.

C.5.4.5. How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On some versions of Unix, anyone can delete files in the `/tmp` directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your `/tmp` directory so that files can be deleted only by their owners or the superuser (`root`). To do this, set the `sticky` bit on the `/tmp` directory by logging in as `root` and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the `sticky` bit is set by executing `ls -ld /tmp`. If the last permission character is `t`, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in `/etc/my.cnf`:

```
[mysqld]
socket=/path/to/socket

[client]
socket=/path/to/socket
```

See [Section 4.2.3.3, “Using Option Files”](#).

- Specify a `--socket [240]` option on the command line to `mysqld_safe` and when you run client programs.
- Set the `MYSQL_UNIX_PORT` environment variable to the path of the Unix socket file.
- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the `MYSQL_UNIX_ADDR [155]` option when you run `CMake`. See [Section 2.10.4, “MySQL Source-Configuration Options”](#).

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

C.5.4.6. Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if `UNIX_TIMESTAMP()` [\[1150\]](#) returns the wrong value. This should be done for the environment in which the server runs; for example, in `mysqld_safe` or `mysql.server`. See [Section 2.13, “Environment Variables”](#).

You can set the time zone for the server with the `--timezone=timezone_name [258]` option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`.

The permissible values for `--timezone [258]` or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

C.5.5. Query-Related Issues

C.5.5.1. Case Sensitivity in String Searches

For nonbinary strings (`CHAR`, `VARCHAR`, `TEXT`), string searches use the collation of the comparison operands. For binary strings (`BINARY`, `VARBINARY`, `BLOB`), comparisons use the numeric values of the bytes in the operands; this means that for alphabetic characters, comparisons will be case sensitive.

A comparison between a nonbinary string and binary string is treated as a comparison of binary strings.

Simple comparison operations (\geq , $>$, $=$, $<$, \leq , sorting, and grouping) are based on each character's "sort value." Characters with the same sort value are treated as the same character. For example, if "e" and "é" have the same sort value in a given collation, they compare as equal.

The default character set and collation are `latin1` and `latin1_swedish_ci`, so nonbinary string comparisons are case insensitive by default. This means that if you search with `col_name LIKE 'a %'`, you get all column values that start with `A` or `a`. To make this search case sensitive, make sure that one of the operands has a case sensitive or binary collation. For example, if you are comparing a column and a string that both have the `latin1` character set, you can use the `COLLATE` operator to cause either operand to have the `latin1_general_cs` or `latin1_bin` collation:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation. See [Section 13.1.17, "CREATE TABLE Syntax"](#).

To cause a case-sensitive comparison of nonbinary strings to be case insensitive, use `COLLATE` to name a case-insensitive collation. The strings in the following example normally are case sensitive, but `COLLATE` changes the comparison to be case insensitive:

```
mysql> SET @s1 = 'MySQL' COLLATE latin1_bin,
->      @s2 = 'mysql' COLLATE latin1_bin;
mysql> SELECT @s1 = @s2;
+-----+
| @s1 = @s2 |
+-----+
|          0 |
+-----+
mysql> SELECT @s1 COLLATE latin1_swedish_ci = @s2;
+-----+
| @s1 COLLATE latin1_swedish_ci = @s2 |
+-----+
|                                     1 |
+-----+
```

A binary string is case sensitive in comparisons. To compare the string as case insensitive, convert it to a nonbinary string and use `COLLATE` to name a case-insensitive collation:

```
mysql> SET @s = BINARY 'MySQL';
mysql> SELECT @s = 'mysql';
+-----+
| @s = 'mysql' |
+-----+
|          0 |
+-----+
mysql> SELECT CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql';
+-----+
| CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql' |
+-----+
|                                     1 |
+-----+
```

To determine whether a value will compare as a nonbinary or binary string, use the `COLLATION()` [1191] function. This example shows that `VERSION()` [1198] returns a string that has a case-insensitive collation, so comparisons are case insensitive:

```
mysql> SELECT COLLATION(VERSION());
```

```

+-----+
| COLLATION(VERSION()) |
+-----+
| utf8_general_ci      |
+-----+

```

For binary strings, the collation value is `binary`, so comparisons will be case sensitive. One context in which you will see `binary` is for compression and encryption functions, which return binary strings as a general rule: string:

```

mysql> SELECT COLLATION(ENCRYPT('x')), COLLATION(SHA1('x'));
+-----+-----+
| COLLATION(ENCRYPT('x')) | COLLATION(SHA1('x')) |
+-----+-----+
| binary                 | binary                |
+-----+-----+

```

C.5.5.2. Problems Using `DATE` Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is permitted. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
SELECT * FROM t1 WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context and vice versa. MySQL also permits a “relaxed” string format when updating and in a `WHERE` clause that compares a date to a `DATE`, `DATETIME`, or `TIMESTAMP` column. “Relaxed” format means that any punctuation character may be used as the separator between parts. For example, `'2004-08-15'` and `'2004#08#15'` are equivalent. MySQL can also convert a string containing no separators (such as `'20040815'`), provided it makes sense as a date.

When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` to a constant string with the `<`, `<=`, `=`, `>=`, `>`, or `BETWEEN` operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns
- When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` column to an expression
- When you use any comparison method other than those just listed, such as `IN` or `STRCMP()` [1113].

For those exceptions, the comparison is done by converting the objects to strings and performing a string comparison.

To be on the safe side, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special “zero” date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'`. When a `'0000-00-00'` date is used through Connector/ODBC, it is automatically converted to `NULL` because ODBC cannot handle that kind of date.

Because MySQL performs the conversions just described, the following statements work (assume that `idate` is a `DATE` column):

```

INSERT INTO t1 (idate) VALUES (19970505);
INSERT INTO t1 (idate) VALUES ('19970505');
INSERT INTO t1 (idate) VALUES ('97-05-05');
INSERT INTO t1 (idate) VALUES ('1997.05.05');

```

```

INSERT INTO t1 (idate) VALUES ('1997 05 05');
INSERT INTO t1 (idate) VALUES ('0000-00-00');

SELECT idate FROM t1 WHERE idate >= '1997-05-05';
SELECT idate FROM t1 WHERE idate >= 19970505;
SELECT MOD(idate,100) FROM t1 WHERE idate >= 19970505;
SELECT idate FROM t1 WHERE idate >= '19970505';

```

However, the following statement does not work:

```

SELECT idate FROM t1 WHERE STRCMP(idate,'20030505')=0;

```

`STRCMP()` [1113] is a string function, so it converts `idate` to a string in 'YYYY-MM-DD' format and performs a string comparison. It does not convert '20030505' to the date '2003-05-05' and perform a date comparison.

If you enable the `ALLOW_INVALID_DATES` [612] SQL mode, MySQL permits you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12. This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

MySQL permits you to store dates where the day or month and day are zero. This is convenient if you want to store a birthdate in a `DATE` column and you know only part of the date. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` [615] SQL mode.

MySQL permits you to store a “zero” value of '0000-00-00' as a “dummy date.” This is in some cases more convenient than using `NULL` values. If a date to be stored in a `DATE` column cannot be converted to any reasonable value, MySQL stores '0000-00-00'. To disallow '0000-00-00', enable the `NO_ZERO_DATE` [615] SQL mode.

To have MySQL check all dates and accept only legal dates (unless overridden by `IGNORE`), set the `sql_mode` [556] system variable to "NO_ZERO_IN_DATE,NO_ZERO_DATE".

C.5.5.3. Problems with `NULL` Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string ''. This is not the case. For example, the following statements are completely different:

```

mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');

```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as “phone number is not known” and the meaning of the second can be regarded as “the person is known to have no phone, and thus no phone number.”

To help with `NULL` handling, you can use the `IS NULL` [1091] and `IS NOT NULL` [1091] operators and the `IFNULL()` [1098] function.

In SQL, the `NULL` value is never true in comparison to any other value, even `NULL`. An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return `NULL`:

```

mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);

```

To search for column values that are `NULL`, you cannot use an `expr = NULL` test. The following statement returns no rows, because `expr = NULL` is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for `NULL` values, you must use the `IS NULL` [1091] test. The following statements show how to find the `NULL` phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

See [Section 3.3.4.6, “Working with `NULL` Values”](#), for additional information and examples.

You can add an index on a column that can have `NULL` values if you are using the `MyISAM`, `InnoDB`, or `MEMORY` storage engine. Otherwise, you must declare an indexed column `NOT NULL`, and you cannot insert `NULL` into the column.

When reading data with `LOAD DATA INFILE`, empty or missing columns are updated with `' '`. To load a `NULL` value into a column, use `\N` in the data file. The literal word “`NULL`” may also be used under some circumstances. See [Section 13.2.6, “LOAD DATA INFILE Syntax”](#).

When using `DISTINCT`, `GROUP BY`, or `ORDER BY`, all `NULL` values are regarded as equal.

When using `ORDER BY`, `NULL` values are presented first, or last if you specify `DESC` to sort in descending order.

Aggregate (summary) functions such as `COUNT()` [1204], `MIN()` [1206], and `SUM()` [1206] ignore `NULL` values. The exception to this is `COUNT(*)` [1204], which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-`NULL` values in the `age` column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles `NULL` values specially. If you insert `NULL` into a `TIMESTAMP` column, the current date and time is inserted. If you insert `NULL` into an integer or floating-point column that has the `AUTO_INCREMENT` attribute, the next number in the sequence is inserted.

C.5.5.4. Problems with Column Aliases

An alias can be used in a query select list to give a column a different name. You can use the alias in `GROUP BY`, `ORDER BY`, or `HAVING` clauses to refer to the column:

```
SELECT SQRT(a*b) AS root FROM tbl_name
GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name
GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL disallows references to column aliases in a `WHERE` clause. This restriction is imposed because when the `WHERE` clause is evaluated, the column value may not yet have been determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name
WHERE cnt > 0 GROUP BY id;
```

The `WHERE` clause determines which rows should be included in the `GROUP BY` clause, but it refers to the alias of a column value that is not known until after the rows have been selected, and grouped by the `GROUP BY`.

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
SELECT 1 AS `one`, 2 AS 'two';
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal. For example, this statement groups by the values in column `id`, referenced using the alias ``a``:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
GROUP BY `a`;
```

But this statement groups by the literal string `'a'` and will not work as expected:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
GROUP BY 'a';
```

C.5.5.5. Rollback Failure for Nontransactional Tables

If you receive the following message when trying to perform a `ROLLBACK`, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These nontransactional tables are not affected by the `ROLLBACK` statement.

If you were not deliberately mixing transactional and nontransactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your `mysqld` server (or that was disabled with a startup option). If `mysqld` does not support a storage engine, it instead creates the table as a `MyISAM` table, which is nontransactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See [Section 13.7.5.37](#), “`SHOW TABLE STATUS Syntax`”, and [Section 13.7.5.12](#), “`SHOW CREATE TABLE Syntax`”.

You can check which storage engines your `mysqld` server supports by using this statement:

```
SHOW ENGINES;
```

You can also use the following statement, and check the value of the variable that is associated with the storage engine in which you are interested:

```
SHOW VARIABLES LIKE 'have_%';
```

For example, to determine whether the `InnoDB` storage engine is available, check the value of the `have_innodb` [\[504\]](#) variable.

See [Section 13.7.5.17](#), “`SHOW ENGINES Syntax`”, and [Section 13.7.5.40](#), “`SHOW VARIABLES Syntax`”.

C.5.5.6. Deleting Rows from Related Tables

If the total length of the `DELETE` statement for `related_table` is more than 1MB (the default value of the `max_allowed_packet` [\[519\]](#) system variable), you should split it into smaller parts and execute multiple `DELETE` statements. You probably get the fastest `DELETE` by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the `IN` clause.

C.5.5.7. Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that returns no rows, you should use the following procedure to find out what is wrong:

1. Test the query with `EXPLAIN` to check whether you can find something that is obviously wrong. See [Section 13.8.2, “EXPLAIN Syntax”](#).
2. Select only those columns that are used in the `WHERE` clause.
3. Remove one table at a time from the query until it returns some rows. If the tables are large, it is a good idea to use `LIMIT 10` with the query.
4. Issue a `SELECT` for the column that should have matched a row against the table that was last removed from the query.
5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you cannot use equality (`=`) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the `FLOAT` to a `DOUBLE` fixes this. See [Section C.5.5.8, “Problems with Floating-Point Values”](#).
6. If you still cannot figure out what is wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql`. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your `SELECT` statement at the end of the file.

Verify that the test file demonstrates the problem by executing these commands:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Attach the test file to a bug report, which you can file using the instructions in [Section 1.7, “How to Report Bugs or Problems”](#).

C.5.5.8. Problems with Floating-Point Values

Floating-point numbers sometimes cause confusion because they are approximate and not stored as exact values. A floating-point value as written in an SQL statement may not be the same as the value represented internally. Attempts to treat floating-point values as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. The `FLOAT` and `DOUBLE` data types are subject to these issues. For `DECIMAL` columns, MySQL performs operations with a precision of 65 decimal digits, which should solve most common inaccuracy problems.

The following example uses `DOUBLE` to demonstrate how calculations that are done using floating-point operations are subject to floating-point error.

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
-> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
-> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
-> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
-> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
-> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

```
+-----+-----+-----+
```

| i | a | b |
|---|-------|------|
| 1 | 21.4 | 21.4 |
| 2 | 76.8 | 76.8 |
| 3 | 7.4 | 7.4 |
| 4 | 15.4 | 15.4 |
| 5 | 7.2 | 7.2 |
| 6 | -51.4 | 0 |

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of `a` and `b` do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

If columns `d1` and `d2` had been defined as `DECIMAL` rather than `DOUBLE`, the result of the `SELECT` query would have contained only one row—the last one shown above.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) > 0.0001;
+-----+-----+-----+
| i     | a     | b     |
+-----+-----+-----+
| 6     | -51.4 | 0     |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+-----+-----+-----+
| i     | a     | b     |
+-----+-----+-----+
1	21.4	21.4
2	76.8	76.8
3	7.4	7.4
4	15.4	15.4
5	7.2	7.2
+-----+-----+-----+
5 rows in set (0.03 sec)
```

Floating-point values are subject to platform or implementation dependencies. Suppose that you execute the following statements:

```
CREATE TABLE t1(c1 FLOAT(53,0), c2 FLOAT(53,0));
INSERT INTO t1 VALUES('1e+52', '-1e+52');
SELECT * FROM t1;
```

On some platforms, the `SELECT` statement returns `inf` and `-inf`. On others, it returns `0` and `-0`.

An implication of the preceding issues is that if you attempt to create a replication slave by dumping table contents with `mysqldump` on the master and reloading the dump file into the slave, tables containing floating-point columns might differ between the two hosts.

C.5.6. Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL does not have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the “right” thing, tools that you have available to help MySQL are:

- Use the `EXPLAIN` statement to get information about how MySQL processes a query. To use it, just add the keyword `EXPLAIN` to the front of your `SELECT` statement:

```
mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
```

`EXPLAIN` is discussed in more detail in [Section 13.8.2, “EXPLAIN Syntax”](#).

- Use `ANALYZE TABLE tbl_name` to update the key distributions for the scanned table. See [Section 13.7.2.1, “ANALYZE TABLE Syntax”](#).
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

`USE INDEX` and `IGNORE INDEX` may also be useful. See [Section 13.2.9.3, “Index Hint Syntax”](#).

- Global and table-level `STRAIGHT_JOIN`. See [Section 13.2.9, “SELECT Syntax”](#).
- You can tune global or thread-specific system variables. For example, start `mysqld` with the `--max-seeks-for-key=1000` [524] option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See [Section 5.1.4, “Server System Variables”](#).

C.5.7. Table Definition-Related Issues

C.5.7.1. Problems with `ALTER TABLE`

If you get a duplicate-key error when using `ALTER TABLE` to change the character set or collation of a character column, the cause is either that the new column collation maps two keys to the same value or that the table is corrupted. In the latter case, you should run `REPAIR TABLE` on the table.

If `ALTER TABLE` dies with the following error, the problem may be that MySQL crashed during an earlier `ALTER TABLE` operation and there is an old table named `A-xxx` or `B-xxx` lying around:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with `A-` or `B-`. (You may want to move them elsewhere instead of deleting them.)

`ALTER TABLE` works in the following way:

- Create a new table named `A-xxx` with the requested structural changes.
- Copy all rows from the original table to `A-xxx`.
- Rename the original table to `B-xxx`.
- Rename `A-xxx` to your original table name.

- Delete `B-xxx`.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as `B-xxx`. A simple rename of the table files at the system level should get your data back.

If you use `ALTER TABLE` on a transactional table or if you are using Windows, `ALTER TABLE` unlocks the table if you had done a `LOCK TABLE` on it. This is done because `InnoDB` and these operating systems cannot drop a table that is in use.

C.5.7.2. TEMPORARY Table Problems

The following list indicates limitations on the use of `TEMPORARY` tables:

- A `TEMPORARY` table can only be of type `MEMORY`, `MyISAM`, `MERGE`, or `InnoDB`.

Temporary tables are not supported for MySQL Cluster.

- You cannot refer to a `TEMPORARY` table more than once in the same query. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

This error also occurs if you refer to a temporary table multiple times in a stored function under different aliases, even if the references occur in different statements within the function.

- The `SHOW TABLES` statement does not list `TEMPORARY` tables.
- You cannot use `RENAME` to rename a `TEMPORARY` table. However, you can use `ALTER TABLE` instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

- There are known issues in using temporary tables with replication. See [Section 16.4.1, “Replication Features and Issues”](#), for more information.

C.5.8. Known Issues in MySQL

This section lists known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and porting instructions in [Section 2.1, “General Installation Guidance”](#), and [MySQL Internals: Porting to Other Systems](#).

The following problems are known:

- Subquery optimization for `IN` is not as effective as for `=`.
- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` [1193] or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint does not work in replication because the constraint may have another name on the slave.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` does not work inside `GROUP_CONCAT()` [1205] if you do not use all and only those columns that are in the `DISTINCT` list.

- When inserting a big integer value (between 2^{63} and $2^{64}-1$) into a decimal or string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` may cause problems on tables for which you are using `INSERT DELAYED`.
- With statement-based binary logging, the master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases. However, it is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is nondeterministic (generally not a recommended practice, even outside of replication).

For example:

- `CREATE TABLE ... SELECT` or `INSERT ... SELECT` statements that insert zero or `NULL` values into an `AUTO_INCREMENT` column.
- `DELETE` if you are deleting rows from a table that has foreign keys with `ON DELETE CASCADE` properties.
- `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` if you have duplicate key values in the inserted data.

If and only if the preceding queries have no `ORDER BY` clause guaranteeing a deterministic order.

For example, for `INSERT ... SELECT` with no `ORDER BY`, the `SELECT` may return rows in a different order (which results in a row having different ranks, hence getting a different number in the `AUTO_INCREMENT` column), depending on the choices made by the optimizers on the master and slave.

A query is optimized differently on the master and slave only if:

- The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use `InnoDB` on the master, but `MyISAM` on the slave if the slave has less available disk space.)
- MySQL buffer sizes (`key_buffer_size` [508], and so on) are different on the master and slave.
- The master and slave run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using `mysqlbinlog|mysql`.

The easiest way to avoid this problem is to add an `ORDER BY` clause to the aforementioned nondeterministic queries to ensure that the rows are always stored or modified in the same order. Using row-based or mixed logging format also avoids the problem.

- Log file names are based on the server host name if you do not specify a file name with the startup option. To retain the same log file names if you change your host name to something else, you must explicitly use options such as `--log-bin=old_host_name-bin` [1892]. See [Section 5.1.3, “Server Command Options”](#). Alternatively, rename the old files to reflect your host name change. If these are binary logs, you must edit the binary log index file and fix the binary log file names there as well. (The same is true for the relay logs on a slave server.)
- `mysqlbinlog` does not delete temporary files left after a `LOAD DATA INFILE` statement. See [Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”](#).
- `RENAME` does not work with `TEMPORARY` tables or tables used in a `MERGE` table.

- When using `SET CHARACTER SET`, you cannot use translated characters in database, table, and column names.
- You cannot use “_” or “%” with `ESCAPE` in `LIKE ... ESCAPE [1111]`.
- Only the first `max_sort_length [524]` bytes are used when comparing data values. This means that values cannot reliably be used in `GROUP BY`, `ORDER BY` or `DISTINCT` if they are not distinct in the first `max_sort_length [524]` bytes. To work around this, increase the variable value. The default value of `max_sort_length [524]` is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with `BIGINT` precision, `IF()` [1097] and `ELT()` [1102] with `BIGINT` or `DOUBLE` precision, and the rest with `DOUBLE` precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.
- You can have up to 255 `ENUM` and `SET` columns in one table.
- In `MIN()` [1206], `MAX()` [1205], and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set.
- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by 2, **not 1**:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns do not participate in the `DISTINCT` comparison.

An example of this is:

```
SELECT DISTINCT mp3id FROM band_downloads
WHERE userid = 9 ORDER BY id DESC;
```

and

```
SELECT DISTINCT band_downloads.mp3id
FROM band_downloads,band_mp3
WHERE band_downloads.userid = 9
AND band_mp3.id = band_downloads.mp3id
ORDER BY band_downloads.id DESC;
```

In the second case, using MySQL Server 3.23.x, you may get two identical rows in the result set (because the values in the hidden `id` column may differ).

Note that this happens only for queries that do not have the `ORDER BY` columns in the result.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.
- Creation of a table of type `MERGE` does not check whether the underlying tables are compatible types.

- If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table and then add a normal index on the `MERGE` table, the key order is different for the tables if there was an old, non-`UNIQUE` key in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

Appendix D. MySQL Release Notes

Table of Contents

| | |
|--|------|
| D.1. Changes in MySQL 5.5 (Production) | 4070 |
| D.2. MySQL Connector/ODBC Change History | 4070 |
| D.3. MySQL Connector/Net Change History | 4070 |
| D.4. MySQL Visual Studio Plugin Change History | 4070 |
| D.5. MySQL Connector/J Change History | 4070 |
| D.6. MySQL Connector/C++ Change History | 4070 |
| D.7. MySQL Proxy Change History | 4070 |
| D.8. MySQL for Excel Change History | 4070 |
| D.9. MySQL Installer Change History | 4071 |

This appendix lists the changes from version to version in MySQL through the latest version of MySQL 5.5, which is currently MySQL 5.5.31. We offer a version of the Manual for each series of MySQL releases (5.0, 5.1, and so forth). For information about changes in another release series of the MySQL database software, see the corresponding version of this Manual.

We update these notes as we add new features in the 5.5 series, so that everybody can follow the development process.

We tend to update these notes at the same time we make changes to MySQL. If you find a recent version of MySQL listed here that you cannot find on our download page (<http://dev.mysql.com/downloads/>), it means that the version has not yet been released.

The date mentioned with a release version is the date of the last Bazaar ChangeSet on which the release was based, not the date when the packages were made available. The binaries are usually made available a few days after the date of the tagged ChangeSet, because building and testing all packages takes some time.

The manual included in the source and binary distributions may not be fully accurate when it comes to the release changelog entries, because the integration of the manual happens at build time. For the most up-to-date release changelog, please refer to the online version instead.

Previously, MySQL development proceeded by including a large set of features and moving them over many versions within a release series through several stages of maturity (Alpha, Beta, and so forth). This development model had a disadvantage in that problems with only part of the code could hinder timely release of the whole.

MySQL development now uses a milestone model. The move to this model provides for more frequent milestone releases, with each milestone proceeding through a small number of releases having a focus on a specific subset of thoroughly tested features. Following the releases for one milestone, development proceeds with the next milestone; that is, another small number of releases that focuses on the next small set of features, also thoroughly tested.

MySQL 5.5.0-m2 is the first release for Milestone 2. The new features of this milestone may be considered to be initially of beta quality. For subsequent Milestone 2 releases, we use increasing version numbers (5.5.1 and higher) while continuing to employ the “-m2” suffix. For Milestone 3, we change the suffix to “-m3”. Version designators with “-alpha” or “-beta” suffixes are no used.

You may notice that the MySQL 5.5.0 release is designated as Milestone 2 rather than Milestone 1. This is because MySQL 5.4 was actually designated as Milestone 1, although we had not yet begun referring to milestone numbers as part of version numbers at the time.

D.1. Changes in MySQL 5.5 (Production)

MySQL 5.5 release notes are no longer published in the MySQL 5.5 Reference Manual.

For release notes detailing the changes in each release of MySQL 5.5, see the [MySQL 5.5 Release Notes](#).

D.2. MySQL Connector/ODBC Change History

MySQL Connector/ODBC release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Connector/ODBC are located at [MySQL Connector/ODBC Release Notes](#).

D.3. MySQL Connector/Net Change History

MySQL Connector/Net release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Connector/Net are located at [MySQL Connector/Net Release Notes](#).

D.4. MySQL Visual Studio Plugin Change History

MySQL Visual Studio Plugin release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Visual Studio Plugin are located at [MySQL Connector/Net Release Notes](#).

D.5. MySQL Connector/J Change History

MySQL Connector/J release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Connector/J are located at [MySQL Connector/J Release Notes](#).

D.6. MySQL Connector/C++ Change History

MySQL Connector/C++ release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Connector/C++ are located at [MySQL Connector/C++ Release Notes](#).

D.7. MySQL Proxy Change History

MySQL Proxy release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Proxy are located at [MySQL Proxy Release Notes](#).

D.8. MySQL for Excel Change History

MySQL for Excel release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL for Excel are located at [MySQL for Excel Release Notes](#).

D.9. MySQL Installer Change History

MySQL Installer release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Installer are located at [MySQL Installer Release Notes](#).

Appendix E. Restrictions and Limits

Table of Contents

| | |
|--|------|
| E.1. Restrictions on Stored Programs | 4073 |
| E.2. Restrictions on Condition Handling | 4077 |
| E.3. Restrictions on Server-Side Cursors | 4077 |
| E.4. Restrictions on Subqueries | 4077 |
| E.5. Restrictions on Views | 4079 |
| E.6. Restrictions on XA Transactions | 4081 |
| E.7. Restrictions on Character Sets | 4081 |
| E.8. Restrictions on Performance Schema | 4082 |
| E.9. Restrictions on Pluggable Authentication | 4082 |
| E.10. Limits in MySQL | 4084 |
| E.10.1. Limits of Joins | 4084 |
| E.10.2. Limits on Number of Databases and Tables | 4084 |
| E.10.3. Limits on Table Size | 4085 |
| E.10.4. Table Column-Count and Row-Size Limits | 4086 |
| E.10.5. Windows Platform Limitations | 4088 |

The discussion here describes restrictions that apply to the use of MySQL features such as subqueries or views.

E.1. Restrictions on Stored Programs

These restrictions apply to the features described in [Chapter 19, *Stored Programs and Views*](#).

Some of the restrictions noted here apply to all stored routines; that is, both to stored procedures and stored functions. There are also some [restrictions specific to stored functions](#) but not to stored procedures.

The restrictions for stored functions also apply to triggers. There are also some [restrictions specific to triggers](#).

The restrictions for stored procedures also apply to the `DO` clause of Event Scheduler event definitions. There are also some [restrictions specific to events](#).

SQL Statements Not Permitted in Stored Routines

Stored routines cannot contain arbitrary SQL statements. The following statements are not permitted:

- The locking statements `LOCK TABLES` and `UNLOCK TABLES`.
- `ALTER VIEW`.
- `LOAD DATA` and `LOAD TABLE`.
- SQL prepared statements (`PREPARE`, `EXECUTE`, `DEALLOCATE PREPARE`) can be used in stored procedures, but not stored functions or triggers. Thus, stored functions and triggers cannot use dynamic SQL (where you construct statements as strings and then execute them).
- Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. For a list of statements supported as prepared statements, see [Section 13.5, “SQL Syntax for Prepared Statements”](#). Exceptions are `SIGNAL` and `RESIGNAL`, which are not permissible as prepared statements but are permitted in stored programs.

- Inserts cannot be delayed. `INSERT DELAYED` syntax is accepted, but the statement is handled as a normal `INSERT`.
- Within all stored programs (stored procedures and functions, triggers, and events), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. To begin a transaction in this context, use `START TRANSACTION` instead.

Restrictions for Stored Functions

The following additional statements or operations are not permitted within stored functions. They are permitted within stored procedures, except stored procedures that are invoked from within a stored function or trigger. For example, if you use `FLUSH` in a stored procedure, that stored procedure cannot be called from a stored function or trigger.

- Statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.
- Statements that return a result set. This includes `SELECT` statements that do not have an `INTO var_list` clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. A function can process a result set either with `SELECT ... INTO var_list` or by using a cursor and `FETCH` statements. See [Section 13.2.9.1, “SELECT ... INTO Syntax”](#), and [Section 13.6.6, “Cursors”](#).
- `FLUSH` statements.
- Stored functions cannot be used recursively.
- A stored function or trigger cannot modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.
- If you refer to a temporary table multiple times in a stored function under different aliases, a `Can't reopen table: 'tbl_name'` error occurs, even if the references occur in different statements within the function.
- `HANDLER ... READ` statements that invoke stored functions can cause replication errors. As of MySQL 5.5.7, such statements are disallowed.

Restrictions for Triggers

For triggers, the following additional restrictions apply:

- Triggers currently are not activated by foreign key actions.
- When using row-based replication, triggers on the slave are not activated by statements originating on the master. The triggers on the slave are activated when using statement-based replication. For more information, see [Section 16.4.1.32, “Replication and Triggers”](#).
- The `RETURN` statement is not permitted in triggers, which cannot return a value. To exit a trigger immediately, use the `LEAVE` statement.
- Triggers are not permitted on tables in the `mysql` database.
- The trigger cache does not detect when metadata of the underlying objects has changed. If a trigger uses a table and the table has changed since the trigger was loaded into the cache, the trigger operates using the outdated metadata.

Name Conflicts within Stored Routines

The same identifier might be used for a routine parameter, a local variable, and a table column. Also, the same local variable name can be used in nested blocks. For example:

```
CREATE PROCEDURE p (i INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  SELECT i FROM t;
  BEGIN
    DECLARE i INT DEFAULT 1;
    SELECT i FROM t;
  END;
END;
```

In such cases, the identifier is ambiguous and the following precedence rules apply:

- A local variable takes precedence over a routine parameter or table column.
- A routine parameter takes precedence over a table column.
- A local variable in an inner block takes precedence over a local variable in an outer block.

The behavior that variables take precedence over table columns is nonstandard.

Replication Considerations

Use of stored routines can cause replication problems. This issue is discussed further in [Section 19.7](#), “Binary Logging of Stored Programs”.

The `--replicate-wild-do-table=db_name.tbl_name` [1879] option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Debugging Considerations

There are no stored routine debugging facilities.

Unsupported Syntax from the SQL:2003 Standard

The MySQL stored routine syntax is based on the SQL:2003 standard. The following items from that standard are not currently supported:

- `UNDO` handlers
- `FOR` loops

Concurrency Considerations

To prevent problems of interaction between sessions, when a client issues a statement, the server uses a snapshot of routines and triggers available for execution of the statement. That is, the server calculates a list of procedures, functions, and triggers that may be used during execution of the statement, loads them, and then proceeds to execute the statement. While the statement executes, it does not see changes to routines performed by other sessions.

For maximum concurrency, stored functions should minimize their side-effects; in particular, updating a table within a stored function can reduce concurrent operations on that table. A stored function acquires table locks before executing, to avoid inconsistency in the binary log due to mismatch of the order in which statements execute and when they appear in the log. When statement-based binary logging is used, statements that invoke a function are recorded rather than the statements executed within the function. Consequently, stored functions that update the same underlying tables do not execute in parallel. In contrast, stored procedures do not acquire table-level locks. All statements executed within stored

procedures are written to the binary log, even for statement-based binary logging. See [Section 19.7, “Binary Logging of Stored Programs”](#).

Event Scheduler Restrictions

The following limitations are specific to the Event Scheduler:

- Event names are handled in case-insensitive fashion. For example, you cannot have two events in the same database with the names `anEvent` and `AnEvent`.
- An event may not be created, altered, or dropped by a stored routine, trigger, or another event. An event also may not create, alter, or drop stored routines or triggers. (Bug #16409, Bug #18896)
- As of MySQL 5.5.8, DDL statements on events are prohibited while a `LOCK TABLES` statement is in effect.
- Event timings using the intervals `YEAR`, `QUARTER`, `MONTH`, and `YEAR_MONTH` are resolved in months; those using any other interval are resolved in seconds. There is no way to cause events scheduled to occur at the same second to execute in a given order. In addition—due to rounding, the nature of threaded applications, and the fact that a nonzero length of time is required to create events and to signal their execution—events may be delayed by as much as 1 or 2 seconds. However, the time shown in the `INFORMATION_SCHEMA.EVENTS` table's `LAST_EXECUTED` column or the `mysql.event` table's `last_executed` column is always accurate to within one second of the actual event execution time. (See also Bug #16522.)
- Each execution of the statements contained in the body of an event takes place in a new connection; thus, these statements has no effect in a given user session on the server's statement counts such as `Com_select` and `Com_insert` that are displayed by `SHOW STATUS`. However, such counts *are* updated in the global scope. (Bug #16422)
- Events do not support times later than the end of the Unix Epoch; this is approximately the beginning of the year 2038. Such dates are specifically not permitted by the Event Scheduler. (Bug #16396)
- References to stored functions, user-defined functions, and tables in the `ON SCHEDULE` clauses of `CREATE EVENT` and `ALTER EVENT` statements are not supported. These sorts of references are not permitted. (See Bug #22830 for more information.)
- Generally speaking, statements that are not permitted in stored routines or in SQL prepared statements are also not permitted in the body of an event. For more information, see [Section 13.5, “SQL Syntax for Prepared Statements”](#).

Stored routines and triggers in MySQL Cluster. Stored procedures, stored functions, and triggers are all supported by tables using the `NDB` storage engine; however, it is important to keep in mind that they do *not* propagate automatically between MySQL Servers acting as Cluster SQL nodes. This is because of the following:

- Stored routine definitions are kept in tables in the `mysql` system database using the `MyISAM` storage engine, and so do not participate in clustering.
- The `.TRN` and `.TRG` files containing trigger definitions are not read by the `NDB` storage engine, and are not copied between Cluster nodes.

Any stored routine or trigger that interacts with MySQL Cluster tables must be re-created by running the appropriate `CREATE PROCEDURE`, `CREATE FUNCTION`, or `CREATE TRIGGER` statements on each MySQL Server that participates in the cluster where you wish to use the stored routine or trigger. Similarly, any changes to existing stored routines or triggers must be carried out explicitly on all Cluster SQL nodes, using the appropriate `ALTER` or `DROP` statements on each MySQL Server accessing the cluster.



Warning

Do *not* attempt to work around the issue described in the first item mentioned previously by converting any `mysql` database tables to use the `NDB` storage engine. *Altering the system tables in the `mysql` database is not supported* and is very likely to produce undesirable results.

E.2. Restrictions on Condition Handling

`SIGNAL` and `RESIGNAL` are not permissible as prepared statements. For example, this statement is invalid:

```
PREPARE stmt1 FROM 'SIGNAL SQLSTATE "02000"';
```

`SQLSTATE` values in class `'04'` are not treated specially. They are handled the same as other exceptions.

E.3. Restrictions on Server-Side Cursors

Server-side cursors are implemented in the C API using the `mysql_stmt_attr_set()` function. The same implementation is used for cursors in stored routines. A server-side cursor enables a result set to be generated on the server side, but not transferred to the client except for those rows that the client requests. For example, if a client executes a query but is only interested in the first row, the remaining rows are not transferred.

In MySQL, a server-side cursor is materialized into an internal temporary table. Initially, this is a `MEMORY` table, but is converted to a `MyISAM` table when its size exceeds the minimum value of the `max_heap_table_size` [521] and `tmp_table_size` [568] system variables. Note that the same restrictions apply to internal temporary tables created to hold the result set for a cursor as for other uses of internal temporary tables. See [Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#). One limitation of the implementation is that for a large result set, retrieving its rows through a cursor might be slow.

Cursors are read only; you cannot use a cursor to update rows.

`UPDATE WHERE CURRENT OF` and `DELETE WHERE CURRENT OF` are not implemented, because updatable cursors are not supported.

Cursors are nonholdable (not held open after a commit).

Cursors are asensitive.

Cursors are nonscrollable.

Cursors are not named. The statement handler acts as the cursor ID.

You can have open only a single cursor per prepared statement. If you need several cursors, you must prepare several statements.

You cannot use a cursor for a statement that generates a result set if the statement is not supported in prepared mode. This includes statements such as `CHECK TABLE`, `HANDLER READ`, and `SHOW BINLOG EVENTS`.

E.4. Restrictions on Subqueries

- Subquery optimization for `IN` is not as effective as for the `=` operator or for the `IN(value_list)` [1092] operator.

A typical case for poor `IN` subquery performance is when the subquery returns a small number of rows but the outer query returns a large number of rows to be compared to the subquery result.

The problem is that, for a statement that uses an `IN` subquery, the optimizer rewrites it as a correlated subquery. Consider the following statement that uses an uncorrelated subquery:

```
SELECT ... FROM t1 WHERE t1.a IN (SELECT b FROM t2);
```

The optimizer rewrites the statement to a correlated subquery:

```
SELECT ... FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.b = t1.a);
```

If the inner and outer queries return M and N rows, respectively, the execution time becomes on the order of $O(M \times N)$, rather than $O(M+N)$ as it would be for an uncorrelated subquery.

An implication is that an `IN` subquery can be much slower than a query written using an `IN(value_list)` [1092] operator that lists the same values that the subquery would return.

- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms:

```
DELETE FROM t WHERE ... (SELECT ... FROM t ...);
UPDATE t ... WHERE col = (SELECT ... FROM t ...);
{INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
```

Exception: The preceding prohibition does not apply if you are using a subquery for the modified table in the `FROM` clause. Example:

```
UPDATE t ... WHERE col = (SELECT * FROM (SELECT ... FROM t...) AS _t ...);
```

Here the result from the subquery in the `FROM` clause is stored as a temporary table, so the relevant rows in `t` have already been selected by the time the update to `t` takes place.

- Row comparison operations are only partially supported:
 - For `expr [NOT] IN subquery`, `expr` can be an n -tuple (specified using row constructor syntax) and the subquery can return rows of n -tuples. The permitted syntax is therefore more specifically expressed as `row_constructor [NOT] IN table_subquery`
 - For `expr op {ALL|ANY|SOME} subquery`, `expr` must be a scalar value and the subquery must be a column subquery; it cannot return multiple-column rows.

In other words, for a subquery that returns rows of n -tuples, this is supported:

```
(expr_1, ..., expr_n) [NOT] IN table_subquery
```

But this is not supported:

```
(expr_1, ..., expr_n) op {ALL|ANY|SOME} subquery
```

The reason for supporting row comparisons for `IN` but not for the others is that `IN` is implemented by rewriting it as a sequence of `=` [1089] comparisons and `AND` [1094] operations. This approach cannot be used for `ALL`, `ANY`, or `SOME`.

- Subqueries in the `FROM` clause cannot be correlated subqueries. They are materialized in whole (evaluated to produce a result set) before evaluating the outer query, so they cannot be evaluated per row of the outer query.

- MySQL does not support `LIMIT` in subqueries for certain subquery operators:

```
mysql> SELECT * FROM t1
  ->  WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1);
ERROR 1235 (42000): This version of MySQL doesn't yet support
'LIMIT & IN/ALL/ANY/SOME subquery'
```

- The optimizer is more mature for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as a join.

An exception occurs for the case where an `IN` subquery can be rewritten as a `SELECT DISTINCT` join. Example:

```
SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE condition);
```

That statement can be rewritten as follows:

```
SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND condition;
```

But in this case, the join requires an extra `DISTINCT` operation and is not more efficient than the subquery.

- MySQL permits a subquery to refer to a stored function that has data-modifying side effects such as inserting rows into a table. For example, if `f()` inserts rows, the following query can modify data:

```
SELECT ... WHERE x IN (SELECT f() ...);
```

This behavior is an extension to the SQL standard. In MySQL, it can produce indeterminate results because `f()` might be executed a different number of times for different executions of a given query depending on how the optimizer chooses to handle it.

For statement-based or mixed-format replication, one implication of this indeterminism is that such a query can produce different results on the master and its slaves.

E.5. Restrictions on Views

View processing is not optimized:

- It is not possible to create an index on a view.
- Indexes can be used for views processed using the merge algorithm. However, a view that is processed with the temptable algorithm is unable to take advantage of indexes on its underlying tables (although indexes can be used during generation of the temporary tables).

Subqueries cannot be used in the `FROM` clause of a view.

There is a general principle that you cannot modify a table and select from the same table in a subquery. See [Section E.4, “Restrictions on Subqueries”](#).

The same principle also applies if you select from a view that selects from the table, if the view selects from the table in a subquery and the view is evaluated using the merge algorithm. Example:

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

If the view is evaluated using a temporary table, you *can* select from the table in the view subquery and still modify that table in the outer query. In this case the view will be stored in a temporary table and thus

you are not really selecting from the table in a subquery and modifying it “at the same time.” (This is another reason you might wish to force MySQL to use the temptable algorithm by specifying `ALGORITHM = TEMPTABLE` in the view definition.)

You can use `DROP TABLE` or `ALTER TABLE` to drop or alter a table that is used in a view definition. No warning results from the `DROP` or `ALTER` operation, even though this invalidates the view. Instead, an error occurs later, when the view is used. `CHECK TABLE` can be used to check for views that have been invalidated by `DROP` or `ALTER` operations.

A view definition is “frozen” by certain statements:

- If a statement prepared by `PREPARE` refers to a view, the view definition seen each time the statement is executed later will be the definition of the view at the time it was prepared. This is true even if the view definition is changed after the statement is prepared and before it is executed. Example:

```
CREATE VIEW v AS SELECT RAND();
PREPARE s FROM 'SELECT * FROM v';
ALTER VIEW v AS SELECT NOW();
EXECUTE s;
```

The result returned by the `EXECUTE` statement is a random number, not the current date and time.

With regard to view updatability, the overall goal for views is that if any view is theoretically updatable, it should be updatable in practice. This includes views that have `UNION` in their definition. Currently, not all views that are theoretically updatable can be updated. The initial view implementation was deliberately written this way to get usable, updatable views into MySQL as quickly as possible. Many theoretically updatable views can be updated now, but limitations still exist:

- Updatable views with subqueries anywhere other than in the `WHERE` clause. Some views that have subqueries in the `SELECT` list may be updatable.
- You cannot use `UPDATE` to update more than one underlying table of a view that is defined as a join.
- You cannot use `DELETE` to update a view that is defined as a join.

There exists a shortcoming with the current implementation of views. If a user is granted the basic privileges necessary to create a view (the `CREATE VIEW [696]` and `SELECT [697]` privileges), that user will be unable to call `SHOW CREATE VIEW` on that object unless the user is also granted the `SHOW VIEW [697]` privilege.

That shortcoming can lead to problems backing up a database with `mysqldump`, which may fail due to insufficient privileges. This problem is described in Bug #22062.

The workaround to the problem is for the administrator to manually grant the `SHOW VIEW [697]` privilege to users who are granted `CREATE VIEW [696]`, since MySQL doesn't grant it implicitly when views are created.

Views do not have indexes, so index hints do not apply. Use of index hints when selecting from a view is not permitted.

`SHOW CREATE VIEW` displays view definitions using an `AS alias_name` clause for each column. If a column is created from an expression, the default alias is the expression text, which can be quite long. Aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters). As a result, views created from the output of `SHOW CREATE VIEW` fail if any column alias exceeds 64 characters. This can cause problems in the following circumstances for views with too-long aliases:

- View definitions fail to replicate to newer slaves that enforce the column-length restriction.
- Dump files created with `mysqldump` cannot be loaded into servers that enforce the column-length restriction.

A workaround for either problem is to modify each problematic view definition to use aliases that provide shorter column names. Then the view will replicate properly, and can be dumped and reloaded without causing an error. To modify the definition, drop and create the view again with `DROP VIEW` and `CREATE VIEW`, or replace the definition with `CREATE OR REPLACE VIEW`.

For problems that occur when reloading view definitions in dump files, another workaround is to edit the dump file to modify its `CREATE VIEW` statements. However, this does not change the original view definitions, which may cause problems for subsequent dump operations.

E.6. Restrictions on XA Transactions

XA transaction support is limited to the `InnoDB` storage engine.

For “external XA,” a MySQL server acts as a Resource Manager and client programs act as Transaction Managers. For “Internal XA”, storage engines within a MySQL server act as RMs, and the server itself acts as a TM. Internal XA support is limited by the capabilities of individual storage engines. Internal XA is required for handling XA transactions that involve more than one storage engine. The implementation of internal XA requires that a storage engine support two-phase commit at the table handler level, and currently this is true only for `InnoDB`.

For `XA START`, the `JOIN` and `RESUME` clauses are not supported.

For `XA END`, the `SUSPEND [FOR MIGRATE]` clause is not supported.

The requirement that the `bqual` part of the `xid` value be different for each XA transaction within a global transaction is a limitation of the current MySQL XA implementation. It is not part of the XA specification.

If an XA transaction has reached the `PREPARED` state and the MySQL server is killed (for example, with `kill -9` on Unix) or shuts down abnormally, the transaction can be continued after the server restarts. However, if the client reconnects and commits the transaction, the transaction will be absent from the binary log even though it has been committed. This means the data and the binary log have gone out of synchrony. An implication is that XA cannot be used safely together with replication.

It is possible that the server will roll back a pending XA transaction, even one that has reached the `PREPARED` state. This happens if a client connection terminates and the server continues to run, or if clients are connected and the server shuts down gracefully. (In the latter case, the server marks each connection to be terminated, and then rolls back the `PREPARED` XA transaction associated with it.) It should be possible to commit or roll back a `PREPARED` XA transaction, but this cannot be done without changes to the binary logging mechanism.

E.7. Restrictions on Character Sets

- Identifiers are stored in `mysql` database tables (`user`, `db`, and so forth) using `utf8`, but identifiers can contain only characters in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted in identifiers.
- The `ucs2`, `utf16`, and `utf32` character sets have the following restrictions:
 - They cannot be used as a client character set, which means that they do not work for `SET NAMES` or `SET CHARACTER SET`. (See [Section 10.1.4, “Connection Character Sets and Collations”](#).)

- It is currently not possible to use `LOAD DATA INFILE` to load data files that use these character sets.
- `FULLTEXT` indexes cannot be created on a column that uses any of these character sets. However, you can perform `IN BOOLEAN MODE` searches on the column without an index.
- The use of `ENCRYPT()` [1187] with these character sets is not recommended because the underlying system call expects a string terminated by a zero byte.
- The `REGEXP` [1114] and `RLIKE` [1114] operators work in byte-wise fashion, so they are not multi-byte safe and may produce unexpected results with multi-byte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

E.8. Restrictions on Performance Schema

The Performance Schema avoids using mutexes to collect or produce data, so there are no guarantees of consistency and results can sometimes be incorrect. Event values in `performance_schema` tables are nondeterministic and nonrepeatable.

If you save event information in another table, you should not assume that the original events will still be available later. For example, if you select events from a `performance_schema` table into a temporary table, intending to join that table with the original table later, there might be no matches.

`mysqldump` and `BACKUP DATABASE` ignore tables in the `performance_schema` database.

Tables in the `performance_schema` database cannot be locked with `LOCK TABLES`, except the `setup_XXX` tables.

Tables in the `performance_schema` database cannot be indexed.

Results for queries that refer to tables in the `performance_schema` database are not saved in the query cache.

Tables in the `performance_schema` database are not replicated.

The Performance Schema is not available in `libmysqld`, the embedded server.

The types of timers might vary per platform. The `performance_timers` table shows which event timers are available. If the values in this table for a given timer name are `NULL`, that timer is not supported on your platform.

Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer.

E.9. Restrictions on Pluggable Authentication

The first part of this section describes general restrictions on the applicability of the pluggable authentication framework described at [Section 6.3.6, “Pluggable Authentication”](#). The second part describes how third-party connector developers can determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

The term “native authentication” used here refers to authentication against passwords stored in the `Password` column of the `mysql.user` table. This is the same authentication method provided by MySQL servers older than 5.5.7, before pluggable authentication was implemented. It remains the default method in 5.5.7 and up, although now it is implemented using plugins. “Windows native authentication” refers to

authentication using the credentials of a user who has already logged in to Windows, as implemented by the Windows Native Authentication plugin (“Windows plugin” for short).

General Pluggable Authentication Restrictions

- **Connector/C, Connector/C++:** Clients that use these connectors can connect to the server only through accounts that use native authentication.

Exception: A connector supports pluggable authentication if it was built to link to `libmysql` dynamically (rather than statically) and it loads the current version of `libmysql` if that version is installed, or if the connector is recompiled from source to link against the current `libmysql`.

- **Connector/J:** Clients that use this connector can connect to the server only through accounts that use native authentication.
- **Connector/Net:** Before Connector/Net 6.4.4, clients that use this connector can connect to the server only through accounts that use native authentication. As of 6.4.4, clients can also connect to the server through accounts that use the Windows plugin.
- **Connector/ODBC:** Before Connector/ODBC 3.51.29 and 5.1.9, clients that use this connector can connect to the server only through accounts that use native authentication. As of 3.51.29 and 5.1.9, clients that use binary releases of this connector for Windows can also connect to the server through accounts that use the PAM or Windows plugins. (These capabilities result from linking the Connector/ODBC binaries against the MySQL 5.5.16 `libmysql` rather than the MySQL 5.1 `libmysql` used previously. The newer `libmysql` includes the client-side support needed for the server-side PAM and Windows authentication plugins.)
- **Connector/PHP:** Clients that use this connector can connect to the server only through accounts that use native authentication, when compiled using the MySQL native driver for PHP (`mysqlnd`).
- **MySQL Proxy:** Before MySQL Proxy 0.8.2, clients can connect to the server only through accounts that use native authentication. As of 0.8.2, clients can also connect to the server through accounts that use the PAM plugin. As of 0.8.3, clients can also connect to the server through accounts that use the Windows plugin.
- **MySQL Enterprise Backup:** MySQL Enterprise Backup before version 3.6.1 supports connections to the server only through accounts that use native authentication. As of 3.6.1, MySQL Enterprise Backup can connect to the server through accounts that use nonnative authentication.
- **Windows native authentication:** Connecting through an account that uses the Windows plugin requires Windows Domain setup. Without it, NTLM authentication is used and then only local connections are possible; that is, the client and server must run on the same computer.
- **Proxy users:** Proxy user support is available to the extent that clients can connect through accounts authenticated with plugins that implement proxy user capability (that is, plugins that can return a user name different from that of the connecting user). For example, the native authentication plugins do not support proxy users, whereas the PAM and Windows plugins do.
- **Replication:** Before MySQL 5.5.17, replication slaves can connect to the master server only through master accounts that use native authentication. As of 5.5.17 (or 5.5.19 for Windows native authentication), replication slaves can also connect through master accounts that use nonnative authentication if the required client-side plugin is available. If the plugin is built into `libmysql`, it is available by default. Otherwise, the plugin must be installed on the slave side in the directory named by the slave `plugin_dir` [537] system variable.
- **FEDERATED tables:** A `FEDERATED` table can access the remote table only through accounts on the remote server that use native authentication.

Pluggable Authentication and Third-Party Connectors

Third-party connector developers can use the following guidelines to determine readiness of a connector to take advantage of pluggable authentication capabilities and what steps to take to become more compliant:

- An existing connector to which no changes have been made uses native authentication and clients that use the connector can connect to the server only through accounts that use native authentication. *However, you should test the connector against a recent version of the server to verify that such connections still work without problem.*

Exception: A connector might work with pluggable authentication without any changes if it links to `libmysql` dynamically (rather than statically) and it loads the current version of `libmysql` if that version is installed.

- To take advantage of pluggable authentication capabilities, a connector that is `libmysql`-based should be relinked against the current version of `libmysql`. This enables the connector to support connections through accounts that require client-side plugins now built into `libmysql` (such as the cleartext plugin needed for PAM authentication and the Windows plugin needed for Windows native authentication). Linking with a current `libmysql` also enables the connector to access client-side plugins installed in the default MySQL plugin directory (typically the directory named by the default value of the local server's `plugin_dir` [537] system variable).

If a connector links to `libmysql` dynamically, it must be ensured that the newer version of `libmysql` is installed on the client host and that the connector loads it at runtime.

- Another way for a connector to support a given authentication method is to implement it directly in the client/server protocol. Connector/Net uses this approach to provide support for Windows native authentication.
- If a connector should be able to load client-side plugins from a directory different from the default plugin directory, it must implement some means for client users to specify the directory. Possibilities for this include a command-line option or environment variable from which the connector can obtain the directory name. Standard MySQL client programs such as `mysql` and `mysqladmin` implement a `--plugin-dir` option. See also [Section 22.8.10, “C API Client Plugin Functions”](#).
- Proxy user support by a connector depends, as described earlier in this section, on whether the authentication methods that it supports permit proxy users.

E.10. Limits in MySQL

This section lists current limits in MySQL 5.5.

E.10.1. Limits of Joins

The maximum number of tables that can be referenced in a single join is 61. This also applies to the number of tables that can be referenced in the definition of a view.

E.10.2. Limits on Number of Databases and Tables

MySQL has no limit on the number of databases. The underlying file system may have a limit on the number of directories.

MySQL has no limit on the number of tables. The underlying file system may have a limit on the number of files that represent tables. Individual storage engines may impose engine-specific constraints. `InnoDB` permits up to 4 billion tables.

E.10.3. Limits on Table Size

The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. The following table lists some examples of operating system file-size limits. This is only a rough guide and is not intended to be definitive. For the most up-to-date information, be sure to check the documentation specific to your operating system.

| Operating System | File-size Limit |
|------------------------|------------------------------|
| Win32 w/ FAT/FAT32 | 2GB/4GB |
| Win32 w/ NTFS | 2TB (possibly larger) |
| Linux 2.2-Intel 32-bit | 2GB (LFS: 4GB) |
| Linux 2.4+ | (using ext3 file system) 4TB |
| Solaris 9/10 | 16TB |
| MacOS X w/ HFS+ | 2TB |

Windows users, please note that FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

On Linux 2.2, you can get [MyISAM](#) tables larger than 2GB in size by using the Large File Support (LFS) patch for the ext2 file system. Most current Linux distributions are based on kernel 2.4 or higher and include all the required LFS patches. On Linux 2.4, patches also exist for ReiserFS to get support for big files (up to 2TB). With JFS and XFS, petabyte and larger files are possible on Linux.

For a detailed overview about LFS in Linux, have a look at Andreas Jaeger's *Large File Support in Linux* page at http://www.suse.de/~aj/linux_lfs.html.

If you do encounter a full-table error, there are several reasons why it might have occurred:

- The disk might be full.
- The [InnoDB](#) storage engine maintains [InnoDB](#) tables within a tablespace that can be created from several files. This enables a table to exceed the maximum individual file size. The tablespace can include raw disk partitions, which permits extremely large tables. The maximum tablespace size is 64TB.

If you are using [InnoDB](#) tables and run out of room in the [InnoDB](#) tablespace. In this case, the solution is to extend the [InnoDB](#) tablespace. See [Section 14.3.6, “Adding, Removing, or Resizing InnoDB Data and Log Files”](#).

- You are using [MyISAM](#) tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a [MyISAM](#) table and the space required for the table exceeds what is permitted by the internal pointer size. [MyISAM](#) permits data and index files to grow up to 256TB by default, but this limit can be changed up to the maximum permissible size of 65,536TB ($256^7 - 1$ bytes).

If you need a [MyISAM](#) table that is larger than the default limit and your operating system supports large files, the `CREATE TABLE` statement supports `AVG_ROW_LENGTH` and `MAX_ROWS` options. See [Section 13.1.17, “CREATE TABLE Syntax”](#). The server uses these options to determine how large a table to permit.

If the pointer size is too small for an existing table, you can change the options with `ALTER TABLE` to increase a table's maximum permissible size. See [Section 13.1.7, “ALTER TABLE Syntax”](#).

```
ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

You have to specify `AVG_ROW_LENGTH` only for tables with `BLOB` or `TEXT` columns; in this case, MySQL can't optimize the space required based only on the number of rows.

To change the default size limit for `MyISAM` tables, set the `myisam_data_pointer_size` [527], which sets the number of bytes used for internal row pointers. The value is used to set the pointer size for new tables if you do not specify the `MAX_ROWS` option. The value of `myisam_data_pointer_size` [527] can be from 2 to 7. A value of 4 permits tables up to 4GB; a value of 6 permits tables up to 256TB.

You can check the maximum data and index sizes by using this statement:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name' ;
```

You also can use `myisamchk -dv /path/to/table-index-file`. See [Section 13.7.5, "SHOW Syntax"](#), or [Section 4.6.3, "myisamchk — MyISAM Table-Maintenance Utility"](#).

Other ways to work around file-size limits for `MyISAM` tables are as follows:

- If your large table is read only, you can use `myisampack` to compress it. `myisampack` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. `myisampack` also can merge multiple tables into a single table. See [Section 4.6.5, "myisampack — Generate Compressed, Read-Only MyISAM Tables"](#).
- MySQL includes a `MERGE` library that enables you to handle a collection of `MyISAM` tables that have identical structure as a single `MERGE` table. See [Section 14.10, "The MERGE Storage Engine"](#).
- You are using the `NDB` storage engine, in which case you need to increase the values for the `DataMemory` and `IndexMemory` configuration parameters in your `config.ini` file. See [Section 17.3.3.1, "MySQL Cluster Data Node Configuration Parameters"](#).
- You are using the `MEMORY (HEAP)` storage engine; in this case you need to increase the value of the `max_heap_table_size` [521] system variable. See [Section 5.1.4, "Server System Variables"](#).

E.10.4. Table Column-Count and Row-Size Limits

There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table. The exact limit depends on several interacting factors.

- Every table (regardless of storage engine) has a maximum row size of 65,535 bytes. Storage engines may place additional constraints on this limit, reducing the effective maximum row size.

The maximum row size constrains the number (and possibly size) of columns because the total length of all columns cannot exceed this size. For example, `utf8` characters require up to three bytes per character, so for a `CHAR(255) CHARACTER SET utf8` column, the server must allocate $255 \times 3 = 765$ bytes per value. Consequently, a table cannot contain more than $65,535 / 765 = 85$ such columns.

Storage for variable-length columns includes length bytes, which are assessed against the row size. For example, a `VARCHAR(255) CHARACTER SET utf8` column takes two bytes to store the length of the value, so each value can take up to 767 bytes.

`BLOB` and `TEXT` columns count from one to four plus eight bytes each toward the row-size limit because their contents are stored separately from the rest of the row.

Declaring columns `NULL` can reduce the maximum number of columns permitted. For `MyISAM` tables, `NULL` columns require additional space in the row to record whether their values are `NULL`. Each `NULL` column takes one bit extra, rounded up to the nearest byte. The maximum row length in bytes can be calculated as follows:

```
row length = 1
             + (sum of column lengths)
             + (number of NULL columns + delete_flag + 7)/8
             + (number of variable-length columns)
```

`delete_flag` is 1 for tables with static row format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. `delete_flag` is 0 for dynamic tables because the flag is stored in the dynamic row header. For information about `MyISAM` table formats, see [Section 14.5.3, “MyISAM Table Storage Formats”](#).

These calculations do not apply for `InnoDB` tables. Storage size is the same for `NULL` and `NOT NULL` columns.

The following statement to create table `t1` succeeds because the columns require 32,765 + 2 bytes and 32,766 + 2 bytes, which falls within the maximum row size of 65,535 bytes:

```
mysql> CREATE TABLE t1
-> (c1 VARCHAR(32765) NOT NULL, c2 VARCHAR(32766) NOT NULL)
-> ENGINE = MyISAM CHARACTER SET latin1;
Query OK, 0 rows affected (0.02 sec)
```

The following statement to create table `t2` fails because the columns are `NULL` and `MyISAM` requires additional space that causes the row size to exceed 65,535 bytes:

```
mysql> CREATE TABLE t2
-> (c1 VARCHAR(32765) NULL, c2 VARCHAR(32766) NULL)
-> ENGINE = MyISAM CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

The following statement to create table `t3` fails because although the column length is within the maximum length of 65,535 bytes, two additional bytes are required to record the length, which causes the row size to exceed 65,535 bytes:

```
mysql> CREATE TABLE t3
-> (c1 VARCHAR(65535) NOT NULL)
-> ENGINE = MyISAM CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

Reducing the column length to 65,533 or less permits the statement to succeed.

- Each table has an `.frm` file that contains the table definition. The server uses the following expression to check some of the table information stored in the file against an upper limit of 64KB:

```
if (info_length+(ulong) create_fields.elements*FCOMP+288+
    n_length+int_length+com_length > 65535L || int_count > 255)
```

The portion of the information stored in the `.frm` file that is checked against the expression cannot grow beyond the 64KB limit, so if the table definition reaches this size, no more columns can be added.

The relevant factors in the expression are:

- `info_length` is space needed for “screens.” This is related to MySQL's Unireg heritage.
- `create_fields.elements` is the number of columns.
- `FCOMP` is 17.

- `n_length` is the total length of all column names, including one byte per name as a separator.
- `int_length` is related to the list of values for `ENUM` and `SET` columns.
- `com_length` is the total length of column and table comments.

Thus, using long column names can reduce the maximum number of columns, as can the inclusion of `ENUM` or `SET` columns, or use of column, index, or table comments.

- Individual storage engines might impose additional restrictions that limit table column count. Examples:
 - `InnoDB` permits up to 1000 columns.
 - `InnoDB` restricts row size to something less than half a database page (approximately 8000 bytes), not including `VARBINARY`, `VARCHAR`, `BLOB`, or `TEXT` columns.
 - Different `InnoDB` storage formats (`COMPRESSED`, `REDUNDANT`) use different amounts of page header and trailer data, which affects the amount of storage available for rows.

E.10.5. Windows Platform Limitations

The following limitations apply to use of MySQL on the Windows platform:

- **Process memory**

On Windows 32-bit platforms, it is not possible by default to use more than 2GB of RAM within a single process, including MySQL. This is because the physical address limit on Windows 32-bit is 4GB and the default setting within Windows is to split the virtual address space between kernel (2GB) and user/ applications (2GB).

Some versions of Windows have a boot time setting to enable larger applications by reducing the kernel application. Alternatively, to use more than 2GB, use a 64-bit version of Windows.

- **File system aliases**

When using `MyISAM` tables, you cannot use aliases within Windows link to the data files on another volume and then link back to the main MySQL `datadir` [439] location.

This facility is often used to move the data and index files to a RAID or other fast solution, while retaining the main `.frm` files in the default data directory configured with the `datadir` [439] option.

- **Limited number of ports**

Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Note that ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

For more information about this problem, see <http://support.microsoft.com/default.aspx?scid=kb;en-us;196271>.

- **DATA DIRECTORY and INDEX DIRECTORY**

The `DATA DIRECTORY` and `INDEX DIRECTORY` options for `CREATE TABLE` are ignored on Windows, because MySQL does not support Windows symbolic links. These options also are ignored on systems that have a nonfunctional `realpath()` call.

- **DROP DATABASE**

You cannot drop a database that is in use by another session.

- **Case-insensitive names**

File names are not case sensitive on Windows, so MySQL database and table names are also not case sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See [Section 9.2.2, “Identifier Case Sensitivity”](#).

- **Directory and file names**

On Windows, MySQL Server supports only directory and file names that are compatible with the current ANSI code pages. For example, the following Japanese directory name will not work in the Western locale (code page 1252):

```
datadir="C:/私たちのプロジェクトのデータ"
```

The same limitation applies to directory and file names referred to in SQL statements, such as the data file path name in `LOAD DATA INFILE`.

- **The “\” path name separator character**

Path name components in Windows are separated by the “\” character, which is also the escape character in MySQL. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, use Unix-style file names with “/” characters:

```
mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;  
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

Alternatively, you must double the “\” character:

```
mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;  
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

- **Problems with pipes**

Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z / CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

This is mainly a problem when you try to apply a binary log as follows:

```
C:\> mysqlbinlog binary_log_file | mysql --user=root
```

If you have a problem applying the log and suspect that it is because of a `^Z / CHAR(24)` character, you can use the following workaround:

```
C:\> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql  
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

The latter command also can be used to reliably read in any SQL file that may contain binary data.

Appendix F. MySQL Glossary

These terms are commonly used in information about the MySQL database server. This glossary originated as a reference for terminology about the InnoDB storage engine, and the majority of definitions are InnoDB-related.

A

ACID

An acronym standing for atomicity, consistency, isolation, and durability. These properties are all desirable in a database system, and are all closely tied to the notion of a **transaction**. The transactional features of InnoDB adhere to the ACID principles.

Transactions are **atomic** units of work that can be **committed** or **rolled back**. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

The database remains in a consistent state at all times -- after each commit or rollback, and while transactions are in progress. If related data is being updated across multiple tables, queries see either all old values or all new values, not a mix of old and new values.

Transactions are protected (isolated) from each other while they are in progress; they cannot interfere with each other or see each other's uncommitted data. This isolation is achieved through the **locking** mechanism. Experienced users can adjust the **isolation level**, trading off less protection in favor of increased performance and **concurrency**, when they can be sure that the transactions really do not interfere with each other.

The results of transactions are durable: once a commit operation succeeds, the changes made by that transaction are safe from power failures, system crashes, race conditions, or other potential dangers that many non-database applications are vulnerable to. Durability typically involves writing to disk storage, with a certain amount of redundancy to protect against power failures or software crashes during write operations. (In InnoDB, the **doublewrite buffer** assists with durability.)

See also [atomic](#).

See also [commit](#).

See also [concurrency](#).

See also [doublewrite buffer](#).

See also [isolation level](#).

See also [locking](#).

See also [rollback](#).

See also [transaction](#).

adaptive flushing

An algorithm for **InnoDB** tables that smooths out the I/O overhead introduced by **checkpoints**. Instead of **flushing** all modified **pages** from the **buffer pool** to the **data files** at once, MySQL periodically flushes small sets of modified pages. The adaptive flushing algorithm extends this process by estimating the optimal rate to perform these periodic flushes, based on the rate of flushing and how fast **redo** information is generated. First introduced in MySQL 5.1, in the InnoDB Plugin.

See also [buffer pool](#).

See also [checkpoint](#).

See also [data files](#).

See also [flush](#).

See also [InnoDB](#).

See also [page](#).

See also [redo log](#).

adaptive hash index

An optimization for InnoDB tables that can speed up lookups using = and **IN** operators, by constructing a **hash index** in memory. MySQL monitors index searches for InnoDB tables, and if queries could benefit from a hash index, it builds one automatically for index **pages** that are frequently accessed. In a sense, the adaptive hash index configures MySQL at runtime to take advantage of ample main memory, coming closer to the architecture of main-memory databases. This feature is controlled by the [innodb_adaptive_hash_index \[1547\]](#) configuration option. Because this feature benefits some workloads and not others, and the memory used for the hash index is reserved in the **buffer pool**, typically you should benchmark with this feature both enabled and disabled.

The hash index is always built based on an existing InnoDB **secondary index**, which is organized as a **B-tree** structure. MySQL can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches against the index. A hash index can be partial; the whole B-tree index does not need to be cached in the buffer pool.

In MySQL 5.6 and higher, another way to take advantage of fast single-value lookups with InnoDB tables is to use the **memcached** interface to InnoDB. See [InnoDB Integration with memcached](#) for details.

See also [B-tree](#).

See also [buffer pool](#).

See also [hash index](#).

See also [memcached](#).

See also [page](#).

| | |
|---|--|
| | See also secondary index . |
| AHI | Acronym for adaptive hash index . |
| | See also adaptive hash index . |
| AIO | Acronym for asynchronous I/O . You might see this acronym in InnoDB messages or keywords. |
| | See also asynchronous I/O . |
| Antelope | <p>The code name for the original InnoDB file format. It supports the redundant and compact row formats, but not the newer dynamic and compressed row formats available in the Barracuda file format.</p> <p>If your application could benefit from InnoDB table compression, or uses BLOBs or large text columns that could benefit from the dynamic row format, you might switch some tables to Barracuda format. You select the file format to use by setting the innodb_file_format [1554] option before creating the table.</p> <p>See also Barracuda.</p> <p>See also compact row format.</p> <p>See also compressed row format.</p> <p>See also dynamic row format.</p> <p>See also file format.</p> <p>See also innodb_file_format.</p> <p>See also redundant row format.</p> |
| application programming interface (API) | A set of functions or procedures. An API provides a stable set of names and types for functions, procedures, parameters, and return values. |
| apply | <p>When a backup produced by the MySQL Enterprise Backup product does not include the most recent changes that occurred while the backup was underway, the process of updating the backup files to include those changes is known as the apply step. It is specified by the apply-log option of the mysqlbackup command.</p> <p>Before the changes are applied, we refer to the files as a raw backup. After the changes are applied, we refer to the files as a prepared backup. The changes are recorded in the ibbackup_logfile file; once the apply step is finished, this file is no longer necessary.</p> <p>See also hot backup.</p> <p>See also ibbackup_logfile.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also prepared backup.</p> |

| | |
|------------------------------------|--|
| | See also raw backup . |
| .ARM file | <p>Metadata for ARCHIVE tables. Contrast with .ARZ file. Files with this extension are always included in backups produced by the mysqlbackup command of the MySQL Enterprise Backup product.</p> <p>See also .ARZ file.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also mysqlbackup command.</p> |
| .ARZ file | <p>Data for ARCHIVE tables. Contrast with .ARM file. Files with this extension are always included in backups produced by the mysqlbackup command of the MySQL Enterprise Backup product.</p> <p>See also .ARM file.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also mysqlbackup command.</p> |
| asynchronous I/O | <p>A type of I/O operation that allows other processing to proceed before the I/O is completed. Also known as non-blocking I/O and abbreviated as AIO. InnoDB uses this type of I/O for certain operations that can run in parallel without affecting the reliability of the database, such as reading pages into the buffer pool that have not actually been requested, but might be needed soon.</p> <p>Historically, InnoDB has used asynchronous I/O on Windows systems only. Starting with the InnoDB Plugin 1.1, InnoDB uses asynchronous I/O on Linux systems. This change introduces a dependency on libaio. On other Unix-like systems, InnoDB uses synchronous I/O only.</p> <p>See also buffer pool.</p> <p>See also non-blocking I/O.</p> |
| atomic | <p>In the SQL context, transactions are units of work that either succeed entirely (when committed) or have no effect at all (when rolled back). The indivisible (“atomic”) property of transactions is the “A” in the acronym ACID.</p> <p>See also ACID.</p> <p>See also commit.</p> <p>See also rollback.</p> <p>See also transaction.</p> |
| atomic instruction | <p>Special instructions provided by the CPU, to ensure that critical low-level operations cannot be interrupted.</p> |
| auto-increment | <p>A property of a table column (specified by the AUTO_INCREMENT keyword) that automatically adds an ascending sequence of values</p> |

in the column. InnoDB supports auto-increment only for **primary key** columns.

It saves work for the developer, not to have to produce new unique values when inserting new rows. It provides useful information for the query optimizer, because the column is known to be not null and with unique values. The values from such a column can be used as lookup keys in various contexts, and because they are auto-generated there is no reason to ever change them; for this reason, primary key columns are often specified as auto-incrementing.

Auto-increment columns can be problematic with statement-based replication, because replaying the statements on a slave might not produce the same set of column values as on the master, due to timing issues. When you have an auto-incrementing primary key, you can use statement-based replication only with the setting `innodb_autoinc_lock_mode=1` [1548]. If you have `innodb_autoinc_lock_mode=2`, which allows higher concurrency for insert operations, use **row-based replication** rather than **statement-based replication**. The setting `innodb_autoinc_lock_mode=0` is the previous (traditional) default setting and should not be used except for compatibility purposes.

See also [auto-increment locking](#).

See also [innodb_autoinc_lock_mode](#).

See also [primary key](#).

See also [row-based replication](#).

See also [statement-based replication](#).

auto-increment locking

The convenience of an **auto-increment** primary key involves some tradeoff with concurrency. In the simplest case, if one transaction is inserting values into the table, any other transactions must wait to do their own inserts into that table, so that rows inserted by the first transaction receive consecutive primary key values. InnoDB includes optimizations, and the `innodb_autoinc_lock_mode` [1548] option, so that you can choose how to trade off between predictable sequences of auto-increment values and maximum **concurrency** for insert operations.

See also [auto-increment](#).

See also [concurrency](#).

See also [innodb_autoinc_lock_mode](#).

autocommit

A setting that causes a **commit** operation after each **SQL** statement. This mode is not recommended for working with InnoDB tables with **transactions** that span several statements. It can help performance for **read-only transactions** on InnoDB tables, where it minimizes overhead from **locking** and generation of **undo** data, especially in MySQL 5.6.4 and up. It is also appropriate for working with MyISAM tables, where transactions are not applicable.

See also [commit](#).

See also [locking](#).

See also [read-only transaction](#).

See also [SQL](#).

See also [transaction](#).

See also [undo](#).

B

B-tree

A tree data structure that is popular for use in database indexes. The structure is kept sorted at all times, enabling fast lookup for exact matches (equals operator) and ranges (for example, greater than, less than, and [BETWEEN](#) operators). This type of index is available for most storage engines, such as InnoDB and MyISAM.

Because B-tree nodes can have many children, a B-tree is not the same as a binary tree, which is limited to 2 children per node.

Contrast with **hash index**, which is only available in the MEMORY storage engine. The MEMORY storage engine can also use B-tree indexes, and you should choose B-tree indexes for MEMORY tables if some queries use range operators.

See also [hash index](#).

backticks

Identifiers within MySQL SQL statements must be quoted using the backtick character (`) if they contain special characters or reserved words. For example, to refer to a table named `FOO#BAR` or a column named `SELECT`, you would specify the identifiers as ``FOO#BAR`` and ``SELECT``. Since the backticks provide an extra level of safety, they are used extensively in program-generated SQL statements, where the identifier names might not be known in advance.

Many other database systems use double quotation marks (") around such special names. For portability, you can enable [ANSI_QUOTES](#) mode in MySQL and use double quotation marks instead of backticks to qualify identifier names.

See also [SQL](#).

backup

The process of copying some or all table data and metadata from a MySQL instance, for safekeeping. Can also refer to the set of copied files. This is a crucial task for DBAs. The reverse of this process is the **restore** operation.

With MySQL, **physical backups** are performed by the **MySQL Enterprise Backup** product, and **logical backups** are performed by the `mysqldump` command. These techniques have different characteristics in terms of size and representation of the backup data, and speed (especially speed of the restore operation).

Backups are further classified as **hot**, **warm**, or **cold** depending on how much they interfere with normal database operation. (Hot backups have the least interference, cold backups the most.)

See also [cold backup](#).

See also [hot backup](#).

See also [logical backup](#).

See also [MySQL Enterprise Backup](#).

See also [mysqldump](#).

See also [physical backup](#).

See also [warm backup](#).

Barracuda

The code name for an InnoDB **file format** that supports compression for table data. This file format was first introduced in the InnoDB Plugin. It supports the **compressed** row format that enables InnoDB table compression, and the **dynamic** row format that improves the storage layout for BLOB and large text columns. You can select it through the [innodb_file_format \[1554\]](#) option.

Because the InnoDB **system tablespace** is stored in the original **Antelope** file format, to use the Barracuda file format you must also enable the **file-per-table** setting, which puts newly created tables in their own tablespaces separate from the system tablespace.

The **MySQL Enterprise Backup** product version 3.5 and above supports backing up tablespaces that use the Barracuda file format.

See also [Antelope](#).

See also [compact row format](#).

See also [compressed row format](#).

See also [dynamic row format](#).

See also [file format](#).

See also [file-per-table](#).

See also [innodb_file_format](#).

See also [MySQL Enterprise Backup](#).

See also [row format](#).

See also [system tablespace](#).

beta

An early stage in the life of a software product, when it is available only for evaluation, typically without a definite release number or a number less than 1. InnoDB does not use the beta designation, preferring an **early adopter** phase that can extend over several point releases, leading to a **GA** release.

| | |
|-----------------------|---|
| | <p>See also early adopter.</p> <p>See also GA.</p> |
| binary log | <p>A file containing a record of all statements that attempt to change table data. These statements can be replayed to bring slave servers up to date in a replication scenario, or to bring a database up to date after restoring table data from a backup. The binary logging feature can be turned on and off, although Oracle recommends always enabling it if you use replication or perform backups.</p> <p>You can examine the contents of the binary log, or replay those statements during replication or recovery, by using the <code>mysqlbinlog</code> command. For full information about the binary log, see Section 5.2.4, “The Binary Log”. For MySQL configuration options related to the binary log, see Section 16.1.3.4, “Binary Log Options and Variables”.</p> <p>For the MySQL Enterprise Backup product, the file name of the binary log and the current position within the file are important details. To record this information for the master server when taking a backup in a replication context, you can specify the <code>--slave-info</code> option.</p> <p>Prior to MySQL 5.0, a similar capability was available, known as the update log. In MySQL 5.0 and higher, the binary log replaces the update log.</p> <p>See also binlog.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also replication.</p> |
| binlog | <p>An informal name for the binary log file. For example, you might see this abbreviation used in e-mail messages or forum discussions.</p> <p>See also binary log.</p> |
| blind query expansion | <p>A special mode of full-text search enabled by the <code>WITH QUERY EXPANSION</code> clause. It performs the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. This technique is mainly applicable for short search phrases, perhaps only a single word. It can uncover relevant matches where the precise search term does not occur in the document.</p> <p>See also full-text search.</p> |
| bottleneck | <p>A portion of a system that is constrained in size or capacity, that has the effect of limiting overall throughput. For example, a memory area might be smaller than necessary; access to a single required resource might prevent multiple CPU cores from running simultaneously; or waiting for disk I/O to complete might prevent the CPU from running at full capacity. Removing bottlenecks tends to improve concurrency. For example, the ability to have multiple InnoDB buffer pool instances reduces contention when multiple sessions read from and write to the buffer pool simultaneously.</p> <p>See also buffer pool.</p> |

| | |
|-----------------|--|
| | See also concurrency . |
| bounce | A shutdown operation immediately followed by a restart. Ideally with a relatively short warmup period so that performance and throughput quickly return to a high level. |
| | See also shutdown . |
| buddy allocator | A mechanism for managing different-sized pages in the InnoDB buffer pool . |
| | See also buffer pool . |
| | See also page . |
| | See also page size . |
| buffer | A memory or disk area used for temporary storage. Data is buffered in memory so that it can be written to disk efficiently, with a few large I/O operations rather than many small ones. Data is buffered on disk for greater reliability, so that it can be recovered even when a crash or other failure occurs at the worst possible time. The main types of buffers used by InnoDB are the buffer pool , the doublewrite buffer , and the insert buffer . |
| | See also buffer pool . |
| | See also crash . |
| | See also doublewrite buffer . |
| | See also insert buffer . |
| buffer pool | The memory area that holds cached InnoDB data for both tables and indexes. For efficiency of high-volume read operations, the buffer pool is divided into pages that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache, using a variation of the LRU algorithm. On systems with large memory, you can improve concurrency by dividing the buffer pool into multiple buffer pool instances . |
| | Several InnoDB status variables, information_schema tables, and performance_schema tables help to monitor the internal workings of the buffer pool. Starting in MySQL 5.6, you can also dump and restore the contents of the buffer pool, either automatically during shutdown and restart, or manually at any time, through a set of InnoDB configuration variables such as innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup . |
| | See also buffer pool instance . |
| | See also LRU . |
| | See also page . |
| | See also warm up . |

buffer pool instance Any of the multiple regions into which the **buffer pool** can be divided, controlled by the [innodb_buffer_pool_instances \[1549\]](#) configuration option. The total memory size specified by the [innodb_buffer_pool_size \[1549\]](#) is divided among all the instances. Typically, multiple buffer pool instances are appropriate for systems devoting multiple gigabytes to the InnoDB buffer pool, with each instance 1 gigabyte or larger. On systems loading or looking up large amounts of data in the buffer pool from many concurrent sessions, having multiple instances reduces the contention for exclusive access to the data structures that manage the buffer pool.

See also [buffer pool](#).

built-in The built-in InnoDB storage engine within MySQL is the original form of distribution for the storage engine. Contrast with the **InnoDB Plugin**. Starting with MySQL 5.5, the InnoDB Plugin is merged back into the MySQL code base as the built-in InnoDB storage engine (known as InnoDB 1.1).

This distinction is important mainly in MySQL 5.1, where a feature or bug fix might apply to the InnoDB Plugin but not the built-in InnoDB, or vice versa.

See also [InnoDB](#).

See also [plugin](#).

business rules The relationships and sequences of actions that form the basis of business software, used to run a commercial company. Sometimes these rules are dictated by law, other times by company policy. Careful planning ensures that the relationships encoded and enforced by the database, and the actions performed through application logic, accurately reflect the real policies of the company and can handle real-life situations.

For example, an employee leaving a company might trigger a sequence of actions from the human resources department. The human resources database might also need the flexibility to represent data about a person who has been hired, but not yet started work. Closing an account at an online service might result in data being removed from a database, or the data might be moved or flagged so that it could be recovered if the account is re-opened. A company might establish policies regarding salary maximums, minimums, and adjustments, in addition to basic sanity checks such as the salary not being a negative number. A retail database might not allow a purchase with the same serial number to be returned more than once, or might not allow credit card purchases above a certain value, while a database used to detect fraud might allow these kinds of things.

See also [relational](#).

C

cache The general term for any memory area that stores copies of data for frequent or high-speed retrieval. In InnoDB, the primary kind of cache structure is the **buffer pool**.

See also [buffer](#).

See also [buffer pool](#).

cardinality

The number of different values in a table **column**. When queries refer to columns that have an associated **index**, the cardinality of each column influences which access method is most efficient. For example, for a column with a **unique constraint**, the number of different values is equal to the number of rows in the table. If a table has a million rows but only 10 different values for a particular column, each value occurs (on average) 100,000 times. A query such as `SELECT c1 FROM t1 WHERE c1 = 50;` thus might return 1 row or a huge number of rows, and the database server might process the query differently depending on the cardinality of `c1`.

If the values in a column have a very uneven distribution, the cardinality might not be a good way to determine the best query plan. For example, `SELECT c1 FROM t1 WHERE c1 = x;` might return 1 row when `x=50` and a million rows when `x=30`. In such a case, you might need to use **index hints** to pass along advice about which lookup method is more efficient for a particular query.

Cardinality can also apply to the number of distinct values present in multiple columns, as in a **composite index**.

For InnoDB, the process of estimating cardinality for indexes is influenced by the `innodb_stats_sample_pages [1571]` and the `innodb_stats_on_metadata [1571]` configuration options. The estimated values are more stable when the **persistent statistics** feature is enabled (in MySQL 5.6 and higher).

See also [column](#).

See also [composite index](#).

See also [index](#).

See also [index hint](#).

See also [persistent statistics](#).

See also [random dive](#).

See also [selectivity](#).

See also [unique constraint](#).

.cfg file

A metadata file used with the InnoDB **transportable tablespace** feature. It is produced by the command `FLUSH TABLES ... FOR EXPORT`, puts one or more tables in a consistent state that can be copied to another server. The `.cfg` file is copied along with the corresponding **.ibd file**, and used to adjust the internal values of the `.ibd` file, such as the **space ID**, during the `ALTER TABLE ... IMPORT TABLESPACE` step.

See also [.ibd file](#).

See also [space ID](#).

See also [transportable tablespace](#).

change buffer

A special data structure that records changes to **pages** in **secondary indexes**. These values could result from SQL `INSERT`, `UPDATE`, or `DELETE`

statements (**DML**). The set of features involving the change buffer is known collectively as **change buffering**, consisting of **insert buffering**, **delete buffering**, and **purge buffering**.

Changes are only recorded in the change buffer when the relevant page from the secondary index is not in the **buffer pool**. When the relevant index page is brought into the buffer pool while associated changes are still in the change buffer, the changes for that page are applied in the buffer pool (**merged**) using the data from the change buffer. Periodically, the **purge** operation that runs during times when the system is mostly idle, or during a slow shutdown, writes the new index pages to disk. The purge operation can write the disk blocks for a series of index values more efficiently than if each value were written to disk immediately.

Physically, the change buffer is part of the **system tablespace**, so that the index changes remain buffered across database restarts. The changes are only applied (**merged**) when the pages are brought into the buffer pool due to some other read operation.

The kinds and amount of data stored in the change buffer are governed by the [innodb_change_buffering \[1550\]](#) and [innodb_change_buffer_max_size](#) configuration options. To see information about the current data in the change buffer, issue the [SHOW ENGINE INNODB STATUS](#) command.

Formerly known as the **insert buffer**.

See also [buffer pool](#).

See also [change buffering](#).

See also [delete buffering](#).

See also [DML](#).

See also [insert buffer](#).

See also [insert buffering](#).

See also [merge](#).

See also [page](#).

See also [purge](#).

See also [purge buffering](#).

See also [secondary index](#).

See also [system tablespace](#).

change buffering

The general term for the features involving the **change buffer**, consisting of **insert buffering**, **delete buffering**, and **purge buffering**. Index changes resulting from SQL statements, which could normally involve random I/O operations, are held back and performed periodically by a background **thread**. This sequence of operations can write the disk blocks for a series of index values more efficiently than if each value were written to disk

immediately. Controlled by the `innodb_change_buffering` [1550] and `innodb_change_buffer_max_size` configuration options.

See also [change buffer](#).

See also [delete buffering](#).

See also [insert buffering](#).

See also [purge buffering](#).

checkpoint

As changes are made to data pages that are cached in the **buffer pool**, those changes are written to the **data files** sometime later, a process known as **flushing**. The checkpoint is a record of the latest changes (represented by an **LSN** value) that have been successfully written to the data files.

See also [buffer pool](#).

See also [data files](#).

See also [flush](#).

See also [fuzzy checkpointing](#).

See also [LSN](#).

checksum

In **InnoDB**, a validation mechanism to detect corruption when a **page** in a **tablespace** is read from disk into the InnoDB **buffer pool**. This feature is turned on and off by the `innodb_checksums` [1551] configuration option. In MySQL 5.6, you can enable a faster checksum algorithm by also specifying the configuration option `innodb_checksum_algorithm=crc32`.

The `innochecksum` command helps to diagnose corruption problems by testing the checksum values for a specified **tablespace** file while the MySQL server is shut down.

MySQL also uses checksums for replication purposes. For details, see the configuration options `binlog_checksum`, `master_verify_checksum`, and `slave_sql_verify_checksum`.

See also [buffer pool](#).

See also [page](#).

See also [tablespace](#).

child table

In a **foreign key** relationship, a child table is one whose rows refer (or point) to rows in another table with an identical value for a specific column. This is the table that contains the `FOREIGN KEY ... REFERENCES` clause and optionally `ON UPDATE` and `ON DELETE` clauses. The corresponding row in the **parent table** must exist before the row can be created in the child table. The values in the child table can prevent delete or update operations on the parent table, or can cause automatic deletion or updates in the child table, based on the `ON CASCADE` option used when creating the foreign key.

| | |
|-----------------|---|
| | See also foreign key . |
| | See also parent table . |
| clean page | <p>A page in the InnoDB buffer pool where all changes made in memory have also been written (flushed) to the data files. The opposite of a dirty page.</p> <p>See also buffer pool.</p> <p>See also data files.</p> <p>See also dirty page.</p> <p>See also flush.</p> <p>See also page.</p> |
| clean shutdown | <p>A shutdown that completes without errors and applies all changes to InnoDB tables before finishing, as opposed to a crash or a fast shutdown. Synonym for slow shutdown.</p> <p>See also crash.</p> <p>See also fast shutdown.</p> <p>See also shutdown.</p> <p>See also slow shutdown.</p> |
| client | <p>A type of program that sends requests to a server, and interprets or processes the results. The client software might run only some of the time (such as a mail or chat program), and might run interactively (such as the mysql command processor).</p> <p>See also mysql.</p> <p>See also server.</p> |
| clustered index | <p>The InnoDB term for a primary key index. InnoDB table storage is organized based on the values of the primary key columns, to speed up queries and sorts involving the primary key columns. For best performance, choose the primary key columns carefully based on the most performance-critical queries. Because modifying the columns of the clustered index is an expensive operation, choose primary columns that are rarely or never updated.</p> <p>In the Oracle Database product, this type of table is known as an index-organized table.</p> <p>See also index.</p> <p>See also primary key.</p> <p>See also secondary index.</p> |

| | |
|---------------|---|
| cold backup | <p>A backup taken while the database is shut down. For busy applications and web sites, this might not be practical, and you might prefer a warm backup or a hot backup.</p> <p>See also backup.</p> <p>See also hot backup.</p> <p>See also warm backup.</p> |
| column | <p>A data item within a row, whose storage and semantics are defined by a data type. Each table and index is largely defined by the set of columns it contains.</p> <p>Each column has a cardinality value. A column can be the primary key for its table, or part of the primary key. A column can be subject to a unique constraint, a NOT NULL constraint, or both. Values in different columns, even across different tables, can be linked by a foreign key relationship.</p> <p>In discussions of MySQL internal operations, sometimes field is used as a synonym, while row is the preferred term when discussing SQL.</p> <p>See also cardinality.</p> <p>See also foreign key.</p> <p>See also index.</p> <p>See also primary key.</p> <p>See also row.</p> <p>See also SQL.</p> <p>See also table.</p> <p>See also unique constraint.</p> |
| column index | <p>An index on a single column.</p> <p>See also composite index.</p> <p>See also index.</p> |
| column prefix | <p>When an index is created with a length specification, such as <code>CREATE INDEX idx ON t1 (c1(N))</code>, only the first N characters of the column value are stored in the index. Keeping the index prefix small makes the index compact, and the memory and disk I/O savings help performance. (Although making the index prefix too small can hinder query optimization by making rows with different values appear to the query optimizer to be duplicates.)</p> <p>For columns containing binary values or long text strings, where sorting is not a major consideration and storing the entire value in the index would waste space, the index automatically uses the first N (typically 768) characters of the value to do lookups and sorts.</p> |

| | |
|--------------------|---|
| | See also index . |
| commit | <p>A SQL statement that ends a transaction, making permanent any changes made by the transaction. It is the opposite of rollback, which undoes any changes made in the transaction.</p> <p>InnoDB uses an optimistic mechanism for commits, so that changes can be written to the data files before the commit actually occurs. This technique makes the commit itself faster, with the tradeoff that more work is required in case of a rollback.</p> <p>By default, MySQL uses the autocommit setting, which automatically issues a commit following each SQL statement.</p> <p>See also autocommit.</p> <p>See also optimistic.</p> <p>See also rollback.</p> <p>See also SQL.</p> <p>See also transaction.</p> |
| compact row format | <p>The default InnoDB row format since MySQL 5.0.3. Available for tables that use the Antelope file format. It has a more compact representation for nulls and variable-length fields than the prior default (redundant row format).</p> <p>Because of the B-tree indexes that make row lookups so fast in InnoDB, there is little if any performance benefit to keeping all rows the same size.</p> <p>See also Antelope.</p> <p>See also file format.</p> <p>See also redundant row format.</p> <p>See also row format.</p> |
| composite index | <p>An index that includes multiple columns.</p> <p>See also index.</p> <p>See also index prefix.</p> |
| compressed backup | <p>The compression feature of the MySQL Enterprise Backup product makes a compressed copy of each tablespace, changing the extension from <code>.ibd</code> to <code>.ibz</code>. Compressing the backup data allows you to keep more backups on hand, and reduces the time to transfer backups to a different server. The data is uncompressed during the restore operation. When a compressed backup operation processes a table that is already compressed, it skips the compression step for that table, because compressing again would result in little or no space savings.</p> |

A set of files produced by the **MySQL Enterprise Backup** product, where each **tablespace** is compressed. The compressed files are renamed with a [.ibz](#) file extension.

Applying **compression** right at the start of the backup process helps to avoid storage overhead during the compression process, and to avoid network overhead when transferring the backup files to another server. The process of **applying the binary log** takes longer, and requires uncompressing the backup files.

See also [apply](#).

See also [binary log](#).

See also [compression](#).

See also [hot backup](#).

See also [MySQL Enterprise Backup](#).

See also [tablespace](#).

compressed row format

A **row format** that enables data and index **compression** for InnoDB tables. It was introduced in the InnoDB Plugin, available as part of the **Barracuda** file format. Large fields are stored away from the page that holds the rest of the row data, as in **dynamic row format**. Both index pages and the large fields are compressed, yielding memory and disk savings. Depending on the structure of the data, the decrease in memory and disk usage might or might not outweigh the performance overhead of uncompressing the data as it is used. See [Section 14.4.3, “InnoDB Data Compression”](#) for usage details.

See also [Barracuda](#).

See also [compression](#).

See also [dynamic row format](#).

See also [row format](#).

compression

A feature with wide-ranging benefits from using less disk space, performing less I/O, and using less memory for caching. InnoDB table and index data can be kept in a compressed format during database operation.

The data is uncompressed when needed for queries, and re-compressed when changes are made by **DML** operations. After you enable compression for a table, this processing is transparent to users and application developers. DBAs can consult **information_schema** tables to monitor how efficiently the compression parameters work for the MySQL instance and for particular compressed tables.

When InnoDB table data is compressed, the compression applies to the **table** itself, any associated **index** data, and the pages loaded into the **buffer pool**. Compression does not apply to pages in the **undo buffer**.

The table compression feature requires using MySQL 5.5 or higher, or the InnoDB Plugin in MySQL 5.1 or earlier, and creating the table

using the **Barracuda** file format and **compressed row format**, with the **innodb_file_per_table** setting turned on. The compression for each table is influenced by the `KEY_BLOCK_SIZE` clause of the `CREATE TABLE` and `ALTER TABLE` statements. In MySQL 5.6 and higher, compression is also affected by the server-wide configuration options `innodb_compression_failure_threshold_pct`, `innodb_compression_level`, and `innodb_compression_pad_pct_max`. See [Section 14.4.3, “InnoDB Data Compression”](#) for usage details.

Another type of compression is the **compressed backup** feature of the **MySQL Enterprise Backup** product.

See also [Barracuda](#).

See also [buffer pool](#).

See also [compressed row format](#).

See also [DML](#).

See also [hot backup](#).

See also [index](#).

See also [INFORMATION_SCHEMA](#).

See also [innodb_file_per_table](#).

See also [plugin](#).

See also [table](#).

See also [undo buffer](#).

compression failure

Not actually an error, rather an expensive operation that can occur when using **compression** in combination with **DML** operations. It occurs when: updates to a compressed **page** overflow the area on the page reserved for recording modifications; the page is compressed again, with all changes applied to the table data; the re-compressed data does not fit on the original page, requiring MySQL to split the data into two new pages and compress each one separately. To check the frequency of this condition, query the table `INFORMATION_SCHEMA.INNODB_CMP` and check how much the value of the `COMPRESS_OPS` column exceeds the value of the `COMPRESS_OPS_OK` column. Ideally, compression failures do not occur often; when they do, you can adjust the configuration options `innodb_compression_level`, `innodb_compression_failure_threshold_pct`, and `innodb_compression_pad_pct_max`.

See also [compression](#).

See also [DML](#).

See also [page](#).

concatenated index

See [composite index](#).

| | |
|--------------------|--|
| concurrency | <p>The ability of multiple operations (in database terminology, transactions) to run simultaneously, without interfering with each other. Concurrency is also involved with performance, because ideally the protection for multiple simultaneous transactions works with a minimum of performance overhead, using efficient mechanisms for locking.</p> <p>See also ACID.</p> <p>See also locking.</p> <p>See also transaction.</p> |
| configuration file | <p>The file that holds the option values used by MySQL at startup. Traditionally, on Linux and UNIX this file is named <code>my.cnf</code>, and on Windows it is named <code>my.ini</code>. You can set a number of options related to InnoDB under the <code>[mysqld]</code> section of the file.</p> <p>Typically, this file is searched for in the locations <code>/etc/my.cnf</code> <code>/etc/mysql/my.cnf</code> <code>/usr/local/mysql/etc/my.cnf</code> and <code>~/my.cnf</code>. See Section 4.2.3.3, "Using Option Files" for details about the search path for this file.</p> <p>When you use the MySQL Enterprise Backup product, you typically use two configuration files: one that specifies where the data comes from and how it is structured (which could be the original configuration file for your real server), and a stripped-down one containing only a small set of options that specify where the backup data goes and how it is structured. The configuration files used with the MySQL Enterprise Backup product must contain certain options that are typically left out of regular configuration files, so you might need to add some options to your existing configuration file for use with MySQL Enterprise Backup.</p> <p>See also my.cnf.</p> <p>See also option file.</p> |
| consistent read | <p>A read operation that uses snapshot information to present query results based on a point in time, regardless of changes performed by other transactions running at the same time. If queried data has been changed by another transaction, the original data is reconstructed based on the contents of the undo log. This technique avoids some of the locking issues that can reduce concurrency by forcing transactions to wait for other transactions to finish.</p> <p>With the repeatable read isolation level, the snapshot is based on the time when the first read operation is performed. With the read committed isolation level, the snapshot is reset to the time of each consistent read operation.</p> <p>Consistent read is the default mode in which InnoDB processes <code>SELECT</code> statements in READ COMMITTED and REPEATABLE READ isolation levels. Because a consistent read does not set any locks on the tables it accesses, other sessions are free to modify those tables while a consistent read is being performed on the table.</p> |

For technical details about the applicable isolation levels, see [Section 14.3.9.2, “Consistent Nonlocking Reads”](#).

See also [ACID](#).

See also [concurrency](#).

See also [isolation level](#).

See also [locking](#).

See also [MVCC](#).

See also [READ COMMITTED](#).

See also [READ UNCOMMITTED](#).

See also [REPEATABLE READ](#).

See also [SERIALIZABLE](#).

See also [transaction](#).

See also [undo log](#).

constraint

An automatic test that can block database changes to prevent data from becoming inconsistent. (In computer science terms, a kind of assertion related to an invariant condition.) Constraints are a crucial component of the **ACID** philosophy, to maintain data consistency. Constraints supported by MySQL include **FOREIGN KEY constraints** and **unique constraints**.

See also [ACID](#).

See also [foreign key](#).

See also [relational](#).

See also [unique constraint](#).

counter

A value that is incremented by a particular kind of [InnoDB](#) operation. Useful for measuring how busy a server is, troubleshooting the sources of performance issues, and testing whether changes (for example, to configuration settings or indexes used by queries) have the desired low-level effects. Different kinds of counters are available through **performance_schema** tables and **information_schema** tables, particularly [information_schema.innodb_metrics](#).

See also [INFORMATION_SCHEMA](#).

See also [metrics counter](#).

See also [Performance Schema](#).

covering index

An **index** that includes all the columns retrieved by a query. Instead of using the index values as pointers to find the full table rows, the query returns values from the index structure, saving disk I/O. InnoDB can apply this optimization technique to more indexes than MyISAM can, because

InnoDB **secondary indexes** also include the primary key columns. InnoDB cannot apply this technique for queries against tables modified by a transactions, until that transaction ends.

Any **column index** or **composite index** could act as a covering index, given the right query. Design your indexes and queries to take advantage of this optimization technique wherever possible.

See also [column index](#).

See also [composite index](#).

See also [index](#).

See also [secondary index](#).

crash

MySQL uses the term “crash” to refer generally to any unexpected [shutdown](#) operation where the server cannot do its normal cleanup. For example, a crash could happen due to a hardware fault on the database server machine or storage device; a power failure; a potential data mismatch that causes the MySQL server to halt; a **fast shutdown** initiated by the DBA; or many other reasons. The robust, automatic **crash recovery** for **InnoDB** tables ensures that data is made consistent when the server is restarted, without any extra work for the DBA.

See also [crash recovery](#).

See also [fast shutdown](#).

See also [InnoDB](#).

See also [redo log](#).

See also [shutdown](#).

crash recovery

The cleanup activities that occur when MySQL is started again after a **crash**. For **InnoDB** tables, changes from incomplete transactions are replayed using data from the **redo log**. Changes that were **committed** before the crash, but not yet written into the [data files](#), are reconstructed from the **doublewrite buffer**. When the database is shut down normally, this type of activity is performed during shutdown by the **purge** operation.

During normal operation, committed data can be stored in the **change buffer** for a period of time before being written to the data files. There is always a tradeoff between keeping the data files up-to-date, which introduces performance overhead during normal operation, and buffering the data, which can make shutdown and crash recovery take longer.

See also [change buffer](#).

See also [commit](#).

See also [crash](#).

See also [data files](#).

See also [doublewrite buffer](#).

| | |
|--------------------------|---|
| | See also InnoDB . |
| | See also purge . |
| | See also redo log . |
| CRUD | <p>Acronym for “create, read, update, delete”, a common sequence of operations in database applications. Often denotes a class of applications with relatively simple database usage (basic DDL, DML and query statements in SQL) that can be implemented quickly in any language.</p> <p>See also DDL.</p> <p>See also DML.</p> <p>See also query.</p> <p>See also SQL.</p> |
| cursor | <p>An internal data structure that is used to represent the result set of a query, or other operation that performs a search using an SQL WHERE clause. It works like an iterator in other high-level languages, producing each value from the result set as requested.</p> <p>Although usually SQL handles the processing of cursors for you, you might delve into the inner workings when dealing with performance-critical code.</p> <p>See also query.</p> |
| D | |
| data definition language | See DDL . |
| data dictionary | <p>Metadata that keeps track of InnoDB-related objects such as tables, indexes, and table columns. This metadata is physically located in the InnoDB system tablespace. For historical reasons, it overlaps to some degree with information stored in the .frm files.</p> <p>Because the MySQL Enterprise Backup product always backs up the system tablespace, all backups include the contents of the data dictionary.</p> <p>See also column.</p> <p>See also .frm file.</p> <p>See also hot backup.</p> <p>See also index.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also system tablespace.</p> <p>See also table.</p> |

| | |
|----------------------------|---|
| data directory | <p>The directory under which each MySQL instance keeps the data files for InnoDB and the directories representing individual databases. Controlled by the <code>datadir</code> [492] configuration option.</p> |
| | See also data files . |
| | See also instance . |
| data files | <p>The files that physically contain the InnoDB table and index data. There can be a one-to-many relationship between data files and tables, as in the case of the system tablespace, which can hold multiple InnoDB tables as well as the data dictionary. There can also be a one-to-one relationship between data files and tables, as when the file-per-table setting is enabled, causing each newly created table to be stored in a separate tablespace.</p> |
| | See also data dictionary . |
| | See also file-per-table . |
| | See also index . |
| | See also system tablespace . |
| | See also table . |
| | See also tablespace . |
| data manipulation language | See DML . |
| data warehouse | <p>A database system or application that primarily runs large queries. The read-only or read-mostly data might be organized in denormalized form for query efficiency. Can benefit from the optimizations for read-only transactions in MySQL 5.6 and higher. Contrast with OLTP.</p> |
| | See also denormalized . |
| | See also OLTP . |
| | See also query . |
| | See also read-only transaction . |
| database | <p>Within the MySQL data directory, each database is represented by a separate directory. The InnoDB system tablespace, which can hold table data from multiple databases within a MySQL instance, is kept in its data files that reside outside the individual database directories. When file-per-table mode is enabled, the .ibd files representing individual InnoDB tables are stored inside the database directories.</p> |
| | <p>For long-time MySQL users, a database is a familiar notion. Users coming from an Oracle Database background will find that the MySQL meaning of a database is closer to what Oracle Database calls a schema.</p> |
| | See also data files . |

| | |
|----------|--|
| | See also file-per-table . |
| | See also .ibd file . |
| | See also instance . |
| | See also schema . |
| | See also system tablespace . |
| DCL | <p>Data control language, a set of SQL statements for managing privileges. In MySQL, consists of the GRANT and REVOKE statements. Contrast with DDL and DML.</p> <p>See also DDL.</p> <p>See also DML.</p> <p>See also SQL.</p> |
| DDL | <p>Data definition language, a set of SQL statements for manipulating the database itself rather than individual table rows. Includes all forms of the CREATE, ALTER, and DROP statements. Also includes the TRUNCATE statement, because it works differently than a DELETE FROM table_name statement, even though the ultimate effect is similar.</p> <p>DDL statements automatically commit the current transaction; they cannot be rolled back.</p> <p>The InnoDB-related aspects of DDL include speed improvements for CREATE INDEX and DROP INDEX statements, and the way the file-per-table setting affects the behavior of the TRUNCATE TABLE statement.</p> <p>Contrast with DML and DCL.</p> <p>See also commit.</p> <p>See also DCL.</p> <p>See also DML.</p> <p>See also file-per-table.</p> <p>See also rollback.</p> <p>See also SQL.</p> <p>See also transaction.</p> |
| deadlock | <p>A situation where different transactions are unable to proceed, because each holds a lock that the other needs. Because both transactions are waiting for a resource to become available, neither will ever release the locks it holds.</p> |

A deadlock can occur when the transactions lock rows in multiple tables (through statements such as `UPDATE` or `SELECT ... FOR UPDATE`), but in the opposite order. A deadlock can also occur when such statements lock ranges of index records and **gaps**, with each transaction acquiring some locks but not others due to a timing issue.

To reduce the possibility of deadlocks, use transactions rather than `LOCK TABLE` statements; keep transactions that insert or update data small enough that they do not stay open for long periods of time; when different transactions update multiple tables or large ranges of rows, use the same order of operations (such as `SELECT ... FOR UPDATE`) in each transaction; create indexes on the columns used in `SELECT ... FOR UPDATE` and `UPDATE ... WHERE` statements. The possibility of deadlocks is not affected by the **isolation level**, because the isolation level changes the behavior of read operations, while deadlocks occur because of write operations.

If a deadlock does occur, InnoDB detects the condition and **rolls back** one of the transactions (the **victim**). Thus, even if your application logic is perfectly correct, you must still handle the case where a transaction must be retried. To see the last deadlock in an InnoDB user transaction, use the command `SHOW ENGINE INNODB STATUS`. If frequent deadlocks highlight a problem with transaction structure or application error handling, run with the `innodb_print_all_deadlocks` setting enabled to print information about all deadlocks to the `mysqld` error log.

For background information on how deadlocks are automatically detected and handled, see [Section 14.3.9.8, “Deadlock Detection and Rollback”](#). For tips on avoiding and recovering from deadlock conditions, see [Section 14.3.9.9, “How to Cope with Deadlocks”](#).

See also [concurrency](#).

See also [gap](#).

See also [isolation level](#).

See also [lock](#).

See also [locking](#).

See also [rollback](#).

See also [transaction](#).

See also [victim](#).

deadlock detection

A mechanism that automatically detects when a **deadlock** occurs, and automatically **rolls back** one of the **transactions** involved (the **victim**).

See also [deadlock](#).

See also [rollback](#).

See also [transaction](#).

| | |
|------------------|--|
| | See also victim . |
| delete | <p>When InnoDB processes a <code>DELETE</code> statement, the rows are immediately marked for deletion and no longer are returned by queries. The storage is reclaimed sometime later, during the periodic garbage collection known as the purge operation, performed by a separate thread. For removing large quantities of data, related operations with their own performance characteristics are truncate and drop.</p> <p>See also drop.</p> <p>See also purge.</p> <p>See also truncate.</p> |
| delete buffering | <p>The technique of storing index changes due to <code>DELETE</code> operations in the insert buffer rather than writing them immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally marks an index record for deletion.) It is one of the types of change buffering; the others are insert buffering and purge buffering.</p> <p>See also change buffer.</p> <p>See also change buffering.</p> <p>See also insert buffer.</p> <p>See also insert buffering.</p> <p>See also purge buffering.</p> |
| denormalized | <p>A data storage strategy that duplicates data across different tables, rather than linking the tables with foreign keys and join queries. Typically used in data warehouse applications, where the data is not updated after loading. In such applications, query performance is more important than making it simple to maintain consistent data during updates. Contrast with normalized.</p> <p>See also normalized.</p> |
| descending index | <p>A type of index available with some database systems, where index storage is optimized to process <code>ORDER BY column DESC</code> clauses. Currently, although MySQL allows the <code>DESC</code> keyword in the <code>CREATE TABLE</code> statement, it does not use any special storage layout for the resulting index.</p> <p>See also index.</p> |
| dirty page | <p>A page in the InnoDB buffer pool that has been updated in memory, where the changes are not yet written (flushed) to the data files. The opposite of a clean page.</p> <p>See also buffer pool.</p> |

| | |
|------------|--|
| | <p>See also clean page.</p> <p>See also data files.</p> <p>See also flush.</p> <p>See also page.</p> |
| dirty read | <p>An operation that retrieves unreliable data, data that was updated by another transaction but not yet committed. It is only possible with the isolation level known as read uncommitted.</p> <p>This kind of operation does not adhere to the ACID principle of database design. It is considered very risky, because the data could be rolled back, or updated further before being committed; then, the transaction doing the dirty read would be using data that was never confirmed as accurate.</p> <p>Its polar opposite is consistent read, where InnoDB goes to great lengths to ensure that a transaction does not read information updated by another transaction, even if the other transaction commits in the meantime.</p> <p>See also ACID.</p> <p>See also commit.</p> <p>See also consistent read.</p> <p>See also isolation level.</p> <p>See also READ COMMITTED.</p> <p>See also READ UNCOMMITTED.</p> <p>See also rollback.</p> |
| disk-based | <p>A kind of database that primarily organizes data on disk storage (hard drives or equivalent). Data is brought back and forth between disk and memory to be operated upon. It is the opposite of an in-memory database. Although InnoDB is disk-based, it also contains features such as the buffer pool, multiple buffer pool instances, and the adaptive hash index that allow certain kinds of workloads to work primarily from memory.</p> <p>See also adaptive hash index.</p> <p>See also buffer pool.</p> <p>See also in-memory database.</p> |
| disk-bound | <p>A type of workload where the primary bottleneck is CPU operations in memory. Typically involves read-intensive operations where the results can all be cached in the buffer pool.</p> <p>See also bottleneck.</p> |

| | |
|-------------|---|
| | See also buffer pool . |
| | See also disk-bound . |
| | See also workload . |
| disk-bound | <p>A type of workload where the primary bottleneck is disk I/O. (Also known as I/O-bound.) Typically involves frequent writes to disk, or random reads of more data than can fit into the buffer pool.</p> <p>See also bottleneck.</p> <p>See also buffer pool.</p> <p>See also disk-bound.</p> <p>See also workload.</p> |
| DML | <p>Data manipulation language, a set of SQL statements for performing insert, update, and delete operations. The SELECT statement is sometimes considered as a DML statement, because the SELECT . . . FOR UPDATE form is subject to the same considerations for locking as INSERT, UPDATE, and DELETE.</p> <p>DML statements for an InnoDB table operate in the context of a transaction, so their effects can be committed or rolled back as a single unit.</p> <p>Contrast with DDL and DCL.</p> <p>See also commit.</p> <p>See also DCL.</p> <p>See also DDL.</p> <p>See also locking.</p> <p>See also rollback.</p> <p>See also SQL.</p> <p>See also transaction.</p> |
| document id | <p>In the InnoDB full-text search feature, a special column in the table containing the FULLTEXT index, to uniquely identify the document associated with each ilist value. Its name is FTS_DOC_ID (uppercase required). The column itself must be of BIGINT UNSIGNED NOT NULL type, with a unique index named FTS_DOC_ID_INDEX. Preferably, you define this column when creating the table. If InnoDB must add the column to the table while creating a FULLTEXT index, the indexing operation is considerably more expensive.</p> <p>See also full-text search.</p> <p>See also FULLTEXT index.</p> |

doublewrite buffer

See also [ilist](#).

InnoDB uses a novel file flush technique called doublewrite. Before writing **pages** to the **data files**, InnoDB first writes them to a contiguous area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer have completed, does InnoDB write the pages to their proper positions in the data file. If the operating system crashes in the middle of a page write, InnoDB can later find a good copy of the page from the doublewrite buffer during **crash recovery**.

Although data is always written twice, the doublewrite buffer does not require twice as much I/O overhead or twice as many I/O operations. Data is written to the buffer itself as a large sequential chunk, with a single `fsync()` call to the operating system.

To turn off the doublewrite buffer, specify the option `innodb_doublewrite=0` [1553].

See also [crash recovery](#).

See also [data files](#).

See also [page](#).

See also [purge](#).

drop

A kind of **DDL** operation that removes a schema object, through a statement such as `DROP TABLE` or `DROP INDEX`. It maps internally to an `ALTER TABLE` statement. From an InnoDB perspective, the performance considerations of such operations involve the time that the **data dictionary** is locked to ensure that interrelated objects are all updated, and the time to update memory structures such as the **buffer pool**. For a **table**, the drop operation has somewhat different characteristics than a **truncate** operation (`TRUNCATE TABLE` statement).

See also [buffer pool](#).

See also [data dictionary](#).

See also [DDL](#).

See also [table](#).

See also [truncate](#).

dynamic row format

A row format introduced in the InnoDB Plugin, available as part of the **Barracuda file format**. Because `TEXT` and `BLOB` fields are stored outside of the rest of the page that holds the row data, it is very efficient for rows that include large objects. Since the large fields are typically not accessed to evaluate query conditions, they are not brought into the **buffer pool** as often, resulting in fewer I/O operations and better utilization of cache memory.

See also [Barracuda](#).

See also [buffer pool](#).

See also [file format](#).

See also [row format](#).

E

early adopter A stage similar to beta, when a software product is typically evaluated for performance, functionality, and compatibility in a non-mission-critical setting. InnoDB uses the **early adopter** designation rather than **beta**, through a succession of point releases leading up to a **GA** release.

See also [beta](#).

See also [GA](#).

error log A type of **log** showing information about MySQL startup and critical runtime errors and **crash** information. For details, see [Section 5.2.2, “The Error Log”](#).

See also [crash](#).

See also [log](#).

eviction The process of removing an item from a cache or other temporary storage area, such as the InnoDB **buffer pool**. Often, but not always, uses the **LRU** algorithm to determine which item to remove. When a **dirty page** is evicted, its contents are **flushed** to disk, and any **dirty neighbor** pages might be flushed also.

See also [buffer pool](#).

See also [dirty page](#).

See also [flush](#).

See also [LRU](#).

exclusive lock A kind of **lock** that prevents any other **transaction** from locking the same row. Depending on the transaction **isolation level**, this kind of lock might block other transactions from writing to the same row, or might also block other transactions from reading the same row. The default InnoDB isolation level, **REPEATABLE READ**, enables higher **concurrency** by allowing transactions to read rows that have exclusive locks, a technique known as **consistent read**.

See also [concurrency](#).

See also [consistent read](#).

See also [isolation level](#).

See also [lock](#).

See also [REPEATABLE READ](#).

See also [shared lock](#).

See also [transaction](#).

extent A group of **pages** within a **tablespace** totaling 1 megabyte. With the default **page size** of 16KB, an extent contains 64 pages. In MySQL 5.6, the page size can also be 4KB or 8KB, in which case an extent contains more pages, still adding up to 1MB.

InnoDB features such as **segments**, **read-ahead** requests and the **doublewrite buffer** use I/O operations that read, write, allocate, or free data one extent at a time.

See also [doublewrite buffer](#).

See also [neighbor page](#).

See also [page](#).

See also [page size](#).

See also [read-ahead](#).

See also [segment](#).

See also [tablespace](#).

F

Fast Index Creation A capability first introduced in the InnoDB Plugin, now part of the MySQL server in 5.5 and higher, that speeds up creation of InnoDB **secondary indexes** by avoiding the need to completely rewrite the associated table. The speedup applies to dropping secondary indexes also.

Because index maintenance can add performance overhead to many data transfer operations, consider doing operations such as `ALTER TABLE ... ENGINE=INNODB` or `INSERT INTO ... SELECT * FROM ...` without any secondary indexes in place, and creating the indexes afterward.

In MySQL 5.6, this feature becomes more general: you can read and write to tables while an index is being created, and many more kinds of `ALTER TABLE` operations can be performed without copying the table, without blocking **DML** operations, or both. Thus in MySQL 5.6 and higher, we typically refer to this set of features as **online DDL** rather than Fast Index Creation.

See also [DML](#).

See also [index](#).

See also [online DDL](#).

See also [secondary index](#).

fast shutdown The default **shutdown** procedure for InnoDB, based on the configuration setting `innodb_fast_shutdown=1` [1553]. To save time, certain **flush** operations are skipped. This type of shutdown is safe during normal usage, because the flush operations are performed during the next startup, using the same mechanism as in **crash recovery**. In cases where the database is being shut down for an upgrade or downgrade, do a **slow shutdown** instead to ensure that all relevant changes are applied to the **data files** during the shutdown.

See also [crash recovery](#).

See also [data files](#).

See also [flush](#).

See also [shutdown](#).

See also [slow shutdown](#).

file format

The format used by InnoDB for each table, typically with the **file-per-table** setting enabled so that each table is stored in a separate **.ibd file**. Currently, the file formats available in InnoDB are known as **Antelope** and **Barracuda**. Each file format supports one or more **row formats**. The row formats available for Barracuda tables, **COMPRESSED** and **DYNAMIC**, enable important new storage features for InnoDB tables.

See also [Antelope](#).

See also [Barracuda](#).

See also [file-per-table](#).

See also [.ibd file](#).

See also [ibdata file](#).

See also [row format](#).

file-per-table

A general name for the setting controlled by the `innodb_file_per_table` [1555] option. That is a very important configuration option that affects many aspects of InnoDB file storage, availability of features, and I/O characteristics. In MySQL 5.6.7 and higher, it is enabled by default. Prior to MySQL 5.6.7, it is disabled by default.

For each table created while this setting is in effect, the data is stored in a separate **.ibd file** rather than in the **ibdata files** of the **system tablespace**. When table data is stored in individual files, you have more flexibility to choose nondefault **file formats** and **row formats**, which are required for features such as data **compression**. The `TRUNCATE TABLE` operation is also much faster, and the reclaimed space can be used by the operating system rather than remaining reserved for InnoDB.

The **MySQL Enterprise Backup** product is more flexible for tables that are in their own files. For example, tables can be excluded from a backup, but only if they are in separate files. Thus, this setting is suitable for tables that are backed up less frequently or on a different schedule.

See also [compressed row format](#).

See also [compression](#).

See also [file format](#).

See also [.ibd file](#).

| | |
|------------------|--|
| | See also ibdata file . |
| | See also innodb_file_per_table . |
| | See also row format . |
| | See also system tablespace . |
| fill factor | <p>In an InnoDB index, the proportion of a page that is taken up by index data before the page is split. The unused space when index data is first divided between pages allows for rows to be updated with longer string values without requiring expensive index maintenance operations. If the fill factor is too low, the index consumes more space than needed, causing extra I/O overhead when reading the index. If the fill factor is too high, any update that increases the length of column values can cause extra I/O overhead for index maintenance. See Section 14.3.11.2, “Physical Structure of an InnoDB Index” for more information.</p> <p>See also index.</p> <p>See also page.</p> |
| fixed row format | <p>This row format is used by the MyISAM storage engine, not by InnoDB. If you create an InnoDB table with the option <code>row_format=fixed</code>, InnoDB translates this option to use the compact row format instead, although the <code>fixed</code> value might still show up in output such as <code>SHOW TABLE STATUS</code> reports.</p> <p>See also compact row format.</p> <p>See also row format.</p> |
| flush | <p>To write changes to the database files, that had been buffered in a memory area or a temporary disk storage area. The InnoDB storage structures that are periodically flushed include the redo log, the undo log, and the buffer pool.</p> <p>Flushing can happen because a memory area becomes full and the system needs to free some space, because a commit operation means the changes from a transaction can be finalized, or because a slow shutdown operation means that all outstanding work should be finalized. When it is not critical to flush all the buffered data at once, InnoDB can use a technique called fuzzy checkpointing to flush small batches of pages to spread out the I/O overhead.</p> <p>See also buffer pool.</p> <p>See also commit.</p> <p>See also fuzzy checkpointing.</p> <p>See also neighbor page.</p> <p>See also redo log.</p> <p>See also slow shutdown.</p> |

flush list

See also [undo log](#).

An internal InnoDB data structure that tracks **dirty pages** in the **buffer pool**: that is, **pages** that have been changed and need to be written back out to disk. This data structure is updated frequently by InnoDB's internal **mini-transactions**, and so is protected by its own **mutex** to allow concurrent access to the buffer pool.

See also [buffer pool](#).

See also [dirty page](#).

See also [LRU](#).

See also [mini-transaction](#).

See also [mutex](#).

See also [page](#).

See also [page cleaner](#).

foreign key

A type of pointer relationship, between rows in separate InnoDB tables. The foreign key relationship is defined on one column in both the **parent table** and the **child table**.

In addition to enabling fast lookup of related information, foreign keys help to enforce **referential integrity**, by preventing any of these pointers from becoming invalid as data is inserted, updated, and deleted. This enforcement mechanism is a type of **constraint**. A row that points to another table cannot be inserted if the associated foreign key value does not exist in the other table. If a row is deleted or its foreign key value changed, and rows in another table point to that foreign key value, the foreign key can be set up to prevent the deletion, cause the corresponding column values in the other table to become **null**, or automatically delete the corresponding rows in the other table.

One of the stages in designing a **normalized** database is to identify data that is duplicated, separate that data into a new table, and set up a foreign key relationship so that the multiple tables can be queried like a single table, using a **join** operation.

See also [child table](#).

See also [FOREIGN KEY constraint](#).

See also [join](#).

See also [normalized](#).

See also [NULL](#).

See also [parent table](#).

See also [referential integrity](#).

FOREIGN KEY constraint

See also [relational](#).

The type of **constraint** that maintains database consistency through a **foreign key** relationship. Like other kinds of constraints, it can prevent data from being inserted or updated if data would become inconsistent; in this case, the inconsistency being prevented is between data in multiple tables. Alternatively, when a **DML** operation is performed, **FOREIGN KEY** constraints can cause data in **child rows** to be deleted, changed to different values, or set to **null**, based on the **ON CASCADE** option specified when creating the foreign key.

See also [child table](#).

See also [constraint](#).

See also [DML](#).

See also [foreign key](#).

See also [NULL](#).

.frm file

A file containing the metadata, such as the table definition, of a MySQL table.

Although each InnoDB table has a `.frm` file, InnoDB maintains its own table metadata (the **data dictionary**) in the **system tablespace**; the `.frm` files are not needed for InnoDB to operate on InnoDB tables. Problems can occur if a **crash** happens while the `.frm` file is being written, leading to inconsistencies between the `.frm` file and the data dictionary,

For backups, you must always keep the full set of `.frm` files along with the backup data, to be able to restore tables that are altered or dropped after the backup. Files with this extension are always included in backups produced by the **mysqlbackup** command of the **MySQL Enterprise Backup** product. If you use the **ibbackup** command instead, you must copy the `.frm` files yourself.

These files are backed up by the **MySQL Enterprise Backup** product. These files must not be modified by an **ALTER TABLE** operation while the backup is taking place, which is why backups that include non-InnoDB tables perform a **FLUSH TABLES WITH READ LOCK** operation to freeze such activity while backing up the `.frm` files. Restoring a backup can result in `.frm` files being created, changed, or removed to match the state of the database at the time of the backup.

See also [crash](#).

See also [data dictionary](#).

See also [ibbackup command](#).

See also [MySQL Enterprise Backup](#).

See also [mysqlbackup command](#).

See also [system tablespace](#).

| | |
|------------------|--|
| FTS | <p>In most contexts, an acronym for full-text search. Sometimes in performance discussions, an acronym for full table scan.</p> <p>See also full table scan.</p> <p>See also full-text search.</p> |
| full backup | <p>A backup that includes all the tables in each MySQL database, and all the databases in a MySQL instance. Contrast with partial backup.</p> <p>See also backup.</p> <p>See also database.</p> <p>See also instance.</p> <p>See also partial backup.</p> <p>See also table.</p> |
| full table scan | <p>An operation that requires reading the entire contents of a table, rather than just selected portions using an index. Typically performed either with small lookup tables, or in data warehousing situations with large tables where all available data is aggregated and analyzed. How frequently these operations occur, and the sizes of the tables relative to available memory, have implications for the algorithms used in query optimization and managing the buffer pool.</p> <p>The purpose of indexes is to allow lookups for specific values or ranges of values within a large table, thus avoiding full table scans when practical.</p> <p>See also buffer pool.</p> <p>See also index.</p> <p>See also LRU.</p> |
| full-text search | <p>The MySQL feature for finding words, phrases, Boolean combinations of words, and so on within table data, in a faster, more convenient, and more flexible way than using the SQL LIKE operator or writing your own application-level search algorithm. It uses the SQL function MATCH() [1153] and FULLTEXT indexes.</p> <p>See also FULLTEXT index.</p> |
| FULLTEXT index | <p>The special kind of index that holds the search index in the MySQL full-text search mechanism. Represents the words from values of a column, omitting any that are specified as stopwords. Originally, only available for MyISAM tables. Starting in MySQL 5.6.4, it is also available for InnoDB tables.</p> <p>See also full-text search.</p> <p>See also index.</p> <p>See also InnoDB.</p> <p>See also search index.</p> |

| | |
|---------------------|--|
| | See also stopword . |
| fuzzy checkpointing | <p>A technique that flushes small batches of dirty pages from the buffer pool, rather than flushing all dirty pages at once which would disrupt database processing.</p> <p>See also buffer pool.</p> <p>See also dirty page.</p> <p>See also flush.</p> |
| G | |
| GA | <p>“Generally available”, the stage when a software product leaves beta and is available for sale, official support, and production use.</p> <p>See also beta.</p> <p>See also early adopter.</p> |
| gap | <p>A place in an InnoDB index data structure where new values could be inserted. When you lock a set of rows with a statement such as <code>SELECT ... FOR UPDATE</code>, InnoDB can create locks that apply to the gaps as well as the actual values in the index. For example, if you select all values greater than 10 for update, a gap lock prevents another transaction from inserting a new value that is greater than 10. The supremum record and infimum record represent the gaps containing all values greater than or less than all the current index values.</p> <p>See also concurrency.</p> <p>See also gap lock.</p> <p>See also index.</p> <p>See also infimum record.</p> <p>See also isolation level.</p> <p>See also supremum record.</p> |
| gap lock | <p>A lock on a gap between index records, or a lock on the gap before the first or after the last index record. For example, <code>SELECT c1 FOR UPDATE FROM t WHERE c1 BETWEEN 10 and 20;</code> prevents other transactions from inserting a value of 15 into the column <code>t.c1</code>, whether or not there was already any such value in the column, because the gaps between all existing values in the range are locked. Contrast with record lock and next-key lock.</p> <p>Gap locks are part of the tradeoff between performance and concurrency, and are used in some transaction isolation levels and not others.</p> <p>See also gap.</p> <p>See also infimum record.</p> <p>See also lock.</p> |

| | |
|--------------------|--|
| | See also next-key lock . |
| | See also record lock . |
| | See also supremum record . |
| general log | See general query log . |
| general query log | <p>A type of log used for diagnosis and troubleshooting of SQL statements processed by the MySQL server. Can be stored in a file or in a database table. You must enable this feature through the general_log [502] configuration option to use it. You can disable it for a specific connection through the sql_log_off [556] configuration option.</p> <p>Records a broader range of queries than the slow query log. Unlike the binary log, which is used for replication, the general query log contains SELECT statements and does not maintain strict ordering. For more information, see Section 5.2.3, “The General Query Log”.</p> <p>See also binary log.</p> <p>See also general query log.</p> <p>See also log.</p> |
| global_transaction | <p>A type of transaction involved in XA operations. It consists of several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. For this type of distributed transaction, you must use the SERIALIZABLE isolation level to achieve ACID properties.</p> <p>See also ACID.</p> <p>See also SERIALIZABLE.</p> <p>See also transaction.</p> <p>See also XA.</p> |
| group commit | <p>An InnoDB optimization that performs some low-level I/O operations (log write) once for a set of commit operations, rather than flushing and syncing separately for each commit.</p> <p>When the binlog is enabled, you typically also set the configuration option sync_binlog=0, because group commit for the binary log is only supported if it is set to 0.</p> <p>See also commit.</p> <p>See also plugin.</p> <p>See also XA.</p> |

H

- hash index A type of **index** intended for queries that use equality operators, rather than range operators such as greater-than or [BETWEEN](#). It is available for MEMORY tables. Although hash indexes are the default for MEMORY tables for historic reasons, that storage engine also supports **B-tree** indexes, which are often a better choice for general-purpose queries.
- MySQL includes a variant of this index type, the **adaptive hash index**, that is constructed automatically for InnoDB tables if needed based on runtime conditions.
- See also [adaptive hash index](#).
- See also [B-tree](#).
- See also [index](#).
- See also [InnoDB](#).
- HDD Acronym for “hard disk drive”. Refers to storage media using spinning platters, usually when comparing and contrasting with **SSD**. Its performance characteristics can influence the throughput of a **disk-based** workload.
- See also [disk-based](#).
- See also [SSD](#).
- heartbeat A periodic message that is sent to indicate that a system is functioning properly. In a **replication** context, if the **master** stops sending such messages, one of the **slaves** can take its place. Similar techniques can be used between the servers in a cluster environment, to confirm that all of them are operating properly.
- See also [replication](#).
- high-water mark A value representing an upper limit, either a hard limit that should not be exceeded at runtime, or a record of the maximum value that was actually reached. Contrast with **low-water mark**.
- See also [low-water mark](#).
- history list A list of **transactions** with delete-marked records scheduled to be processed by the [InnoDB purge](#) operation. Recorded in the **undo log**. The length of the history list is reported by the command `SHOW ENGINE INNODB STATUS`. If the history list grows longer than the value of the `innodb_max_purge_lag [1565]` configuration option, each **DML** operation is delayed slightly to allow the purge operation to finish **flushing** the deleted records.
- Also known as **purge lag**.
- See also [flush](#).
- See also [purge](#).
- See also [purge lag](#).
- See also [rollback segment](#).
- See also [transaction](#).

See also [undo log](#).

hot A condition where a row, table, or internal data structure is accessed so frequently, requiring some form of locking or mutual exclusion, that it results in a performance or scalability issue.

Although “hot” typically indicates an undesirable condition, a **hot backup** is the preferred type of backup.

See also [hot backup](#).

hot backup A backup taken while the database and is running and applications are reading and writing to it. The backup involves more than simply copying data files: it must include any data that was inserted or updated while the backup was in process; it must exclude any data that was deleted while the backup was in process; and it must ignore any changes that were not committed.

The Oracle product that performs hot backups, of InnoDB tables especially but also tables from MyISAM and other storage engines, is known as **MySQL Enterprise Backup**.

The hot backup process consists of two stages. The initial copying of the data files produces a **raw backup**. The **apply** step incorporates any changes to the database that happened while the backup was running. Applying the changes produces a **prepared** backup; these files are ready to be restored whenever necessary.

See also [apply](#).

See also [MySQL Enterprise Backup](#).

See also [prepared backup](#).

See also [raw backup](#).

I

I/O-bound See [disk-bound](#).

ib-file set The set of files managed by InnoDB within a MySQL database: the **system tablespace**, any **file-per-table** tablespaces, and the (typically 2) **redo log** files. Used sometimes in detailed discussions of InnoDB file structures and formats, to avoid ambiguity between the meanings of **database** between different DBMS products, and the non-InnoDB files that may be part of a MySQL database.

See also [database](#).

See also [file-per-table](#).

See also [redo log](#).

See also [system tablespace](#).

ib_logfile A set of files, typically named `ib_logfile0` and `ib_logfile1`, that form the **redo log**. Also sometimes referred to as the **log group**. These files record statements that attempt to change data in InnoDB tables.

These statements are replayed automatically to correct data written by incomplete transactions, on startup following a crash.

This data cannot be used for manual recovery; for that type of operation, use the **binary log**.

See also [binary log](#).

See also [log group](#).

See also [redo log](#).

ibbackup command

The fundamental command-line tool of the **MySQL Enterprise Backup** product. It performs a **hot backup** operation for InnoDB tables. You use this command directly if all your data is in InnoDB tables, if all the important data that you need to back up is in InnoDB tables, or if you are running on the Windows platform. If you also need to back up tables from MyISAM or other storage engines, you use the **mysqlbackup command** instead (available on UNIX and Linux systems only).

See also [hot backup](#).

See also [MySQL Enterprise Backup](#).

See also [mysqlbackup command](#).

ibbackup_logfile

A supplemental backup file created by the **MySQL Enterprise Backup** product during a **hot backup** operation. It contains information about any data changes that occurred while the backup was running. The initial backup files, including `ibbackup_logfile`, are known as a **raw backup**, because the changes that occurred during the backup operation are not yet incorporated. After you perform the **apply** step to the raw backup files, the resulting files do include those final data changes, and are known as a **prepared backup**. At this stage, the `ibbackup_logfile` file is no longer necessary.

See also [apply](#).

See also [hot backup](#).

See also [MySQL Enterprise Backup](#).

See also [prepared backup](#).

See also [raw backup](#).

.ibd file

Each InnoDB **table** created using the **file-per-table** mode goes into its own **tablespace** file, with a `.ibd` extension, inside the **database** directory. This file contains the table data and any **indexes** for the table. File-per-table mode, controlled by the **innodb_file_per_table** option, affects many aspects of InnoDB storage usage and performance, and is enabled by default in MySQL 5.6.7 and higher.

This extension does not apply to the **system tablespace**, which consists of the **ibdata files**.

When a `.ibd` file is included in a compressed backup by the **MySQL Enterprise Backup** product, the compressed equivalent is a `.ibz` file.

If a table is create with the `DATA DIRECTORY =` clause in MySQL 5.6 and higher, the `.ibd` file is located outside the normal database directory, and is pointed to by a `.isl` file.

See also [database](#).

See also [file-per-table](#).

See also [ibdata file](#).

See also [.ibz file](#).

See also [index](#).

See also [innodb_file_per_table](#).

See also [.isl file](#).

See also [MySQL Enterprise Backup](#).

See also [system tablespace](#).

See also [table](#).

See also [tablespace](#).

ibdata file

A set of files with names such as `ibdata1`, `ibdata2`, and so on, that make up the InnoDB **system tablespace**. These files contain metadata about InnoDB tables, (the **data dictionary**), and the storage areas for the **undo log**, the **change buffer**, and the **doublewrite buffer**. They also can contain some or all of the table data also (depending on whether the **file-per-table** mode is in effect when each table is created). When the **innodb_file_per_table** option is enabled, data and indexes for newly created tables are stored in separate **.ibd files** rather than in the system tablespace.

The growth of the `ibdata` files is influenced by the `innodb_autoextend_increment` [1548] configuration option.

See also [change buffer](#).

See also [data dictionary](#).

See also [doublewrite buffer](#).

See also [file-per-table](#).

See also [.ibd file](#).

See also [innodb_file_per_table](#).

See also [system tablespace](#).

See also [undo log](#).

| | |
|--------------------|--|
| .ibz file | <p>When the MySQL Enterprise Backup product performs a compressed backup, it transforms each tablespace file that is created using the file-per-table setting from a <code>.ibd</code> extension to a <code>.ibz</code> extension.</p> <p>The compression applied during backup is distinct from the compressed row format that keeps table data compressed during normal operation. A compressed backup operation skips the compression step for a tablespace that is already in compressed row format, as compressing a second time would slow down the backup but produce little or no space savings.</p> <p>See also compressed backup.</p> <p>See also compressed row format.</p> <p>See also file-per-table.</p> <p>See also .ibd file.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also tablespace.</p> |
| ilist | <p>Within an InnoDB FULLTEXT index, the data structure consisting of a document ID and positional information for a token (that is, a particular word).</p> <p>See also FULLTEXT index.</p> |
| implicit row lock | <p>A row lock that InnoDB acquires to ensure consistency, without you specifically requesting it.</p> <p>See also row lock.</p> |
| in-memory database | <p>A type of database system that maintains data in memory, to avoid overhead due to disk I/O and translation between disk blocks and memory areas. Some in-memory databases sacrifice durability (the “D” in the ACID design philosophy) and are vulnerable to hardware, power, and other types of failures, making them more suitable for read-only operations. Other in-memory databases do use durability mechanisms such as logging changes to disk or using non-volatile memory.</p> <p>MySQL features that address the same kinds of memory-intensive processing include the InnoDB buffer pool, adaptive hash index, and read-only transaction optimization, the MEMORY storage engine, the MyISAM key cache, and the MySQL query cache.</p> <p>See also ACID.</p> <p>See also adaptive hash index.</p> <p>See also buffer pool.</p> <p>See also disk-based.</p> <p>See also read-only transaction.</p> |

incremental backup

A type of **hot backup**, performed by the **MySQL Enterprise Backup** product, that only saves data changed since some point in time. Having a full backup and a succession of incremental backups lets you reconstruct backup data over a long period, without the storage overhead of keeping several full backups on hand. You can restore the full backup and then apply each of the incremental backups in succession, or you can keep the full backup up-to-date by applying each incremental backup to it, then perform a single restore operation.

The granularity of changed data is at the **page** level. A page might actually cover more than one row. Each changed page is included in the backup.

See also [hot backup](#).

See also [MySQL Enterprise Backup](#).

See also [page](#).

index

A data structure that provides a fast lookup capability for **rows** of a **table**, typically by forming a tree structure (**B-tree**) representing all the values of a particular **column** or set of columns.

InnoDB tables always have a **clustered index** representing the **primary key**. They can also have one or more **secondary indexes** defined on one or more columns. Depending on their structure, secondary indexes can be classified as **partial**, **column**, or **composite** indexes.

Indexes are a crucial aspect of **query** performance. Database architects design tables, queries, and indexes to allow fast lookups for data needed by applications. The ideal database design uses a **covering index** where practical; the query results are computed entirely from the index, without reading the actual table data. Each **foreign key** constraint also requires an index, to efficiently check whether values exist in both the **parent** and **child** tables.

Although a B-tree index is the most common, a different kind of data structure is used for **hash indexes**, as in the [MEMORY](#) storage engine and the InnoDB **adaptive hash index**.

See also [adaptive hash index](#).

See also [B-tree](#).

See also [child table](#).

See also [clustered index](#).

See also [column index](#).

See also [composite index](#).

See also [covering index](#).

See also [foreign key](#).

See also [hash index](#).

| | |
|------------------|--|
| | See also parent table . |
| | See also partial index . |
| | See also primary key . |
| | See also query . |
| | See also row . |
| | See also secondary index . |
| | See also table . |
| index cache | <p>A memory area that holds the token data for InnoDB full-text search. It buffers the data to minimize disk I/O when data is inserted or updated in columns that are part of a FULLTEXT index. The token data is written to disk when the index cache becomes full. Each InnoDB <code>FULLTEXT</code> index has its own separate index cache, whose size is controlled by the configuration option <code>innodb_ft_cache_size</code>.</p> <p>See also full-text search.</p> <p>See also FULLTEXT index.</p> |
| index hint | <p>Extended SQL syntax for overriding the indexes recommended by the optimizer. For example, the <code>FORCE INDEX</code>, <code>USE INDEX</code>, and <code>IGNORE INDEX</code> clauses. Typically used when indexed columns have unevenly distributed values, resulting in inaccurate cardinality estimates.</p> <p>See also cardinality.</p> <p>See also index.</p> |
| index prefix | <p>In an index that applies to multiple columns (known as a composite index), the initial or leading columns of the index. A query that references the first 1, 2, 3, and so on columns of a composite index can use the index, even if the query does not reference all the columns in the index.</p> <p>See also composite index.</p> <p>See also index.</p> |
| index statistics | See statistics . |
| infimum record | <p>A pseudo-record in an index, representing the gap below the smallest value in that index. If a transaction has a statement such as <code>SELECT ... FOR UPDATE ... WHERE col < 10;</code>, and the smallest value in the column is 5, it is a lock on the infimum record that prevents other transactions from inserting even smaller values such as 0, -10, and so on.</p> <p>See also gap.</p> <p>See also index.</p> |

INFORMATION_SCHEMA

See also [pseudo-record](#).

See also [supremum record](#).

The name of the **database** that provides a query interface to the MySQL **data dictionary**. (This name is defined by the ANSI SQL standard.) To examine information (metadata) about the database, you can query tables such as [INFORMATION_SCHEMA.TABLES](#) and [INFORMATION_SCHEMA.COLUMNS](#), rather than using [SHOW](#) commands that produce unstructured output.

The information schema contains some tables that are specific to **InnoDB**, such as [INNODB_LOCKS](#) and [INNODB_TRX](#). You use these tables not to see how the database is structured, but to get real-time information about the workings of InnoDB tables to help with performance monitoring, tuning, and troubleshooting. In particular, these tables provide information about MySQL features related to **compression**, and **transactions** and their associated **locks**.

See also [compression](#).

See also [data dictionary](#).

See also [database](#).

See also [InnoDB](#).

See also [lock](#).

See also [transaction](#).

innobackup command

A **backup** command from the former **InnoDB Hot Backup** product. It is superseded by the [mysqlbackup](#) command in the **MySQL Enterprise Backup** product.

See also [backup](#).

See also [InnoDB Hot Backup](#).

See also [MySQL Enterprise Backup](#).

See also [mysqlbackup command](#).

InnoDB

A MySQL component that combines high performance with **transactional** capability for reliability, robustness, and concurrent access. It embodies the **ACID** design philosophy. Represented as a **storage engine**; it handles tables created or altered with the [ENGINE=INNODB](#) clause. See [Section 14.3, “The InnoDB Storage Engine”](#) for architectural details and administration procedures, and [Section 8.5, “Optimizing for InnoDB Tables”](#) for performance advice.

In MySQL 5.5 and higher, InnoDB is the default storage engine for new tables and the [ENGINE=INNODB](#) clause is not required. In MySQL 5.1 only, many of the advanced InnoDB features require enabling the component known as the InnoDB Plugin. See [Section 14.3.1, “InnoDB](#)

as the [Default MySQL Storage Engine](#)” for the considerations involved in transitioning to recent releases where InnoDB tables are the default.

InnoDB tables are ideally suited for **hot backups**. See [Section 24.2, “MySQL Enterprise Backup](#)” for information about the **MySQL Enterprise Backup** product for backing up MySQL servers without interrupting normal processing.

See also [ACID](#).

See also [hot backup](#).

See also [storage engine](#).

See also [transaction](#).

InnoDB Hot Backup

A licensed backup product, superseded in MySQL 5.1 and above by **MySQL Enterprise Backup**.

See also [MySQL Enterprise Backup](#).

innodb_autoinc_lock_mode

The [innodb_autoinc_lock_mode \[1548\]](#) option controls the algorithm used for **auto-increment locking**. When you have an auto-incrementing **primary key**, you can use statement-based replication only with the setting [innodb_autoinc_lock_mode=1 \[1548\]](#). This setting is known as **consecutive** lock mode, because multi-row inserts within a transaction receive consecutive auto-increment values. If you have [innodb_autoinc_lock_mode=2](#), which allows higher concurrency for insert operations, use row-based replication rather than statement-based replication. This setting is known as **interleaved** lock mode, because multiple multi-row insert statements running at the same time can receive autoincrement values that are interleaved. The setting [innodb_autoinc_lock_mode=0](#) is the previous (traditional) default setting and should not be used except for compatibility purposes.

See also [auto-increment locking](#).

See also [mixed-mode insert](#).

See also [primary key](#).

innodb_file_format

The [innodb_file_format \[1554\]](#) option determines the **file format** for all InnoDB **tablespaces** created after you specify a value for this option. To create tablespaces other than the **system tablespace**, you must also use the **file-per-table** option. Currently, you can specify the **Antelope** and **Barracuda** file formats.

See also [Antelope](#).

See also [Barracuda](#).

See also [file format](#).

See also [file-per-table](#).

See also [innodb_file_per_table](#).

innodb_file_per_table

See also [system tablespace](#).

See also [tablespace](#).

A very important configuration option that affects many aspects of InnoDB file storage, availability of features, and I/O characteristics. In MySQL 5.6.7 and higher, it is enabled by default. Prior to MySQL 5.6.7, it is disabled by default. The [innodb_file_per_table \[1555\]](#) option turns on **file-per-table** mode, which stores each newly created InnoDB table and its associated index in its own **.ibd file**, outside the **system tablespace**.

This option affects the performance and storage considerations for a number of SQL statements, such as [DROP TABLE](#) and [TRUNCATE TABLE](#).

This option is needed to take full advantage of many other InnoDB features, such as such as table **compression**, or backups of named tables in **MySQL Enterprise Backup**.

This option was once static, but can now be set using the [SET GLOBAL](#) command.

For reference information, see [innodb_file_per_table \[1555\]](#). For usage information, see [Section 14.3.3, “Using Per-Table Tablespaces”](#).

See also [compression](#).

See also [file-per-table](#).

See also [.ibd file](#).

See also [MySQL Enterprise Backup](#).

See also [system tablespace](#).

innodb_lock_wait_timeout

The [innodb_lock_wait_timeout \[1560\]](#) option sets the balance between **waiting** for shared resources to become available, or giving up and handling the error, retrying, or doing alternative processing in your application. Rolls back any InnoDB transaction that waits more than a specified time to acquire a **lock**. Especially useful if **deadlocks** are caused by updates to multiple tables controlled by different storage engines; such deadlocks are not **detected** automatically.

See also [deadlock](#).

See also [deadlock detection](#).

See also [lock](#).

See also [wait](#).

innodb_strict_mode

The [innodb_strict_mode \[1572\]](#) option controls whether InnoDB operates in **strict mode**, where conditions that are normally treated as warnings, cause errors instead (and the underlying statements fail).

This mode is the default setting in MySQL 5.5.5 and higher.

See also [strict mode](#).

insert

One of the primary **DML** operations in **SQL**. The performance of inserts is a key factor in **data warehouse** systems that load millions of rows into tables, and **OLTP** systems where many concurrent connections might insert rows into the same table, in arbitrary order. If insert performance is important to you, you should learn about **InnoDB** features such as the **insert buffer** used in **change buffering**, and **auto-increment** columns.

See also [auto-increment](#).

See also [change buffering](#).

See also [data warehouse](#).

See also [DML](#).

See also [InnoDB](#).

See also [insert buffer](#).

See also [OLTP](#).

See also [SQL](#).

insert buffer

Former name for the **change buffer**. Now that **change buffering** includes delete and update operations as well as inserts, “change buffer” is the preferred term.

See also [change buffer](#).

See also [change buffering](#).

insert buffering

The technique of storing secondary index changes due to **INSERT** operations in the **insert buffer** rather than writing them immediately, so that the physical writes can be performed to minimize random I/O. It is one of the types of **change buffering**; the others are **delete buffering** and **purge buffering**.

Insert buffering is not used if the secondary index is **unique**, because the uniqueness of new values cannot be verified before the new entries are written out. Other kinds of change buffering do work for unique indexes.

See also [change buffer](#).

See also [change buffering](#).

See also [delete buffering](#).

See also [insert buffer](#).

See also [purge buffering](#).

| | |
|--------------------------|--|
| instance | <p>See also unique index.</p> <p>A single mysqld daemon managing a data directory representing one or more databases with a set of tables. It is common in development, testing, and some replication scenarios to have multiple instances on the same server machine, each managing its own data directory and listening on its own port or socket. With one instance running a disk-bound workload, the server might still have extra CPU and memory capacity to run additional instances.</p> <p>See also data directory.</p> <p>See also database.</p> <p>See also disk-bound.</p> <p>See also mysqld.</p> <p>See also replication.</p> <p>See also server.</p> |
| instrumentation | <p>Modifications at the source code level to collect performance data for tuning and debugging. In MySQL, data collected by instrumentation is exposed through a SQL interface using the INFORMATION_SCHEMA and PERFORMANCE_SCHEMA databases.</p> <p>See also INFORMATION_SCHEMA.</p> <p>See also Performance Schema.</p> |
| intention exclusive lock | See intention lock . |
| intention lock | <p>A kind of lock that applies to the table level, used to indicate what kind of lock the transaction intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an intention exclusive (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an intention shared (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible. For more details on this locking mechanism, see Section 14.3.9.1, “InnoDB Lock Modes”.</p> <p>See also lock.</p> <p>See also lock mode.</p> <p>See also locking.</p> |
| intention shared lock | See intention lock . |
| inverted index | <p>A data structure optimized for document retrieval systems, used in the implementation of InnoDB full-text search. The InnoDB FULLTEXT index, implemented as an inverted index, records the position of each word within a document, rather than the location of a table row. A single</p> |

column value (a document stored as a text string) is represented by many entries in the inverted index.

See also [full-text search](#).

See also [FULLTEXT index](#).

See also [ilist](#).

IOPS

Acronym for **I/O operations per second**. A common measurement for busy systems, particularly **OLTP** applications. If this value is near the maximum that the storage devices can handle, the application can become **disk-bound**, limiting **scalability**.

See also [disk-bound](#).

See also [OLTP](#).

See also [scalability](#).

.isl file

A file that specifies the location of a **.ibd file** for an InnoDB table created with the `DATA DIRECTORY =` clause in MySQL 5.6 and higher. It functions like a symbolic link, without the platform restrictions of the actual symbolic link mechanism. You can store InnoDB **tablespaces** outside the **database** directory, for example, on an especially large or fast storage device depending on the usage of the table. For details, see [Specifying the Location of a Tablespace](#).

See also [database](#).

See also [.ibd file](#).

See also [table](#).

See also [tablespace](#).

isolation level

One of the foundations of database processing. Isolation is the **I** in the acronym **ACID**; the isolation level is the setting that fine-tunes the balance between performance and reliability, consistency, and reproducibility of results when multiple **transactions** are making changes and performing queries at the same time.

From highest amount of consistency and protection to the least, the isolation levels supported by InnoDB are: **SERIALIZABLE**, **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.

With InnoDB tables, many users can keep the default isolation level (**REPEATABLE READ**) for all operations. Expert users might choose the **read committed** level as they push the boundaries of scalability with OLTP processing, or during data warehousing operations where minor inconsistencies do not affect the aggregate results of large amounts of data. The levels on the edges (**SERIALIZABLE** and **READ UNCOMMITTED**) change the processing behavior to such an extent that they are rarely used.

See also [ACID](#).

See also [READ COMMITTED](#).

See also [READ UNCOMMITTED](#).

See also [REPEATABLE READ](#).

See also [SERIALIZABLE](#).

See also [transaction](#).

J

join A **query** that retrieves data from more than one table, by referencing columns in the tables that hold identical values. Ideally, these columns are part of an InnoDB **foreign key** relationship, which ensures **referential integrity** and that the join columns are **indexed**. Often used to save space and improve query performance by replacing repeated strings with numeric IDs, in a **normalized** data design.

See also [foreign key](#).

See also [index](#).

See also [normalized](#).

See also [query](#).

See also [referential integrity](#).

K

KEY_BLOCK_SIZE An option to specify the size of data pages within an InnoDB table that uses **compressed row format**. The default is 8 kilobytes. Lower values risk hitting internal limits that depend on the combination of row size and compression percentage.

See also [compressed row format](#).

L

latch A lightweight structure used by InnoDB to implement a **lock** for its own internal memory structures, typically held for a brief time measured in milliseconds or microseconds. A general term that includes both **mutexes** (for exclusive access) and **rw-locks** (for shared access). Certain latches are the focus of InnoDB performance tuning, such as the **data dictionary** mutex. Statistics about latch use and contention are available through the **Performance Schema** interface.

See also [data dictionary](#).

See also [lock](#).

See also [locking](#).

See also [mutex](#).

See also [Performance Schema](#).

See also [rw-lock](#).

| | |
|-----------------|---|
| list | <p>The InnoDB buffer pool is represented as a list of memory pages. The list is reordered as new pages are accessed and enter the buffer pool, as pages within the buffer pool are accessed again and are considered newer, and as pages that are not accessed for a long time are evicted from the buffer pool. The buffer pool is actually divided into sublists, and the replacement policy is a variation of the familiar LRU technique.</p> <p>See also buffer pool.</p> <p>See also eviction.</p> <p>See also LRU.</p> <p>See also sublist.</p> |
| lock | <p>The high-level notion of an object that controls access to a resource, such as a table, row, or internal data structure, as part of a locking strategy. For intensive performance tuning, you might delve into the actual structures that implement locks, such as mutexes and latches.</p> <p>See also latch.</p> <p>See also lock mode.</p> <p>See also locking.</p> <p>See also mutex.</p> |
| lock escalation | <p>An operation used in some database systems that converts many row locks into a single table lock, saving memory space but reducing concurrent access to the table. InnoDB uses a space-efficient representation for row locks, so that lock escalation is not needed.</p> <p>See also locking.</p> <p>See also row lock.</p> <p>See also table lock.</p> |
| lock mode | <p>A shared (S) lock allows a transaction to read a row. Multiple transactions can acquire an S lock on that same row at the same time.</p> <p>An exclusive (X) lock allows a transaction to update or delete a row. No other transaction can acquire any kind of lock on that same row at the same time.</p> <p>Intention locks apply to the table level, and are used to indicate what kind of lock the transaction intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an intention exclusive (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an intention shared (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible.</p> <p>See also intention lock.</p> |

| | |
|--------------|---|
| | See also lock . |
| | See also locking . |
| locking | <p>The system of protecting a transaction from seeing or changing data that is being queried or changed by other transactions. The locking strategy must balance reliability and consistency of database operations (the principles of the ACID philosophy) against the performance needed for good concurrency. Fine-tuning the locking strategy often involves choosing an isolation level and ensuring all your database operations are safe and reliable for that isolation level.</p> <p>See also ACID.</p> <p>See also concurrency.</p> <p>See also isolation level.</p> <p>See also latch.</p> <p>See also lock.</p> <p>See also mutex.</p> <p>See also transaction.</p> |
| locking read | <p>A SELECT statement that also performs a locking operation on an InnoDB table. Either SELECT ... FOR UPDATE or SELECT ... LOCK IN SHARE MODE. It has the potential to produce a deadlock, depending on the isolation level of the transaction. The opposite of a non-locking read. Not allowed for global tables in a read-only transaction.</p> <p>See also deadlock.</p> <p>See also isolation level.</p> <p>See also locking.</p> <p>See also non-locking read.</p> <p>See also read-only transaction.</p> |
| log | <p>In the InnoDB context, “log” or “log files” typically refers to the redo log represented by the ib_logfile* files. Another log area, which is physically part of the system tablespace, is the undo log.</p> <p>Other kinds of logs that are important in MySQL are the error log (for diagnosing startup and runtime problems), binary log (for working with replication and performing point-in-time restores), the general query log (for diagnosing application problems), and the slow query log (for diagnosing performance problems).</p> <p>See also binary log.</p> <p>See also error log.</p> <p>See also general query log.</p> <p>See also ib_logfile.</p> |

| | |
|----------------|--|
| | See also redo log . |
| | See also slow query log . |
| | See also system tablespace . |
| | See also undo log . |
| log buffer | <p>The memory area that holds data to be written to the log files that make up the redo log. It is controlled by the <code>innodb_log_buffer_size</code> [1563] configuration option.</p> <p>See also log file.</p> <p>See also redo log.</p> |
| log file | <p>One of the <code>ib_logfileN</code> files that make up the redo log. Data is written to these files from the log buffer memory area.</p> <p>See also ib_logfile.</p> <p>See also log buffer.</p> <p>See also redo log.</p> |
| log group | <p>The set of files that make up the redo log, typically named <code>ib_logfile0</code> and <code>ib_logfile1</code>. (For that reason, sometimes referred to collectively as ib_logfile.)</p> <p>See also ib_logfile.</p> <p>See also redo log.</p> |
| logical | <p>A type of operation that involves high-level, abstract aspects such as tables, queries, indexes, and other SQL concepts. Typically, logical aspects are important to make database administration and application development convenient and usable. Contrast with physical.</p> <p>See also logical backup.</p> <p>See also physical.</p> |
| logical backup | <p>A backup that reproduces table structure and data, without copying the actual data files. For example, the <code>mysqldump</code> command produces a logical backup, because its output contains statements such as <code>CREATE TABLE</code> and <code>INSERT</code> that can re-create the data. Contrast with physical backup. A logical backup offers flexibility (for example, you could edit table definitions or insert statements before restoring), but can take substantially longer to restore than a physical backup.</p> <p>See also backup.</p> <p>See also mysqldump.</p> <p>See also physical backup.</p> <p>See also restore.</p> |

| | |
|----------------|---|
| loose_ | <p>In MySQL 5.1, a prefix added to InnoDB configuration options when installing the InnoDB Plugin after server startup, so any new configuration options not recognized by the current level of MySQL do not cause a startup failure. MySQL processes configuration options that start with this prefix, but gives a warning rather than a failure if the part after the prefix is not a recognized option.</p> <p>See also plugin.</p> |
| low-water mark | <p>A value representing a lower limit, typically a threshold value at which some corrective action begins or becomes more aggressive. Contrast with high-water mark.</p> <p>See also high-water mark.</p> |
| LRU | <p>An acronym for “least recently used”, a common method for managing storage areas. The items that have not been used recently are evicted when space is needed to cache newer items. InnoDB uses the LRU mechanism by default to manage the pages within the buffer pool, but makes exceptions in cases where a page might be read only a single time, such as during a full table scan. This variation of the LRU algorithm is called the midpoint insertion strategy. The ways in which the buffer pool management differs from the traditional LRU algorithm is fine-tuned by the options innodb_old_blocks_pct [1566], innodb_old_blocks_time [1566], and the new MySQL 5.6 options innodb_lru_scan_depth and innodb_flush_neighbors.</p> <p>See also buffer pool.</p> <p>See also eviction.</p> <p>See also full table scan.</p> <p>See also midpoint insertion strategy.</p> |
| LSN | <p>Acronym for “log sequence number”. This arbitrary, ever-increasing value represents a point in time corresponding to operations recorded in the redo log. (This point in time is regardless of transaction boundaries; it can fall in the middle of one or more transactions.) It is used internally by InnoDB during crash recovery and for managing the buffer pool.</p> <p>In the MySQL Enterprise Backup product, you can specify an LSN to represent the point in time from which to take an incremental backup. The relevant LSN is displayed by the output of the ibbackup command. Once you have the LSN corresponding to the time of a full backup, you can specify that value to take a subsequent incremental backup, whose output contains another LSN for the next incremental backup.</p> <p>See also crash recovery.</p> <p>See also incremental backup.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also redo log.</p> <p>See also transaction.</p> |

M

- master server** Frequently shortened to “master”. A database server machine in a **replication** scenario that processes the initial insert, update, and delete requests for data. These changes are propagated to, and repeated on, other servers known as **slave servers**.
- See also [replication](#).
- See also [slave server](#).
- master thread** An InnoDB **thread** that performs various tasks in the background. Most of these tasks are I/O related, such as writing changes from the **insert buffer** to the appropriate secondary indexes.
- To improve **concurrency**, sometimes actions are moved from the master thread to separate background threads. For example, in MySQL 5.6 and higher, **dirty pages** are **flushed** from the **buffer pool** by the **page cleaner** thread rather than the master thread.
- See also [buffer pool](#).
- See also [dirty page](#).
- See also [flush](#).
- See also [insert buffer](#).
- See also [page cleaner](#).
- See also [thread](#).
- MDL** Acronym for “metadata lock”.
- See also [metadata lock](#).
- memcached** A popular component of many MySQL and **NoSQL** software stacks, allowing fast reads and writes for single values and caching the results entirely in memory. Traditionally, applications required extra logic to write the same data to a MySQL database for permanent storage, or to read data from a MySQL database when it was not cached yet in memory. Now, applications can use the simple [memcached](#) protocol, supported by client libraries for many languages, to communicate directly with MySQL servers using **InnoDB** or MySQL Cluster tables. These NoSQL interfaces to MySQL tables allow applications to achieve higher read and write performance than by issuing SQL commands directly, and can simplify application logic and deployment configurations for systems that already incorporated [memcached](#) for in-memory caching.
- The [memcached](#) interface to InnoDB tables is available in MySQL 5.6 and higher; see [InnoDB Integration with memcached](#) for details. The [memcached](#) interface to MySQL Cluster tables is available in MySQL Cluster 7.2; see <http://dev.mysql.com/doc/ndbapi/en/ndbmemcache.html> for details.
- See also [InnoDB](#).

| | |
|-----------------------------|--|
| | See also NoSQL . |
| merge | <p>To apply changes to data cached in memory, such as when a page is brought into the buffer pool, and any applicable changes recorded in the change buffer are incorporated into the page in the buffer pool. The updated data is eventually written to the tablespace by the flush mechanism.</p> <p>See also buffer pool.</p> <p>See also change buffer.</p> <p>See also flush.</p> <p>See also tablespace.</p> |
| metadata lock | <p>A type of lock that prevents DDL operations on a table that is being used at the same time by another transaction. For details, see Section 8.10.4, “Metadata Locking Within Transactions”.</p> <p>Enhancements to online operations, particularly in MySQL 5.6 and higher, are focused on reducing the amount of metadata locking. The objective is for DDL operations that do not change the table structure (such as <code>CREATE INDEX</code> and <code>DROP INDEX</code> for InnoDB tables) to proceed while the table is being queried, updated, and so on by other transactions.</p> <p>See also DDL.</p> <p>See also lock.</p> <p>See also online.</p> <p>See also transaction.</p> |
| metrics counter | <p>A feature implemented by the <code>innodb_metrics</code> table in the information_schema, in MySQL 5.6 and higher. You can query counts and totals for low-level InnoDB operations, and use the results for performance tuning in combination with data from the performance_schema.</p> <p>See also counter.</p> <p>See also INFORMATION_SCHEMA.</p> <p>See also Performance Schema.</p> |
| midpoint insertion strategy | <p>The technique of initially bringing pages into the InnoDB buffer pool not at the “newest” end of the list, but instead somewhere in the middle. The exact location of this point can vary, based on the setting of the <code>innodb_old_blocks_pct [1566]</code> option. The intent is that blocks that are only read once, such as during a full table scan, can be aged out of the buffer pool sooner than with a strict LRU algorithm.</p> <p>See also buffer pool.</p> |

| | |
|-------------------|---|
| | See also full table scan . |
| | See also LRU . |
| | See also page . |
| mini-transaction | <p>An internal phase of InnoDB processing, when making changes at the physical level to internal data structures during DML operations. A mini-transaction has no notion of rollback; multiple mini-transactions can occur within a single transaction. Mini-transactions write information to the redo log that is used during crash recovery. A mini-transaction can also happen outside the context of a regular transaction, for example during purge processing by background threads.</p> <p>See also commit.</p> <p>See also crash recovery.</p> <p>See also DML.</p> <p>See also physical.</p> <p>See also purge.</p> <p>See also redo log.</p> <p>See also rollback.</p> <p>See also transaction.</p> |
| mixed-mode insert | <p>An INSERT statement where auto-increment values are specified for some but not all of the new rows. For example, a multi-value INSERT could specify a value for the auto-increment column in some cases and NULL in other cases. InnoDB generates auto-increment values for the rows where the column value was specified as NULL. Another example is an INSERT ... ON DUPLICATE KEY UPDATE statement, where auto-increment values might be generated but not used, for any duplicate rows that are processed as UPDATE rather than INSERT statements.</p> <p>Can cause consistency issues between master and slave servers in a replication configuration. Can require adjusting the value of the innodb_autoinc_lock_mode configuration option.</p> <p>See also auto-increment.</p> <p>See also innodb_autoinc_lock_mode.</p> <p>See also master server.</p> <p>See also replication.</p> <p>See also slave server.</p> |
| .MRG file | <p>A file containing references to other tables, used by the MERGE storage engine. Files with this extension are always included in backups</p> |

| | |
|----------------------------------|---|
| | <p>produced by the mysqlbackup command of the MySQL Enterprise Backup product.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also mysqlbackup command.</p> |
| multi-core | <p>A type of processor that can take advantage of multi-threaded programs, such as the MySQL server.</p> |
| multiversion concurrency control | <p>See MVCC.</p> |
| mutex | <p>Informal abbreviation for “mutex variable”. (Mutex itself is short for “mutual exclusion”.) The low-level object that InnoDB uses to represent and enforce exclusive-access locks to internal in-memory data structures. Once the lock is acquired, any other process, thread, and so on is prevented from acquiring the same lock. Contrast with rw-locks, which allow shared access. Mutexes and rw-locks are known collectively as latches.</p> <p>See also latch.</p> <p>See also lock.</p> <p>See also Performance Schema.</p> <p>See also Pthreads.</p> <p>See also rw-lock.</p> |
| MVCC | <p>Acronym for “multiversion concurrency control”. This technique lets InnoDB transactions with certain isolation levels to perform consistent read operations; that is, to query rows that are being updated by other transactions, and see the values from before those updates occurred. This is a powerful technique to increase concurrency, by allowing queries to proceed without waiting due to locks held by the other transactions.</p> <p>This technique is not universal in the database world. Some other database products, and some other MySQL storage engines, do not support it.</p> <p>See also ACID.</p> <p>See also concurrency.</p> <p>See also consistent read.</p> <p>See also isolation level.</p> <p>See also lock.</p> <p>See also transaction.</p> |
| my.cnf | <p>The name, on UNIX or Linux systems, of the MySQL option file.</p> <p>See also my.ini.</p> |

| | |
|-------------------------|---|
| | See also option file . |
| my.ini | The name, on Windows systems, of the MySQL option file.

See also my.cnf .

See also option file . |
| .MYD file | The file that MySQL uses to store data for a MyISAM table. Files with this extension are always included in backups produced by the mysqlbackup command of the MySQL Enterprise Backup product.

See also .MYI file .

See also MySQL Enterprise Backup .

See also mysqlbackup command . |
| .MYI file | A file that MySQL uses to store indexes for a MyISAM table. Files with this extension are always included in backups produced by the mysqlbackup command of the MySQL Enterprise Backup product.

See also .MYD file .

See also MySQL Enterprise Backup .

See also mysqlbackup command . |
| mysql | The mysql program is the command-line interpreter for the MySQL database. It processes SQL statements, and also MySQL-specific commands such as <code>SHOW TABLES</code> , by passing requests to the mysqld daemon.

See also mysqld .

See also SQL . |
| MySQL Enterprise Backup | A licensed product, superseding InnoDB Hot Backup, that performs hot backups of MySQL databases. It offers the most efficiency and flexibility when backing up InnoDB tables, but can also back up MyISAM and other kinds of tables.

See also hot backup .

See also InnoDB . |
| mysqlbackup command | A command-line tool of the MySQL Enterprise Backup product. It performs a hot backup operation for InnoDB tables, and a warm backup for MyISAM and other kinds of tables. See Section 24.2, “MySQL Enterprise Backup” for more information about this command.

See also hot backup .

See also ibbackup command .

See also MySQL Enterprise Backup . |

| | |
|-----------|---|
| | See also warm backup . |
| mysqld | The <code>mysqld</code> program is the database engine for the MySQL database. It runs as a UNIX daemon or Windows service, constantly waiting for requests and performing maintenance work in the background. |
| | See also mysql . |
| mysqldump | A command that performs a logical backup of some combination of databases, tables, and table data. The results are SQL statements that reproduce the original schema objects, data, or both. For substantial amounts of data, a physical backup solution such as MySQL Enterprise Backup is faster, particularly for the restore operation. |
| | See also logical backup . |
| | See also MySQL Enterprise Backup . |
| | See also physical backup . |
| | See also restore . |

N

| | |
|---------------|---|
| natural key | <p>A indexed column, typically a primary key, where the values have some real-world significance. Usually advised against because:</p> <ul style="list-style-type: none"> • If the value should ever change, there is potentially a lot of index maintenance to re-sort the clustered index and update the copies of the primary key value that are repeated in each secondary index. • Even seemingly stable values can change in unpredictable ways that are difficult to represent correctly in the database. For example, one country can change into two or several, making the original country code obsolete. Or, rules about unique values might have exceptions. For example, even if taxpayer IDs are intended to be unique to a single person, a database might have to handle records that violate that rule, such as in cases of identity theft. Taxpayer IDs and other sensitive ID numbers also make poor primary keys, because they may need to be secured, encrypted, and otherwise treated differently than other columns. <p>Thus, it is typically better to use arbitrary numeric values to form a synthetic key, for example using an auto-increment column.</p> <p>See also auto-increment.</p> <p>See also primary key.</p> <p>See also secondary index.</p> <p>See also synthetic key.</p> |
| neighbor page | Any page in the same extent as a particular page. When a page is selected to be flushed , any neighbor pages that are dirty are typically flushed as well, as an I/O optimization for traditional hard disks. In MySQL 5.6 and up, this behavior can be controlled by the configuration variable |

[innodb_flush_neighbors](#); you might turn that setting off for SSD drives, which do not have the same overhead for writing smaller batches of data at random locations.

See also [dirty page](#).

See also [extent](#).

See also [flush](#).

See also [page](#).

next-key lock A combination of a **record lock** on the index record and a [gap lock](#) on the gap before the index record.

See also [gap lock](#).

See also [locking](#).

See also [record lock](#).

non-blocking I/O An industry term that means the same as **asynchronous I/O**.

See also [asynchronous I/O](#).

non-locking read A **query** that does not use the [SELECT ... FOR UPDATE](#) or [SELECT ... LOCK IN SHARE MODE](#) clauses. The only kind of query allowed for global tables in a **read-only transaction**. The opposite of a **locking read**.

See also [locking read](#).

See also [query](#).

See also [read-only transaction](#).

non-repeatable read The situation when a query retrieves data, and a later query within the same **transaction** retrieves what should be the same data, but the queries return different results (changed by another transaction committing in the meantime).

This kind of operation goes against the **ACID** principle of database design. Within a transaction, data should be consistent, with predictable and stable relationships.

Among different **isolation levels**, non-repeatable reads are prevented by the **serializable read** and **repeatable read** levels, and allowed by the **consistent read**, and **read uncommitted** levels.

See also [ACID](#).

See also [consistent read](#).

See also [isolation level](#).

See also [READ UNCOMMITTED](#).

See also [REPEATABLE READ](#).

See also [SERIALIZABLE](#).

normalized

See also [transaction](#).

A database design strategy where data is split into multiple tables, and duplicate values condensed into single rows represented by an ID, to avoid storing, querying, and updating redundant or lengthy values. It is typically used in **OLTP** applications.

For example, an address might be given a unique ID, so that a census database could represent the relationship **lives at this address** by associating that ID with each member of a family, rather than storing multiple copies of a complex value such as **123 Main Street, Anytown, USA**.

For another example, although a simple address book application might store each phone number in the same table as a person's name and address, a phone company database might give each phone number a special ID, and store the numbers and IDs in a separate table. This normalized representation could simplify large-scale updates when area codes split apart.

Normalization is not always recommended. Data that is primarily queried, and only updated by deleting entirely and reloading, is often kept in fewer, larger tables with redundant copies of duplicate values. This data representation is referred to as **denormalized**, and is frequently found in data warehousing applications.

See also [denormalized](#).

See also [foreign key](#).

See also [OLTP](#).

See also [relational](#).

NoSQL

A broad term for a set of data access technologies that do not use the **SQL** language as their primary mechanism for reading and writing data. Some NoSQL technologies act as key-value stores, only accepting single-value reads and writes; some relax the restrictions of the **ACID** methodology; still others do not require a pre-planned **schema**. MySQL users can combine NoSQL-style processing for speed and simplicity with SQL operations for flexibility and convenience, by using the **memcached** API to directly access some kinds of MySQL tables. The [memcached](#) interface to InnoDB tables is available in MySQL 5.6 and higher; see [InnoDB Integration with memcached](#) for details. The [memcached](#) interface to MySQL Cluster tables is available in MySQL Cluster 7.2; see <http://dev.mysql.com/doc/ndbapi/en/ndbmemcache.html> for details.

See also [ACID](#).

See also [InnoDB](#).

See also [memcached](#).

See also [schema](#).

See also [SQL](#).

NOT NULL constraint A type of **constraint** that specifies that a **column** cannot contain any **NULL** values. It helps to preserve **referential integrity**, as the database server can identify data with erroneous missing values. It also helps in the arithmetic involved in query optimization, allowing the optimizer to predict the number of entries in an index on that column.

See also [column](#).

See also [constraint](#).

See also [NULL](#).

See also [primary key](#).

See also [referential integrity](#).

NULL A special value in **SQL**, indicating the absence of data. Any arithmetic operation or equality test involving a **NULL** value, in turn produces a **NULL** result. (Thus it is similar to the IEEE floating-point concept of NaN, “not a number”.) Any aggregate calculation such as **AVG ()** ignores rows with **NULL** values, when determining how many rows to divide by. The only test that works with **NULL** values uses the SQL idioms **IS NULL** or **IS NOT NULL**.

NULL values play a part in index operations, because for performance a database must minimize the overhead of keeping track of missing data values. Typically, **NULL** values are not stored in an index, because a query that tests an indexed column using a standard comparison operator could never match a row with a **NULL** value for that column. For the same reason, unique indexes do not prevent **NULL** values; those values simply are not represented in the index. Declaring a **NOT NULL** constraint on a column provides reassurance that there are no rows left out of the index, allowing for better query optimization (accurate counting of rows and estimation of whether to use the index).

Because the **primary key** must be able to uniquely identify every row in the table, a single-column primary key cannot contain any **NULL** values, and a multi-column primary key cannot contain any rows with **NULL** values in all columns.

Although the Oracle database allows a **NULL** value to be concatenated with a string, InnoDB treats the result of such an operation as **NULL**.

See also [index](#).

See also [primary key](#).

See also [SQL](#).

O

off-page column A column containing variable-length data (such as **BLOB** and **VARCHAR**) that is too long to fit on a **B-tree** page. The data is stored in **overflow pages**. The **DYNAMIC** row format in the InnoDB **Barracuda** file format is more efficient for such storage than the older **COMPACT** row format.

See also [B-tree](#).

See also [Barracuda](#).

See also [overflow page](#).

OLTP

Acronym for “Online Transaction Processing”. A database system, or a database application, that runs a workload with many **transactions**, with frequent writes as well as reads, typically affecting small amounts of data at a time. For example, an airline reservation system or an application that processes bank deposits. The data might be organized in **normalized** form for a balance between **DML** (insert/update/delete) efficiency and **query** efficiency. Contrast with **data warehouse**.

With its **row-level locking** and **transactional** capability, **InnoDB** is the ideal storage engine for MySQL tables used in OLTP applications.

See also [data warehouse](#).

See also [DML](#).

See also [InnoDB](#).

See also [query](#).

See also [row lock](#).

See also [transaction](#).

online

A type of operation that involves no downtime, blocking, or restricted operation for the database. Typically applied to **DDL**. Operations that shorten the periods of restricted operation, such as **fast index creation**, have evolved into a wider set of **online DDL** operations in MySQL 5.6.

In the context of backups, a **hot backup** is an online operation and a **warm backup** is partially an online operation.

See also [DDL](#).

See also [Fast Index Creation](#).

See also [hot backup](#).

See also [online DDL](#).

See also [warm backup](#).

online DDL

A feature that improves the performance, concurrency, and availability of InnoDB tables during **DDL** (primarily [ALTER TABLE](#)) operations. See [Online DDL for InnoDB Tables](#) for details.

The details vary according to the type of operation. In some cases, the table can be modified concurrently while the [ALTER TABLE](#) is in progress. The operation might be able to be performed without doing a table copy, or using a specially optimized type of table copy. Space usage is controlled by the [innodb_online_alter_log_max_size](#) configuration option.

This feature is an enhancement of the **Fast Index Creation** feature in MySQL 5.5 and the InnoDB Plugin for MySQL 5.1.

| | |
|----------------------------|--|
| | See also DDL . |
| | See also Fast Index Creation . |
| | See also online . |
| .OPT file | <p>A file containing database configuration information. Files with this extension are always included in backups produced by the mysqlbackup command of the MySQL Enterprise Backup product.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also mysqlbackup command.</p> |
| optimistic | <p>A methodology that guides low-level implementation decisions for a relational database system. The requirements of performance and concurrency in a relational database mean that operations must be started or dispatched quickly. The requirements of consistency and referential integrity mean that any operation could fail: a transaction might be rolled back, a DML operation could violate a constraint, a request for a lock could cause a deadlock, a network error could cause a timeout. An optimistic strategy is one that assumes most requests or attempts will succeed, so that relatively little work is done to prepare for the failure case. When this assumption is true, the database does little unnecessary work; when requests do fail, extra work must be done to clean up and undo changes.</p> <p>InnoDB uses optimistic strategies for operations such as locking and commits. For example, data changed by a transaction can be written to the data files before the commit occurs, making the commit itself very fast, but requiring more work to undo the changes if the transaction is rolled back.</p> <p>The opposite of an optimistic strategy is a pessimistic one, where a system is optimized to deal with operations that are unreliable and frequently unsuccessful. This methodology is rare in a database system, because so much care goes into choosing reliable hardware, networks, and algorithms.</p> <p>See also commit.</p> <p>See also concurrency.</p> <p>See also DML.</p> <p>See also locking.</p> <p>See also pessimistic.</p> |
| optimizer | <p>The MySQL component that determines the best indexes and join order to use for a query, based on characteristics and data distribution of the relevant tables.</p> <p>See also index.</p> <p>See also join.</p> <p>See also query.</p> <p>See also table.</p> |

| | |
|---------------|--|
| option | <p>A configuration parameter for MySQL, either stored in the option file or passed on the command line.</p> <p>For the options that apply to InnoDB tables, each option name starts with the prefix innodb_.</p> <p>See also InnoDB.</p> <p>See also option file.</p> |
| option file | <p>The file that holds the configuration options for the MySQL instance. Traditionally, on Linux and UNIX this file is named <code>my.cnf</code>, and on Windows it is named <code>my.ini</code>.</p> <p>See also configuration file.</p> <p>See also my.cnf.</p> <p>See also option.</p> |
| overflow page | <p>Separately allocated disk pages that hold variable-length columns (such as BLOB and VARCHAR) that are too long to fit on a B-tree page. The associated columns are known as off-page columns.</p> <p>See also B-tree.</p> <p>See also off-page column.</p> <p>See also page.</p> |

P

| | |
|------|---|
| page | <p>A unit representing how much data InnoDB transfers at any one time between disk (the data files) and memory (the buffer pool). A page can contain one or more rows, depending on how much data is in each row. If a row does not fit entirely into a single page, InnoDB sets up additional pointer-style data structures so that the information about the row can be stored in one page.</p> <p>One way to fit more data in each page is to use compressed row format. For tables that use BLOBs or large text fields, compact row format allows those large columns to be stored separately from the rest of the row, reducing I/O overhead and memory usage for queries that do not reference those columns.</p> <p>When InnoDB reads or writes sets of pages as a batch to increase I/O throughput, it reads or writes an extent at a time.</p> <p>All the InnoDB disk data structures within a MySQL instance share the same page size.</p> <p>See also buffer pool.</p> <p>See also compact row format.</p> <p>See also compressed row format.</p> |
|------|---|

| | |
|--------------|---|
| | See also extent . |
| | See also page size . |
| | See also row . |
| page cleaner | <p>An InnoDB background thread that flushes dirty pages from the buffer pool. Prior to MySQL 5.6, this activity was performed by the master thread</p> <p>See also buffer pool.</p> <p>See also dirty page.</p> <p>See also flush.</p> <p>See also master thread.</p> <p>See also thread.</p> |
| page size | <p>For releases up to and including MySQL 5.5, the size of each InnoDB page is fixed at 16 kilobytes. This value represents a balance: large enough to hold the data for most rows, yet small enough to minimize the performance overhead of transferring unneeded data to memory. Other values are not tested or supported.</p> <p>Starting in MySQL 5.6, the page size for an InnoDB instance can be either 4KB, 8KB, or 16KB, controlled by the innodb_page_size configuration option. You set the size when creating the MySQL instance, and it remains constant afterwards. The same page size applies to all InnoDB tablespaces, both the system tablespace and any separate tablespaces created in file-per-table mode.</p> <p>Smaller page sizes can help performance with storage devices that use small block sizes, particularly for SSD devices in disk-bound workloads, such as for OLTP applications. As individual rows are updated, less data is copied into memory, written to disk, reorganized, locked, and so on.</p> <p>See also disk-bound.</p> <p>See also file-per-table.</p> <p>See also instance.</p> <p>See also OLTP.</p> <p>See also page.</p> <p>See also SSD.</p> <p>See also system tablespace.</p> <p>See also tablespace.</p> |
| .PAR file | <p>A file containing partition definitions. Files with this extension are always included in backups produced by the mysqlbackup command of the MySQL Enterprise Backup product.</p> |

| | |
|-----------------------|---|
| | See also MySQL Enterprise Backup . |
| | See also mysqlbackup command . |
| parent table | <p>The table in a foreign key relationship that holds the initial column values pointed to from the child table. The consequences of deleting, or updating rows in the parent table depend on the ON UPDATE and ON DELETE clauses in the foreign key definition. Rows with corresponding values in the child table could be automatically deleted or updated in turn, or those columns could be set to NULL, or the operation could be prevented.</p> <p>See also child table.</p> <p>See also foreign key.</p> |
| partial backup | <p>A backup that contains some of the tables in a MySQL database, or some of the databases in a MySQL instance. Contrast with full backup.</p> <p>See also backup.</p> <p>See also full backup.</p> <p>See also table.</p> |
| partial index | <p>An index that represents only part of a column value, typically the first N characters (the prefix) of a long VARCHAR value.</p> <p>See also index.</p> <p>See also index prefix.</p> |
| Performance Schema | <p>The performance_schema schema, in MySQL 5.5 and up, presents a set of tables that you can query to get detailed information about the performance characteristics of many internal parts of the MySQL server.</p> <p>See also latch.</p> <p>See also mutex.</p> <p>See also rw-lock.</p> |
| persistent statistics | <p>A feature in MySQL 5.6 that stores index statistics for InnoDB tables on disk, providing better plan stability for queries. The statistics are recalculated only by running the ALTER TABLE statement.</p> <p>See also index.</p> <p>See also optimizer.</p> <p>See also plan stability.</p> <p>See also query.</p> <p>See also table.</p> |
| pessimistic | <p>A methodology that sacrifices performance or concurrency in favor of safety. It is appropriate if a high proportion of requests or attempts might</p> |

fail, or if the consequences of a failed request are severe. InnoDB uses what is known as a pessimistic **locking** strategy, to minimize the chance of **deadlocks**. At the application level, you might avoid deadlocks by using a pessimistic strategy of acquiring all locks needed by a transaction at the very beginning.

Many built-in database mechanisms use the opposite **optimistic** methodology.

See also [deadlock](#).

See also [locking](#).

See also [optimistic](#).

phantom

A row that appears in the result set of a query, but not in the result set of an earlier query. For example, if a query is run twice within a **transaction**, and in the meantime, another transaction commits after inserting a new row or updating a row so that it matches the [WHERE](#) clause of the query.

This occurrence is known as a phantom read. It is harder to guard against than a **non-repeatable read**, because locking all the rows from the first query result set does not prevent the changes that cause the phantom to appear.

Among different **isolation levels**, phantom reads are prevented by the **serializable read** level, and allowed by the **repeatable read**, **consistent read**, and **read uncommitted** levels.

See also [consistent read](#).

See also [isolation level](#).

See also [non-repeatable read](#).

See also [READ UNCOMMITTED](#).

See also [REPEATABLE READ](#).

See also [SERIALIZABLE](#).

See also [transaction](#).

physical

A type of operation that involves hardware-related aspects such as disk blocks, memory pages, files, bits, disk reads, and so on. Typically, physical aspects are important during expert-level performance tuning and problem diagnosis. Contrast with **logical**.

See also [logical](#).

See also [physical backup](#).

physical backup

A **backup** that copies the actual data files. For example, the [mysqlbackup](#) command of the **MySQL Enterprise Backup** product produces a physical backup, because its output contains data files that can be used directly by the [mysqld](#) server, resulting in a faster **restore** operation. Contrast with **logical backup**.

| | |
|------------------------|---|
| | See also backup . |
| | See also logical backup . |
| | See also MySQL Enterprise Backup . |
| | See also restore . |
| PITR | Acronym for point-in-time recovery . |
| | See also point-in-time recovery . |
| plan stability | A property of a query execution plan , where the optimizer makes the same choices each time for a given query , so that performance is consistent and predictable. |
| | See also query . |
| | See also query execution plan . |
| plugin | In MySQL 5.1 and earlier, a separately installable form of the InnoDB storage engine that includes features and performance enhancements not included in the built-in InnoDB for those releases. |
| | For MySQL 5.5 and higher, the MySQL distribution includes the very latest InnoDB features and performance enhancements, known as InnoDB 1.1, and there is no longer a separate InnoDB Plugin. |
| | This distinction is important mainly in MySQL 5.1, where a feature or bug fix might apply to the InnoDB Plugin but not the built-in InnoDB, or vice versa. |
| | See also built-in . |
| | See also InnoDB . |
| point-in-time recovery | The process of restoring a backup to recreate the state of the database at a specific date and time. Commonly abbreviated PITR . Because it is unlikely that the specified time corresponds exactly to the time of a backup, this technique usually requires a combination of a physical backup and a logical backup . For example, with the MySQL Enterprise Backup product, you restore the last backup that you took before the specified point in time, then replay changes from the binary log between the time of the backup and the PITR time. |
| | See also backup . |
| | See also logical backup . |
| | See also MySQL Enterprise Backup . |
| | See also physical backup . |
| | See also PITR . |
| prefix | See index prefix . |

| | |
|-----------------|--|
| prepared backup | <p>A set of backup files, produced by the MySQL Enterprise Backup product, after all the stages of applying binary logs and incremental backups are finished. The resulting files are ready to be restored. Prior to the apply steps, the files are known as a raw backup.</p> <p>See also binary log.</p> <p>See also hot backup.</p> <p>See also incremental backup.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also raw backup.</p> <p>See also restore.</p> |
| primary key | <p>A set of columns -- and by implication, the index based on this set of columns -- that can uniquely identify every row in a table. As such, it must be a unique index that does not contain any NULL values.</p> <p>InnoDB requires that every table has such an index (also called the clustered index or cluster index), and organizes the table storage based on the column values of the primary key.</p> <p>When choosing primary key values, consider using arbitrary values (a synthetic key) rather than relying on values derived from some other source (a natural key).</p> <p>See also clustered index.</p> <p>See also index.</p> <p>See also natural key.</p> <p>See also synthetic key.</p> |
| process | <p>An instance of an executing program. The operating switches between multiple running processes, allowing for a certain degree of concurrency. On most operating systems, processes can contain multiple threads of execution that share resources. Context-switching between threads is faster than the equivalent switching between processes.</p> <p>See also concurrency.</p> <p>See also thread.</p> |
| pseudo-record | <p>An artificial record in an index, used for locking key values or ranges that do not currently exist.</p> <p>See also infimum record.</p> <p>See also locking.</p> <p>See also supremum record.</p> |

| | |
|-----------------|--|
| Pthreads | <p>The POSIX threads standard, which defines an API for threading and locking operations on UNIX and Linux systems. On UNIX and Linux systems, InnoDB uses this implementation for mutexes.</p> <p>See also mutex.</p> |
| purge | <p>A type of garbage collection performed by a separate thread, running on a periodic schedule. The purge includes these actions: removing obsolete values from indexes; physically removing rows that were marked for deletion by previous DELETE statements.</p> <p>See also crash recovery.</p> <p>See also delete.</p> <p>See also doublewrite buffer.</p> |
| purge buffering | <p>The technique of storing index changes due to DELETE operations in the insert buffer rather than writing them immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally purges an index record that was previously marked for deletion.) It is one of the types of change buffering; the others are insert buffering and delete buffering.</p> <p>See also change buffer.</p> <p>See also change buffering.</p> <p>See also delete buffering.</p> <p>See also insert buffer.</p> <p>See also insert buffering.</p> |
| purge lag | <p>Another name for the InnoDB history list. Related to the innodb_max_purge_lag [1565] configuration option.</p> <p>See also history list.</p> <p>See also purge.</p> |
| purge thread | <p>A thread within the InnoDB process that is dedicated to performing the periodic purge operation. In MySQL 5.6 and higher, multiple purge threads are enabled by the innodb_purge_threads [1567] configuration option.</p> <p>See also purge.</p> <p>See also thread.</p> |
| Q | |
| query | <p>In SQL, an operation that reads information from one or more tables. Depending on the organization of data and the parameters of the query, the lookup might be optimized by consulting an index. If multiple tables are involved, the query is known as a join.</p> |

For historical reasons, sometimes discussions of internal processing for statements use “query” in a broader sense, including other types of MySQL statements such as **DDL** and **DML** statements.

See also [DDL](#).

See also [DML](#).

See also [index](#).

See also [join](#).

See also [SQL](#).

See also [table](#).

query execution plan

The set of decisions made by the optimizer about how to perform a **query** most efficiently, including which **index** or indexes to use, and the order in which to **join** tables. **Plan stability** involves the same choices being made consistently for a given query.

See also [index](#).

See also [join](#).

See also [plan stability](#).

See also [query](#).

query log

See [general query log](#).

quiesce

To reduce the amount of database activity, often in preparation for an operation such as an [ALTER TABLE](#), a **backup**, or a **shutdown**. Might or might not involve doing as much **flushing** as possible, so that **InnoDB** does not continue doing background I/O.

In MySQL 5.6 and higher, the syntax [FLUSH TABLES ... FOR EXPORT](#) writes some data to disk for **InnoDB** tables that make it simpler to back up those tables by copying the data files.

See also [backup](#).

See also [flush](#).

See also [InnoDB](#).

See also [shutdown](#).

R

RAID

Acronym for “Redundant Array of Inexpensive Drives”. Spreading I/O operations across multiple drives enables greater **concurrency** at the hardware level, and improves the efficiency of low-level write operations that otherwise would be performed in sequence.

See also [concurrency](#).

random dive

A technique for quickly estimating the number of different values in a column (the column's cardinality). InnoDB samples pages at random from the index and uses that data to estimate the number of different values. This operation occurs when each table is first opened.

Originally, the number of sampled pages was fixed at 8; now, it is determined by the setting of the [innodb_stats_sample_pages \[1571\]](#) parameter.

The way the random pages are picked depends on the setting of the [innodb_use_legacy_cardinality_algorithm](#) parameter. The default setting (OFF) has better randomness than in older releases.

See also [cardinality](#).

raw backup

The initial set of backup files produced by the **MySQL Enterprise Backup** product, before the changes reflected in the **binary log** and any **incremental backups** are applied. At this stage, the files are not ready to **restore**. After these changes are applied, the files are known as a **prepared backup**.

See also [binary log](#).

See also [hot backup](#).

See also [ibbackup_logfile](#).

See also [incremental backup](#).

See also [MySQL Enterprise Backup](#).

See also [prepared backup](#).

See also [restore](#).

READ COMMITTED

An **isolation level** that uses a **locking** strategy that relaxes some of the protection between **transactions**, in the interest of performance. Transactions cannot see uncommitted data from other transactions, but they can see data that is committed by another transaction after the current transaction started. Thus, a transaction never sees any bad data, but the data that it does see may depend to some extent on the timing of other transactions.

When a transaction with this isolation level performs [UPDATE ... WHERE](#) or [DELETE ... WHERE](#) operations, other transactions might have to wait. The transaction can perform [SELECT ... FOR UPDATE](#), and [LOCK IN SHARE MODE](#) operations without making other transactions wait.

See also [ACID](#).

See also [isolation level](#).

See also [locking](#).

See also [REPEATABLE READ](#).

See also [SERIALIZABLE](#).

| | |
|------------------|--|
| | See also transaction . |
| READ UNCOMMITTED | <p>The isolation level that provides the least amount of protection between transactions. Queries employ a locking strategy that allows them to proceed in situations where they would normally wait for another transaction. However, this extra performance comes at the cost of less reliable results, including data that has been changed by other transactions and not committed yet (known as dirty read). Use this isolation level only with great caution, and be aware that the results might not be consistent or reproducible, depending on what other transactions are doing at the same time. Typically, transactions with this isolation level do only queries, not insert, update, or delete operations.</p> <p>See also ACID.</p> <p>See also dirty read.</p> <p>See also isolation level.</p> <p>See also locking.</p> <p>See also transaction.</p> |
| read view | <p>An internal snapshot used by the MVCC mechanism of InnoDB. Certain transactions, depending on their isolation level, see the data values as they were at the time the transaction (or in some cases, the statement) started. Isolation levels that use a read view are REPEATABLE READ, READ COMMITTED, and READ UNCOMMITTED.</p> <p>See also isolation level.</p> <p>See also MVCC.</p> <p>See also READ COMMITTED.</p> <p>See also READ UNCOMMITTED.</p> <p>See also REPEATABLE READ.</p> <p>See also transaction.</p> |
| read-ahead | <p>A type of I/O request that prefetches a group of pages (an entire extent) into the buffer pool asynchronously, in anticipation that these pages will be needed soon. The linear read-ahead technique prefetches all the pages of one extent based on access patterns for pages in the preceding extent, and is part of all MySQL versions starting with the InnoDB Plugin for MySQL 5.1. The random read-ahead technique prefetches all the pages for an extent once a certain number of pages from the same extent are in the buffer pool. Random read-ahead is not part of MySQL 5.5, but is re-introduced in MySQL 5.6 under the control of the <code>innodb_random_read_ahead</code> configuration option.</p> <p>See also buffer pool.</p> <p>See also extent.</p> |

| | |
|-----------------------|---|
| | See also page . |
| read-only transaction | <p>A type of transaction that can be optimized for InnoDB tables by eliminating some of the bookkeeping involved with creating a read view for each transaction. Can only perform non-locking read queries. It can be started explicitly with the syntax <code>START TRANSACTION READ ONLY</code>, or automatically under certain conditions. See Optimizations for Read-Only Transactions for details.</p> <p>See also non-locking read.</p> <p>See also read view.</p> <p>See also transaction.</p> |
| record lock | <p>A lock on an index record. For example, <code>SELECT c1 FOR UPDATE FROM t WHERE c1 = 10;</code> prevents any other transaction from inserting, updating, or deleting rows where the value of <code>t.c1</code> is 10. Contrast with gap lock and next-key lock.</p> <p>See also gap lock.</p> <p>See also lock.</p> <p>See also next-key lock.</p> |
| redo | <p>The data, in units of records, recorded in the redo log when DML statements make changes to InnoDB tables. It is used during crash recovery to correct data written by incomplete transactions. The ever-increasing LSN value represents the cumulative amount of redo data that has passed through the redo log.</p> <p>See also crash recovery.</p> <p>See also DML.</p> <p>See also LSN.</p> <p>See also redo log.</p> <p>See also transaction.</p> |
| redo log | <p>A disk-based data structure used during crash recovery, to correct data written by incomplete transactions. During normal operation, it encodes requests to change InnoDB table data, which result from SQL statements or low-level API calls through NoSQL interfaces. Modifications that did not finish updating the data files before an unexpected shutdown are replayed automatically.</p> <p>The redo log is physically represented as a set of files, typically named ib_logfile0 and ib_logfile1. The data in the redo log is encoded in terms of records affected; this data is collectively referred to as redo. The passage of data through the redo logs is represented by the ever-increasing LSN value. The 4GB limit on maximum size for the redo log is raised in MySQL 5.6.</p> |

The disk layout of the redo log is influenced by the configuration options [innodb_log_file_size \[1564\]](#), [innodb_log_group_home_dir \[1564\]](#), and (rarely) [innodb_log_files_in_group \[1564\]](#). The performance of redo log operations is also affected by the **log buffer**, which is controlled by the [innodb_log_buffer_size \[1563\]](#) configuration option.

See also [crash recovery](#).

See also [data files](#).

See also [ib_logfile](#).

See also [log buffer](#).

See also [LSN](#).

See also [redo](#).

See also [shutdown](#).

See also [transaction](#).

redundant row format

The oldest InnoDB row format, available for tables using the **Antelope file format**. Prior to MySQL 5.0.3, it was the only row format available in InnoDB. In MySQL 5.0.3 and later, the default is **compact row format**. You can still specify redundant row format for compatibility with older InnoDB tables.

See also [Antelope](#).

See also [compact row format](#).

See also [file format](#).

See also [row format](#).

referential integrity

The technique of maintaining data always in a consistent format, part of the **ACID** philosophy. In particular, data in different tables is kept consistent through the use of **foreign key constraints**, which can prevent changes from happening or automatically propagate those changes to all related tables. Related mechanisms include the **unique constraint**, which prevents duplicate values from being inserted by mistake, and the **NOT NULL constraint**, which prevents blank values from being inserted by mistake.

See also [ACID](#).

See also [FOREIGN KEY constraint](#).

See also [NOT NULL constraint](#).

See also [unique constraint](#).

relational

An important aspect of modern database systems. The database server encodes and enforces relationships such as one-to-one, one-to-many, many-to-one, and uniqueness. For example, a person might have zero, one,

or many phone numbers in an address database; a single phone number might be associated with several family members. In a financial database, a person might be required to have exactly one taxpayer ID, and any taxpayer ID could only be associated with one person.

The database server can use these relationships to prevent bad data from being inserted, and to find efficient ways to look up information. For example, if a value is declared to be unique, the server can stop searching as soon as the first match is found, and it can reject attempts to insert a second copy of the same value.

At the database level, these relationships are expressed through SQL features such as **columns** within a table, unique and **NOT NULL constraints**, **foreign keys**, and different kinds of join operations. Complex relationships typically involve data split between more than one table. Often, the data is **normalized**, so that duplicate values in one-to-many relationships are stored only once.

In a mathematical context, the relations within a database are derived from set theory. For example, the **OR** and **AND** operators of a **WHERE** clause represent the notions of union and intersection.

See also [ACID](#).

See also [constraint](#).

See also [foreign key](#).

See also [normalized](#).

relevance

In the **full-text search** feature, a number signifying the similarity between the search string and the data in the **FULLTEXT index**. For example, when you search for a single word, that word is typically more relevant for a row where it occurs several times in the text than a row where it appears only once.

See also [full-text search](#).

See also [FULLTEXT index](#).

REPEATABLE READ

The default **isolation level** for InnoDB. It prevents any rows that are queried from being changed by other transactions, thus blocking **non-repeatable reads** but not **phantom** reads. It uses a moderately strict **locking** strategy so that all queries within a transaction see data from the same snapshot, that is, the data as it was at the time the transaction started.

When a transaction with this isolation level performs **UPDATE ... WHERE**, **DELETE ... WHERE**, **SELECT ... FOR UPDATE**, and **LOCK IN SHARE MODE** operations, other transactions might have to wait.

See also [ACID](#).

See also [consistent read](#).

See also [isolation level](#).

See also [locking](#).

| | |
|------------------|---|
| | See also phantom . |
| | See also SERIALIZABLE . |
| | See also transaction . |
| replication | <p>The practice of sending changes from a master database, to one or more slave databases, so that all databases have the same data. This technique has a wide range of uses, such as load-balancing for better scalability, disaster recovery, and testing software upgrades and configuration changes. The changes can be sent between the database by methods called row-based replication and statement-based replication.</p> <p>See also row-based replication.</p> <p>See also statement-based replication.</p> |
| restore | <p>The process of putting a set of backup files from the MySQL Enterprise Backup product in place for use by MySQL. This operation can be performed to fix a corrupted database, to return to some earlier point in time, or (in a replication context) to set up a new slave database. In the MySQL Enterprise Backup product, this operation is performed by the copy-back option of the mysqlbackup command.</p> <p>See also hot backup.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also mysqlbackup command.</p> <p>See also prepared backup.</p> <p>See also replication.</p> |
| rollback | <p>A SQL statement that ends a transaction, undoing any changes made by the transaction. It is the opposite of commit, which makes permanent any changes made in the transaction.</p> <p>By default, MySQL uses the autocommit setting, which automatically issues a commit following each SQL statement. You must change this setting before you can use the rollback technique.</p> <p>See also ACID.</p> <p>See also commit.</p> <p>See also transaction.</p> |
| rollback segment | <p>The storage area containing the undo log, part of the system tablespace.</p> <p>See also system tablespace.</p> <p>See also undo log.</p> |
| row | <p>The logical data structure defined by a set of columns. A set of rows makes up a table. Within InnoDB data files, each page can contain one or more rows.</p> |

Although InnoDB uses the term **row format** for consistency with MySQL syntax, the row format is a property of each table and applies to all rows in that table.

See also [column](#).

See also [data files](#).

See also [page](#).

See also [row format](#).

See also [table](#).

row format

The disk storage format for a **row** from an InnoDB **table**. As InnoDB gains new capabilities such as compression, new row formats are introduced to support the resulting improvements in storage efficiency and performance.

Each table has its own row format, specified through the `ROW_FORMAT` option. To see the row format for each InnoDB table, issue the command `SHOW TABLE STATUS`. Because all the tables in the system tablespace share the same row format, to take advantage of other row formats typically requires setting the `innodb_file_per_table [1555]` option, so that each table is stored in a separate tablespace.

See also [compact row format](#).

See also [compressed row format](#).

See also [dynamic row format](#).

See also [fixed row format](#).

See also [redundant row format](#).

See also [row](#).

See also [table](#).

row lock

A **lock** that prevents a row from being accessed in an incompatible way by another **transaction**. Other rows in the same table can be freely written to by other transactions. This is the type of **locking** done by **DML** operations on **InnoDB** tables.

Contrast with **table locks** used by MyISAM, or during **DDL** operations on InnoDB tables that cannot be done with **online DDL**; those locks block concurrent access to the table.

See also [DDL](#).

See also [DML](#).

See also [InnoDB](#).

See also [lock](#).

See also [locking](#).

| | |
|-----------------------|--|
| | See also online DDL . |
| | See also table lock . |
| | See also transaction . |
| row-based replication | <p>A form of replication where events are propagated from the master server specifying how to change individual rows on the slave server. It is safe to use for all settings of the <code>innodb_autoinc_lock_mode</code> [1548] option.</p> <p>See also auto-increment locking.</p> <p>See also innodb_autoinc_lock_mode.</p> <p>See also replication.</p> <p>See also statement-based replication.</p> |
| row-level locking | <p>The locking mechanism used for InnoDB tables, relying on row locks rather than table locks. Multiple transactions can modify the same table concurrently. Only if two transactions try to modify the same row does one of the transactions wait for the other to complete (and release its row locks).</p> <p>See also InnoDB.</p> <p>See also locking.</p> <p>See also row lock.</p> <p>See also table lock.</p> <p>See also transaction.</p> |
| rw-lock | <p>The low-level object that InnoDB uses to represent and enforce shared-access locks to internal in-memory data structures. Once the lock is acquired, any other process, thread, and so on can read the data structure, but no one else can write to it. Contrast with mutexes, which enforce exclusive access. Mutexes and rw-locks are known collectively as latches.</p> <p>See also latch.</p> <p>See also lock.</p> <p>See also mutex.</p> <p>See also Performance Schema.</p> |
| S | |
| savepoint | <p>Savepoints help to implement nested transactions. They can be used to provide scope to operations on tables that are part of a larger transaction. For example, scheduling a trip in a reservation system might involve booking several different flights; if a desired flight is unavailable, you might roll back the changes involved in booking that one leg, without rolling back the earlier flights that were successfully booked.</p> |

| | |
|-----------------|--|
| | See also rollback . |
| | See also transaction . |
| scalability | <p>The ability to add more work and issue more simultaneous requests to a system, without a sudden drop in performance due to exceeding the limits of system capacity. Software architecture, hardware configuration, application coding, and type of workload all play a part in scalability. When the system reaches its maximum capacity, popular techniques for increasing scalability are scale up (increasing the capacity of existing hardware or software) and scale out (adding new hardware or more instances of server software).</p> |
| schema | <p>Conceptually, a schema is a set of interrelated database objects, such as tables, table columns, data types of the columns, indexes, foreign keys, and so on. These objects are connected through SQL syntax, because the columns make up the tables, the foreign keys refer to tables and columns, and so on. Ideally, they are also connected logically, working together as part of a unified application or flexible framework. For example, the information_schema and performance_schema databases use “schema” in their names to emphasize the close relationships between the tables and columns they contain.</p> <p>In MySQL, physically, a schema is synonymous with a database. You can substitute the keyword SCHEMA instead of DATABASE in MySQL SQL syntax, for example using CREATE SCHEMA instead of CREATE DATABASE.</p> <p>Some other database products draw a distinction. For example, in the Oracle Database product, a schema represents only a part of a database: the tables and other objects owned by a single user.</p> <p>See also database.</p> <p>See also ib-file set.</p> <p>See also INFORMATION_SCHEMA.</p> <p>See also Performance Schema.</p> |
| search index | <p>In MySQL, full-text search queries use a special kind of index, the FULLTEXT index. In MySQL 5.6.4 and up, InnoDB and MyISAM tables both support FULLTEXT indexes; formerly, these indexes were only available for MyISAM tables.</p> <p>See also full-text search.</p> <p>See also FULLTEXT index.</p> |
| secondary index | <p>A type of InnoDB index that represents a subset of table columns. An InnoDB table can have zero, one, or many secondary indexes. (Contrast with the clustered index, which is required for each InnoDB table, and stores the data for all the table columns.)</p> |

A secondary index can be used to satisfy queries that only require values from the indexed columns. For more complex queries, it can be used to identify the relevant rows in the table, which are then retrieved through lookups using the clustered index.

Creating and dropping secondary indexes has traditionally involved significant overhead from copying all the data in the InnoDB table. The **fast index creation** feature of the InnoDB Plugin makes both `CREATE INDEX` and `DROP INDEX` statements much faster for InnoDB secondary indexes.

See also [clustered index](#).

See also [Fast Index Creation](#).

See also [index](#).

segment

A division within an InnoDB **tablespace**. If a tablespace is analogous to a directory, the segments are analogous to files within that directory. A segment can grow. New segments can be created.

For example, within a **file-per-table** tablespace, the table data is in one segment and each associated index is in its own segment. The **system tablespace** contains many different segments, because it can hold many tables and their associated indexes. The system tablespace also includes up to 128 **rollback segments** making up the **undo log**.

Segments grow and shrink as data is inserted and deleted. When a segment needs more room, it is extended by one **extent** (1 megabyte) at a time. Similarly, a segment releases one extent's worth of space when all the data in that extent is no longer needed.

See also [extent](#).

See also [file-per-table](#).

See also [rollback segment](#).

See also [system tablespace](#).

See also [tablespace](#).

See also [undo log](#).

selectivity

A property of data distribution, the number of distinct values in a column (its **cardinality**) divided by the number of records in the table. High selectivity means that the column values are relatively unique, and can be retrieved efficiently through an index. If you (or the query optimizer) can predict that a test in a `WHERE` clause only matches a small number (or proportion) of rows in a table, the overall **query** tends to be efficient if it evaluates that test first, using an index.

See also [cardinality](#).

See also [query](#).

semi-consistent read

A type of read operation used for [UPDATE](#) statements, that is a combination of **read committed** and **consistent read**. When an [UPDATE](#) statement examines a row that is already locked, InnoDB returns the latest committed version to MySQL so that MySQL can determine whether the row matches the [WHERE](#) condition of the [UPDATE](#). If the row matches (must be updated), MySQL reads the row again, and this time InnoDB either locks it or waits for a lock on it. This type of read operation can only happen when the transaction has the read committed **isolation level**, or when the [innodb_locks_unsafe_for_binlog \[1561\]](#) option is enabled.

See also [consistent read](#).

See also [isolation level](#).

See also [READ COMMITTED](#).

SERIALIZABLE

The **isolation level** that uses the most conservative locking strategy, to prevent any other transactions from inserting or changing data that was read by this transaction, until it is finished. This way, the same query can be run over and over within a transaction, and be certain to retrieve the same set of results each time. Any attempt to change data that was committed by another transaction since the start of the current transaction, cause the current transaction to wait.

This is the default isolation level specified by the SQL standard. In practice, this degree of strictness is rarely needed, so the default isolation level for InnoDB is the next most strict, **repeatable read**.

See also [ACID](#).

See also [consistent read](#).

See also [isolation level](#).

See also [locking](#).

See also [REPEATABLE READ](#).

See also [transaction](#).

server

A type of program that runs continuously, waiting to receive and act upon requests from another program (the client). Because often an entire computer is dedicated to running one or more server programs (such as a database server, a web server, an application server, or some combination of these), the term **server** can also refer to the computer that runs the server software.

See also [client](#).

See also [mysqld](#).

shared lock

A kind of **lock** that allows other **transactions** to read the locked object, and to also acquire other shared locks on it, but not to write to it. The opposite of **exclusive lock**.

See also [exclusive lock](#).

| | |
|-------------------|--|
| | See also lock . |
| | See also transaction . |
| shared tablespace | Another way of referring to the system tablespace . |
| | See also system tablespace . |
| sharp checkpoint | The process of flushing to disk all dirty buffer pool pages whose redo entries are contained in certain portion of the redo log . Occurs before InnoDB reuses a portion of a log file; the log files are used in a circular fashion. Typically occurs with write-intensive workloads . |
| | See also dirty page . |
| | See also flush . |
| | See also redo log . |
| | See also workload . |
| shutdown | The process of stopping the MySQL server. By default, this process does cleanup operations for InnoDB tables, so it can slow to shut down, but fast to start up later. If you skip the cleanup operations, it is fast to shut down but must do the cleanup during the next restart. |
| | The shutdown mode is controlled by the innodb_fast_shutdown [1553] option. |
| | See also fast shutdown . |
| | See also InnoDB . |
| | See also slow shutdown . |
| | See also startup . |
| slave server | Frequently shortened to “slave”. A database server machine in a replication scenario that receives changes from another server (the master) and applies those same changes. Thus it maintains the same contents as the master, although it might lag somewhat behind. |
| | In MySQL, slave servers are commonly used in disaster recovery, to take the place of a master servers that fails. They are also commonly used for testing software upgrades and new settings, to ensure that database configuration changes do not cause problems with performance or reliability. |
| | Slave servers typically have high workloads, because they process all the DML (write) operations relayed from the master, as well as user queries. To ensure that slave servers can apply changes from the master fast enough, they frequently have fast I/O devices and sufficient CPU and memory to run multiple database instances on the same slave server. For example, the master server might use hard drive storage while the slave servers use SSDs . |

| | |
|----------------|--|
| | See also DML . |
| | See also replication . |
| | See also server . |
| | See also SSD . |
| slow query log | <p>A type of log used for performance tuning of SQL statements processed by the MySQL server. The log information is stored in a file. You must enable this feature to use it. You control which categories of “slow” SQL statements are logged. For more information, see Section 5.2.5, “The Slow Query Log”.</p> <p>See also general query log.</p> <p>See also log.</p> |
| slow shutdown | <p>A type of shutdown that does additional InnoDB flushing operations before completing. Also known as a clean shutdown. Specified by the configuration parameter <code>innodb_fast_shutdown=0</code> [1553] or the command <code>SET GLOBAL innodb_fast_shutdown=0;</code>. Although the shutdown itself can take longer, that time will be saved on the subsequent startup.</p> <p>See also clean shutdown.</p> <p>See also fast shutdown.</p> <p>See also shutdown.</p> |
| snapshot | <p>A representation of data at a particular time, which remains the same even as changes are committed by other transactions. Used by certain isolation levels to allow consistent reads.</p> <p>See also commit.</p> <p>See also consistent read.</p> <p>See also isolation level.</p> <p>See also transaction.</p> |
| space ID | <p>An identifier used to uniquely identify an InnoDB tablespace within a MySQL instance. The space ID for the system tablespace is always zero; this same ID applies to all tables within the system tablespace. Each tablespace file created in file-per-table mode also has its own space ID.</p> <p>Prior to MySQL 5.6, this hardcoded value presented difficulties in moving InnoDB tablespace files between MySQL instances. Starting in MySQL 5.6, you can copy tablespace files between instances by using the transportable tablespace feature involving the statements <code>FLUSH TABLES ... FOR EXPORT</code>, <code>ALTER TABLE ... DISCARD TABLESPACE</code>, and <code>ALTER TABLE ... IMPORT TABLESPACE</code>. The information needed to adjust the space ID is conveyed in the .cfg file</p> |

which you copy along with the tablespace. See [Improved Tablespace Management](#) for details.

See also [.cfg file](#).

See also [file-per-table](#).

See also [.ibd file](#).

See also [system tablespace](#).

See also [tablespace](#).

See also [transportable tablespace](#).

spin A type of **wait** operation that continuously tests whether a resource becomes available. This technique is used for resources that are typically held only for brief periods, where it is more efficient to wait in a “busy loop” than to put the thread to sleep and perform a context switch. If the resource does not become available within a short time, the spin loop ceases and another wait technique is used.

See also [latch](#).

See also [lock](#).

See also [mutex](#).

See also [wait](#).

SQL The Structured Query Language that is standard for performing database operations. Often divided into the categories **DDL**, **DML**, and **queries**.

See also [DDL](#).

See also [DML](#).

See also [query](#).

SSD Acronym for “solid-state drive”. A type of storage device with different performance characteristics than a traditional hard disk drive (**HDD**): smaller storage capacity, faster for random reads, no moving parts, and with a number of considerations affecting write performance. Its performance characteristics can influence the throughput of a **disk-bound** workload.

See also [disk-bound](#).

See also [SSD](#).

startup The process of starting the MySQL server. Typically done by one of the programs listed in [Section 4.3, “MySQL Server and Server-Startup Programs”](#). The opposite of **shutdown**.

See also [shutdown](#).

statement-based replication

A form of **replication** where SQL statements are sent from the **master** server and replayed on the **slave** server. It requires some care with the setting for the `innodb_autoinc_lock_mode` [1548] option, to avoid potential timing problems with **auto-increment locking**.

See also [auto-increment locking](#).

See also [innodb_autoinc_lock_mode](#).

See also [replication](#).

See also [row-based replication](#).

statistics

Estimated values relating to each **InnoDB table** and **index**, used to construct an efficient **query execution plan**. The main values are the **cardinality** (number of distinct values) and the total number of table rows or index entries. The statistics for the table represent the data in its **primary key** index. The statistics for a **secondary index** represent the rows covered by that index.

The values are estimated rather than counted precisely because at any moment, different **transactions** can be inserting and deleting rows from the same table. To keep the values from being recalculated frequently, you can enable **persistent statistics**, where the values are stored in **InnoDB** system tables, and refreshed only when you issue an `ANALYZE TABLE` statement.

You can control how **NULL** values are treated when calculating statistics through the `innodb_stats_method` [1570] configuration option.

Other types of statistics are available for database objects and database activity through the **INFORMATION_SCHEMA** and **PERFORMANCE_SCHEMA** tables.

See also [cardinality](#).

See also [index](#).

See also [INFORMATION_SCHEMA](#).

See also [NULL](#).

See also [Performance Schema](#).

See also [persistent statistics](#).

See also [primary key](#).

See also [query execution plan](#).

See also [secondary index](#).

See also [table](#).

See also [transaction](#).

| | |
|----------------|--|
| stemming | <p>The ability to search for different variations of a word based on a common root word, such as singular and plural, or past, present, and future verb tense. This feature is currently supported in MyISAM full-text search feature but not in FULLTEXT indexes for InnoDB tables.</p> <p>See also full-text search.</p> <p>See also FULLTEXT index.</p> |
| stopword | <p>In a FULLTEXT index, a word that is considered common or trivial enough that it is omitted from the search index and ignored in search queries. Different configuration settings control stopwords processing for InnoDB and MyISAM tables. See Section 12.9.4, “Full-Text Stopwords” for details.</p> <p>See also FULLTEXT index.</p> <p>See also search index.</p> |
| storage engine | <p>A component of the MySQL database that performs the low-level work of storing, updating, and querying data. In MySQL 5.5 and higher, InnoDB is the default storage engine for new tables, superseding MyISAM. Different storage engines are designed with different tradeoffs between factors such as memory usage versus disk usage, read speed versus write speed, and speed versus robustness. Each storage engine manages specific tables, so we refer to InnoDB tables, MyISAM tables, and so on.</p> <p>The MySQL Enterprise Backup product is optimized for backing up InnoDB tables. It can also back up tables handled by MyISAM and other storage engines.</p> <p>See also InnoDB.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also table type.</p> |
| strict mode | <p>The general name for the setting controlled by the innodb_strict_mode [1572] option. Turning on this setting causes certain conditions that are normally treated as warnings, to be considered errors. For example, certain invalid combinations of options related to file format and row format, that normally produce a warning and continue with default values, now cause the CREATE TABLE operation to fail.</p> <p>MySQL also has something called strict mode.</p> <p>See also file format.</p> <p>See also innodb_strict_mode.</p> <p>See also row format.</p> |
| sublist | <p>Within the list structure that represents the buffer pool, pages that are relatively old and relatively new are represented by different portions of</p> |

| | |
|-------------------|---|
| | <p>the list. A set of parameters control the size of these portions and the dividing point between the new and old pages.</p> <p>See also buffer pool.</p> <p>See also eviction.</p> <p>See also list.</p> <p>See also LRU.</p> |
| supremum record | <p>A pseudo-record in an index, representing the gap above the largest value in that index. If a transaction has a statement such as <code>SELECT ... FOR UPDATE ... WHERE col > 10;</code>, and the largest value in the column is 20, it is a lock on the supremum record that prevents other transactions from inserting even larger values such as 50, 100, and so on.</p> <p>See also gap.</p> <p>See also infimum record.</p> <p>See also pseudo-record.</p> |
| surrogate key | <p>Synonym name for synthetic key.</p> <p>See also synthetic key.</p> |
| synthetic key | <p>A indexed column, typically a primary key, where the values are assigned arbitrarily. Often done using an auto-increment column. By treating the value as completely arbitrary, you can avoid overly restrictive rules and faulty application assumptions. For example, a numeric sequence representing employee numbers might have a gap if an employee was approved for hiring but never actually joined. Or employee number 100 might have a later hiring date than employee number 500, if they left the company and later rejoined. Numeric values also produce shorter values of predictable length. For example, storing numeric codes meaning “Road”, “Boulevard”, “Expressway”, and so on is more space-efficient than repeating those strings over and over.</p> <p>Also known as a surrogate key. Contrast with natural key.</p> <p>See also auto-increment.</p> <p>See also natural key.</p> <p>See also primary key.</p> <p>See also surrogate key.</p> |
| system tablespace | <p>A small set of data files (the ibdata files) containing the metadata for InnoDB-related objects (the data dictionary), and the storage areas for the undo log, the change buffer, and the doublewrite buffer. Depending on the setting of the <code>innodb_file_per_table</code> [1555], when tables are created, it might also contain table and index data for some or all InnoDB tables. The data and metadata in the system tablespace apply to all the databases in a MySQL instance.</p> |

Prior to MySQL 5.6.7, the default was to keep all InnoDB tables and indexes inside the system tablespace, often causing this file to become very large. Because the system tablespace never shrinks, storage problems could arise if large amounts of temporary data were loaded and then deleted. In MySQL 5.6.7 and higher, the default is **file-per-table** mode, where each table and its associated indexes are stored in a separate **.ibd file**. This new default makes it easier to use InnoDB features that rely on the **Barracuda** file format, such as table **compression** and the **DYNAMIC** row format.

In MySQL 5.6 and higher, setting a value for the [innodb_undo_tablespaces](#) option splits the **undo log** into one or more separate tablespace files. These files are still considered part of the system tablespace.

Keeping all table data in the system tablespace or in separate [.ibd](#) files has implications for storage management in general. The **MySQL Enterprise Backup** product might back up a small set of large files, or many smaller files. On systems with thousands of tables, the filesystem operations to process thousands of [.ibd](#) files can cause bottlenecks.

See also [Barracuda](#).

See also [change buffer](#).

See also [compression](#).

See also [data dictionary](#).

See also [database](#).

See also [doublewrite buffer](#).

See also [dynamic row format](#).

See also [file-per-table](#).

See also [.ibd file](#).

See also [ibdata file](#).

See also [innodb_file_per_table](#).

See also [instance](#).

See also [MySQL Enterprise Backup](#).

See also [tablespace](#).

See also [undo log](#).

T

table

Each MySQL table is associated with a particular **storage engine**. **InnoDB** tables have particular **physical** and **logical** characteristics that

affect performance, **scalability**, **backup**, administration, and application development.

In terms of file storage, each InnoDB table is either part of the single big InnoDB **system tablespace**, or in a separate **.ibd** file if the table is created in **file-per-table** mode. The **.ibd** file holds data for the table and all its **indexes**, and is known as a **tablespace**.

InnoDB tables created in file-per-table mode can use the **Barracuda** file format. Barracuda tables can use the **DYNAMIC row format** or the **COMPRESSED row format**. These relatively new settings enable a number of InnoDB features, such as **compression**, **fast index creation**, and **off-page columns**

For backward compatibility with MySQL 5.1 and earlier, InnoDB tables inside the system tablespace must use the **Antelope** file format, which supports the **compact row format** and the **redundant row format**.

The **rows** of an InnoDB table are organized into an index structure known as the **clustered index**, with entries sorted based on the **primary key** columns of the table. Data access is optimized for queries that filter and sort on the primary key columns, and each index contains a copy of the associated primary key columns for each entry. Modifying values for any of the primary key columns is an expensive operation. Thus an important aspect of InnoDB table design is choosing a primary key with columns that are used in the most important queries, and keeping the primary key short, with rarely changing values.

See also [Antelope](#).

See also [backup](#).

See also [Barracuda](#).

See also [clustered index](#).

See also [compact row format](#).

See also [compressed row format](#).

See also [compression](#).

See also [dynamic row format](#).

See also [Fast Index Creation](#).

See also [file-per-table](#).

See also [.ibd file](#).

See also [index](#).

See also [off-page column](#).

See also [primary key](#).

See also [redundant row format](#).

| | |
|------------------|--|
| | See also row . |
| | See also system tablespace . |
| | See also tablespace . |
| table lock | <p>A lock that prevents any other transaction from accessing a table. InnoDB makes considerable effort to make such locks unnecessary, by using techniques such as online DDL, row locks and consistent reads for processing DML statements and queries. You can create such a lock through SQL using the <code>LOCK TABLE</code> statement; one of the steps in migrating from other database systems or MySQL storage engines is to remove such statements wherever practical.</p> <p>See also consistent read.</p> <p>See also DML.</p> <p>See also lock.</p> <p>See also locking.</p> <p>See also online DDL.</p> <p>See also query.</p> <p>See also row lock.</p> <p>See also table.</p> <p>See also transaction.</p> |
| table scan | See full table scan . |
| table statistics | See statistics . |
| table type | <p>Obsolete synonym for storage engine. We refer to InnoDB tables, MyISAM tables, and so on.</p> <p>See also InnoDB.</p> <p>See also storage engine.</p> |
| tablespace | <p>A data file that can hold data for one or more InnoDB tables and associated indexes. The system tablespace contains the tables that make up the data dictionary, and prior to MySQL 5.6 holds all the other InnoDB tables by default. Turning on the innodb_file_per_table [1555] option, the default in MySQL 5.6 and higher, allows newly created tables to each have their own tablespace, with a separate data file for each table.</p> <p>Using multiple tablespaces, by turning on the innodb_file_per_table [1555] option, is vital to using many MySQL features such as table compression and transportable tablespaces, and managing disk usage. See Section 14.3.3, “Using Per-Table Tablespaces” and Improved Tablespace Management for details.</p> |

Tablespaces created by the built-in InnoDB storage engine are upward compatible with the InnoDB Plugin. Tablespaces created by the InnoDB Plugin are downward compatible with the built-in InnoDB storage engine, if they use the **Antelope** file format.

MySQL Cluster also groups its tables into tablespaces. See [Section 17.5.12.1, “MySQL Cluster Disk Data Objects”](#) for details.

See also [Antelope](#).

See also [Barracuda](#).

See also [compressed row format](#).

See also [data dictionary](#).

See also [data files](#).

See also [file-per-table](#).

See also [index](#).

See also [innodb_file_per_table](#).

See also [system tablespace](#).

See also [table](#).

tablespace dictionary

A representation of the **data dictionary** metadata for a table, within the InnoDB **tablespace**. This metadata can be checked against the **.frm file** for consistency when the table is opened, to diagnose errors resulting from out-of-date **.frm** files. This information is present for InnoDB tables that are part of the **system tablespace**, as well as for tables that have their own **.ibd file** because of the **file-per-table** option.

See also [data dictionary](#).

See also [file-per-table](#).

See also [.frm file](#).

See also [.ibd file](#).

See also [system tablespace](#).

See also [tablespace](#).

temporary table

A table whose data does not need to be truly permanent. For example, temporary tables might be used as storage areas for intermediate results in complicated calculations or transformations; this intermediate data would not need to be recovered after a crash. Database products can take various shortcuts to improve the performance of operations on temporary tables, by being less scrupulous about writing data to disk and other measures to protect the data across restarts.

| | |
|-----------------|--|
| | <p>Sometimes, the data itself is removed automatically at a set time, such as when the transaction ends or when the session ends. With some database products, the table itself is removed automatically too.</p> <p>See also table.</p> |
| text collection | <p>The set of columns included in a FULLTEXT index.</p> <p>See also FULLTEXT index.</p> |
| thread | <p>A unit of processing that is typically more lightweight than a process, allowing for greater concurrency.</p> <p>See also concurrency.</p> <p>See also master thread.</p> <p>See also process.</p> <p>See also Pthreads.</p> |
| torn page | <p>An error condition that can occur due to a combination of I/O device configuration and hardware failure. If data is written out in chunks smaller than the InnoDB page size (by default, 16KB), a hardware failure while writing could result in only part of a page being stored to disk. The InnoDB doublewrite buffer guards against this possibility.</p> <p>See also doublewrite buffer.</p> |
| TPS | <p>Acronym for “transactions per second”, a unit of measurement sometimes used in benchmarks. Its value depends on the workload represented by a particular benchmark test, combined with factors that you control such as the hardware capacity and database configuration.</p> <p>See also transaction.</p> <p>See also workload.</p> |
| transaction | <p>Transactions are atomic units of work that can be committed or rolled back. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.</p> <p>Database transactions, as implemented by InnoDB, have properties that are collectively known by the acronym ACID, for atomicity, consistency, isolation, and durability.</p> <p>See also ACID.</p> <p>See also commit.</p> <p>See also isolation level.</p> <p>See also lock.</p> <p>See also rollback.</p> |

| | |
|--------------------------|--|
| transaction ID | <p>An internal field associated with each row. This field is physically changed by INSERT, UPDATE, and DELETE operations to record which transaction has locked the row.</p> <p>See also implicit row lock.</p> |
| transportable tablespace | <p>A feature that allows a tablespace to be moved from one instance to another. Traditionally, this has not been possible for InnoDB tablespaces because all table data was part of the system tablespace. In MySQL 5.6 and higher, the <code>FLUSH TABLES ... FOR EXPORT</code> syntax prepares an InnoDB table for copying to another server; running <code>ALTER TABLE ... DISCARD TABLESPACE</code> and <code>ALTER TABLE ... IMPORT TABLESPACE</code> on the other server brings the copied data file into the other instance. A separate <code>.cfg</code> file, copied along with the .ibd file, is used to update the table metadata (for example the space ID) as the tablespace is imported. See Improved Tablespace Management for usage information.</p> <p>See also .ibd file.</p> <p>See also space ID.</p> <p>See also system tablespace.</p> <p>See also tablespace.</p> |
| .TRG file | <p>A file containing trigger parameters. Files with this extension are always included in backups produced by the <code>mysqlbackup</code> command of the MySQL Enterprise Backup product.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also mysqlbackup command.</p> <p>See also .TRN file.</p> |
| .TRN file | <p>A file containing trigger namespace information. Files with this extension are always included in backups produced by the <code>mysqlbackup</code> command of the MySQL Enterprise Backup product.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also mysqlbackup command.</p> <p>See also .TRG file.</p> |
| troubleshooting | <p>Resources for troubleshooting InnoDB reliability and performance issues include: the Information Schema tables.</p> |
| truncate | <p>A DDL operation that removes the entire contents of a table, while leaving the table and related indexes intact. Contrast with drop. Although conceptually it has the same result as a <code>DELETE</code> statement with no <code>WHERE</code> clause, it operates differently behind the scenes: InnoDB creates a new empty table, drops the old table, then renames the new table to take the place of the old one. Because this is a DDL operation, it cannot be rolled back.</p> |

If the table being truncated contains foreign keys that reference another table, the truncation operation uses a slower method of operation, deleting one row at a time so that corresponding rows in the referenced table can be deleted as needed by any [ON DELETE CASCADE](#) clause. (MySQL 5.5 and higher do not allow this slower form of truncate, and return an error instead if foreign keys are involved. In this case, use a [DELETE](#) statement instead.

See also [DDL](#).

See also [drop](#).

See also [foreign key](#).

See also [rollback](#).

tuple

A technical term designating an ordered set of elements. It is an abstract notion, used in formal discussions of database theory. In the database field, tuples are usually represented by the columns of a table row. They could also be represented by the result sets of queries, for example, queries that retrieved only some columns of a table, or columns from joined tables.

See also [cursor](#).

two-phase commit

An operation that is part of a distributed **transaction**, under the **XA** specification. (Sometimes abbreviated as 2PC.) When multiple databases participate in the transaction, either all databases **commit** the changes, or all databases **roll back** the changes.

See also [commit](#).

See also [rollback](#).

See also [transaction](#).

See also [XA](#).

U

undo

Data that is maintained throughout the life of a **transaction**, recording all changes so that they can be undone in case of a **rollback** operation. It is stored in the **undo log**, also known as the **rollback segment**, either within the **system tablespace** or in separate **undo tablespaces**.

See also [rollback](#).

See also [rollback segment](#).

See also [system tablespace](#).

See also [transaction](#).

See also [undo log](#).

See also [undo tablespace](#).

| | |
|-------------------|---|
| undo buffer | See undo log . |
| undo log | <p>A storage area that holds copies of data modified by active transactions. If another transaction needs to see the original data (as part of a consistent read operation), the unmodified data is retrieved from this storage area.</p> <p>By default, this area is physically part of the system tablespace. In MySQL 5.6 and higher, you can use the innodb_undo_tablespaces and innodb_undo_directory configuration options to split it into one or more separate tablespace files, the undo tablespaces, optionally stored on another storage device such as an SSD.</p> <p>The undo log is split into separate portions, the insert undo buffer and the update undo buffer. Collectively, these parts are also known as the rollback segment, a familiar term for Oracle DBAs.</p> <p>See also consistent read.</p> <p>See also rollback segment.</p> <p>See also SSD.</p> <p>See also system tablespace.</p> <p>See also transaction.</p> <p>See also undo tablespace.</p> |
| undo tablespace | <p>One of a set of files containing the undo log, when the undo log is separated from the system tablespace by the innodb_undo_tablespaces and innodb_undo_directory configuration options. Only applies to MySQL 5.6 and higher.</p> <p>See also system tablespace.</p> <p>See also undo log.</p> |
| unique constraint | <p>A kind of constraint that asserts that a column cannot contain any duplicate values. In terms of relational algebra, it is used to specify 1-to-1 relationships. For efficiency in checking whether a value can be inserted (that is, the value does not already exist in the column), a unique constraint is supported by an underlying unique index.</p> <p>See also constraint.</p> <p>See also relational.</p> <p>See also unique index.</p> |
| unique index | <p>An index on a column or set of columns that have a unique constraint. Because the index is known not to contain any duplicate values, certain kinds of lookups and count operations are more efficient than in the normal kind of index. Most of the lookups against this type of index are simply to determine if a certain value exists or not. The number of values in the index is the same as the number of rows in the table, or at least the number of rows with non-null values for the associated columns.</p> |

The **insert buffering** optimization does not apply to unique indexes. As a workaround, you can temporarily set `unique_checks=0` while doing a bulk data load into an InnoDB table.

See also [cardinality](#).

See also [insert buffering](#).

See also [unique constraint](#).

See also [unique key](#).

unique key

The set of columns (one or more) comprising a **unique index**. When you can define a `WHERE` condition that matches exactly one row, and the query can use an associated unique index, the lookup and error handling can be performed very efficiently.

See also [cardinality](#).

See also [unique constraint](#).

See also [unique index](#).

V

victim The transaction that is automatically chosen to be rolled back when a **deadlock** is detected. InnoDB rolls back the transaction that has updated the fewest rows.

See also [deadlock](#).

See also [deadlock detection](#).

See also [innodb_lock_wait_timeout](#).

W

wait When an operation, such as acquiring a **lock**, **mutex**, or **latch**, cannot be completed immediately, InnoDB pauses and tries again. The mechanism for pausing is elaborate enough that this operation has its own name, the **wait**. Individual threads are paused using a combination of internal InnoDB scheduling, operating system `wait()` calls, and short-duration **spin** loops.

On systems with heavy load and many transactions, you might use the output from the `SHOW INNODB STATUS` command to determine whether threads are spending too much time waiting, and if so, how you can improve **concurrency**.

See also [concurrency](#).

See also [latch](#).

See also [lock](#).

See also [mutex](#).

See also [spin](#).

| | |
|-----------------|--|
| warm backup | <p>A backup taken while the database is running, but that restricts some database operations during the backup process. For example, tables might become read-only. For busy applications and web sites, you might prefer a hot backup.</p> <p>See also backup.</p> <p>See also cold backup.</p> <p>See also hot backup.</p> |
| warm up | <p>To run a system under a typical workload for some time after startup, so that the buffer pool and other memory regions are filled as they would be under normal conditions.</p> <p>This process happens naturally over time when a MySQL server is restarted or subjected to a new workload. Starting in MySQL 5.6, you can speed up the warmup process by setting the configuration variables innodb_buffer_pool_dump_at_shutdown=ON and innodb_buffer_pool_load_at_startup=ON, to bring the contents of the buffer pool back into memory after a restart. Typically, you run a workload for some time to warm up the buffer pool before running performance tests, to ensure consistent results across multiple runs; otherwise, performance might be artificially low during the first run.</p> <p>See also buffer pool.</p> <p>See also workload.</p> |
| Windows | <p>The built-in InnoDB storage engine and the InnoDB Plugin are supported on all the same Microsoft Windows versions as the MySQL server. The MySQL Enterprise Backup product has more comprehensive support for Windows systems than the InnoDB Hot Backup product that it supercedes.</p> <p>See also InnoDB.</p> <p>See also InnoDB Hot Backup.</p> <p>See also MySQL Enterprise Backup.</p> <p>See also plugin.</p> |
| workload | <p>The combination and volume of SQL and other database operations, performed by a database application during typical or peak usage. You can subject the database to a particular workload during performance testing to identify bottlenecks, or during capacity planning.</p> <p>See also bottleneck.</p> <p>See also disk-bound.</p> <p>See also disk-bound.</p> <p>See also SQL.</p> |
| write combining | <p>An optimization technique that reduces write operations when dirty pages are flushed from the InnoDB buffer pool. If a row in a page is updated multiple times,</p> |

or multiple rows on the same page are updated, all of those changes are stored to the data files in a single write operation rather than one write for each change.

See also [buffer pool](#).

See also [dirty page](#).

See also [flush](#).

X

XA A standard interface for coordinating distributed **transactions**, allowing multiple databases to participate in a transaction while maintaining **ACID** compliance. For full details, see [Section 13.3.7, “XA Transactions”](#).

XA Distributed Transaction support is turned on by default. If you are not using this feature, you can disable the `innodb_support_xa` [\[1572\]](#) configuration option, avoiding the performance overhead of an extra fsync for each transaction.

See also [commit](#).

See also [transaction](#).

See also [two-phase commit](#).

Y

young A characteristic of a **page** in the **InnoDB buffer pool** meaning it has been accessed recently, and so is moved within the buffer pool data structure, so that it will not be **flushed** soon by the **LRU** algorithm. This term is used in some **information schema** column names of tables related to the buffer pool.

See also [buffer pool](#).

See also [flush](#).

See also [INFORMATION_SCHEMA](#).

See also [LRU](#).

See also [page](#).

General Index

Symbols

! (logical NOT), 1094
!= (not equal), 1090
", 943
#mysql50 identifier prefix, 945, 949
%, 1122
% (modulo), 1127
% (wildcard character), 938
& (bitwise AND), 1182
&& (logical AND), 1094
() (parentheses), 1088
(Control+Z) \Z, 938, 1348
* (multiplication), 1122
+ (addition), 1121
- (subtraction), 1122
- (unary minus), 1122
--password option, 679
-? option (MySQL Cluster programs), 2241
-c option (MySQL Cluster programs), 2241
-c option (ndb_mgmd) (OBSOLETE), 2188
-d option (ndb_index_stat), 2211
-d option (ndb_mgmd), 2189
-e option (ndb_mgm), 2195
-f option (ndb_mgmd), 2188
-l option (ndbinfo_select_all), 2184
-n option (ndbd), 2180
-n option (ndbmtd), 2180
-p option, 679
-P option (ndb_mgmd), 2190
-V option (MySQL Cluster programs), 2243
.ARM file, 4094
.ARZ file, 4094
.cfg file, 4101
.frm file, 4125
.ibd file, 4131
.ibz file, 4133
.isl file, 4141
.MRG file, 4149
.my.cnf file, 240, 243, 244, 656, 680, 712
.MYD file, 4151
.MYI file, 4151
.mysql_history file, 289, 680
.NET
 and ODBC, 2628
.OPT file, 4157
.PAR file, 4159
.pid (process ID) file, 794
.TRG file, 4188
.TRN file, 4188
/ (division), 1122
/etc/passwd, 688, 1364

:= (assignment operator), 1095
:= (assignment), 956
< (less than), 1090
<< (left shift), 226, 1183
<= (less than or equal), 1090
<=> (equal to), 1090
<> (not equal), 1090
= (assignment operator), 1096
= (assignment), 956
= (equal), 1089
> (greater than), 1090
>= (greater than or equal), 1090
>> (right shift), 1183
[api] (MySQL Cluster), 2119
[computer] (MySQL Cluster), 2121
[mgm] (MySQL Cluster), 2117
[mysqld] (MySQL Cluster), 2119
[ndbd default] (MySQL Cluster), 2105
[ndbd] (MySQL Cluster), 2105
[ndb_mgmd] (MySQL Cluster), 2117
[sci] (MySQL Cluster), 2121
[shm] (MySQL Cluster), 2121
[tcp] (MySQL Cluster), 2121
\" (double quote), 938
' (single quote), 938
\. (mysql client command), 221, 290
\0 (ASCII NUL), 938, 1348
\b (backspace), 938, 1348
\n (linefeed), 938, 1348
\n (newline), 938, 1348
\N (NULL), 1348
\r (carriage return), 938, 1348
\t (tab), 938, 1348
\Z (Control+Z) ASCII 26, 938, 1348
\\ (escape), 938
^ (bitwise XOR), 1182
_ (wildcard character), 938
_rowid, 1295
, 943
| (bitwise OR), 1182
|| (logical OR), 1095
~ (invert bits), 1183

A

abort-slave-event-count option
 mysqld, 1870
aborted clients, 4043
aborted connection, 4043
ABS(), 1123
access control, 705
access denied errors, 4035
access privileges, 692
account names, 703

- accounts
 - adding privileges, 718
 - anonymous user, 171
 - default, 171
 - root, 171
- ACID, 29, 1529, 4091
- ACLs, 692
- ACOS(), 1124
- action option
 - MySQLInstallerConsole, 77
- activating plugins, 619
- Active Server Pages (ASP), 2643
- ActiveState Perl, 195
- adaptive flushing, 4092
- adaptive hash index, 1611, 1669, 1680, 4092
- add-drop-database option
 - mysqldump, 313
- add-drop-table option
 - mysqldump, 313
- add-drop-trigger option
 - mysqldump, 313
- add-locks option
 - mysqldump, 313
- ADDDATE(), 1134
- adding
 - character sets, 1014
 - native functions, 3852
 - new account privileges, 718
 - new functions, 3841
 - new user privileges, 718
 - new users, 165
 - user-defined functions, 3842
- addition (+), 1121
- ADDTIME(), 1134
- addtodest option
 - mysqlhotcopy, 388
- administration
 - server, 293
- administration of MySQL Cluster, 2194
- administrative programs, 234
- ADO.NET, 2687
- AES_DECRYPT(), 1185
- AES_ENCRYPT(), 1185
- After create
 - thread state, 895
- age
 - calculating, 210
- AHI, 4093
- AIO, 4093
- alias names
 - case sensitivity, 946
- aliases
 - for expressions, 1210
 - for tables, 1360
- in GROUP BY clauses, 1210
 - names, 943
 - on expressions, 1360
- ALL, 1363, 1380
- ALL join type
 - optimizer, 839
- all-databases option
 - mysqlcheck, 303
 - mysqldump, 313
- all-in-1 option
 - mysqlcheck, 303
- all-tablespaces option
 - mysqldump, 313
- allocating local table
 - thread state, 902
- allow-keywords option
 - mysqldump, 313
- allow-suspicious-udfs option
 - mysqld, 434
- allowold option
 - mysqlhotcopy, 388
- ALLOW_INVALID_DATES SQL mode, 612
- ALTER COLUMN, 1257
- ALTER DATABASE, 1248
- ALTER EVENT, 1249
 - and replication, 1947
- ALTER FUNCTION, 1252
- ALTER LOGFILE GROUP, 1251
 - (see also MySQL Cluster Disk Data)
- ALTER PROCEDURE, 1252
- ALTER SCHEMA, 1248
- ALTER SERVER, 1253
- ALTER TABLE, 1253, 1258, 4063
 - ROW_FORMAT, 1657
- ALTER TABLESPACE, 1270
 - (see also MySQL Cluster Disk Data)
- ALTER VIEW, 1271
- altering
 - database, 1248
 - schema, 1248
- ANALYSE()
 - PROCEDURE, 820
- analyze option
 - myisamchk, 352
 - mysqlcheck, 303
- ANALYZE TABLE, 1457
 - and partitioning, 2421
- Analyzing
 - thread state, 895
- AND
 - bitwise, 1182
 - logical, 1094
- anonymous user, 171, 172, 706, 708
- ANSI mode

running, 25
 ansi option
 mysqld, 434
 ANSI SQL mode, 612, 617
 ANSI_QUOTES SQL mode, 612
 answering questions
 etiquette, 19
 Antelope, 4093
 Antelope file format, 1651
 ANY, 1379
 Apache, 229
 API node (MySQL Cluster)
 defined, 1972
 API nodes (see SQL nodes)
 APIs, 2565
 list of, 42
 Perl, 3789
 append option (ndb_restore), 2224
 application programming interface (API), 4093
 apply, 4093
 apply-slave-statements option
 mysqldump, 313
 apply_status table (OBSOLETE), 2353
 (see also MySQL Cluster replication)
 approximate-value literals, 1237
 Arbitration, 2068
 ArbitrationDelay, 2037, 2091
 ArbitrationRank, 2037, 2091
 ArbitrationTimeout, 2068
 arbitrator_validity_detail
 ndbinfo table, 2285
 arbitrator_validity_summary
 ndbinfo table, 2286
 ARCHIVE storage engine, 1524, 1705
 Area(), 1229, 1230
 argument processing, 3847
 arithmetic expressions, 1121
 arithmetic functions, 1182
 AS, 1360, 1366
 AsBinary(), 1225
 ASCII(), 1101
 ASIN(), 1124
 ASP.NET, 2697, 2779
 assignment operator
 :=, 1095
 =, 1096
 assignment operators, 1095
 AsText(), 1225
 asynchronous I/O, 1671, 4094
 asynchronous replication (see MySQL Cluster replication)
 ATAN(), 1124
 ATAN2(), 1124
 atomic, 4094
 atomic instruction, 4094
 attackers
 security against, 687
 attribute demotion
 replication, 1943
 attribute promotion
 ndb_restore, 2221
 replication, 1943
 audit log plugin
 , 757
 audit plugins, 3799
 audit-log option
 mysqld, 764
 audit_log_buffer_size system variable, 764
 audit_log_file system variable, 765
 audit_log_flush system variable, 765
 audit_log_policy system variable, 766
 audit_log_rotate_on_size system variable, 766
 audit_log_strategy system variable, 767
 authentication plugins, 3800
 AUTHENTICATION_PAM_LOG environment variable, 735
 authentication_windows_log_level system variable, 482
 authentication_windows_use_principal_name system variable, 483
 auto-generate-sql option
 mysqslap, 337
 auto-generate-sql-add-autoincrement option
 mysqslap, 337
 auto-generate-sql-execute-number option
 mysqslap, 337
 auto-generate-sql-guid-primary option
 mysqslap, 338
 auto-generate-sql-load-type option
 mysqslap, 338
 auto-generate-sql-secondary-indexes option
 mysqslap, 338
 auto-generate-sql-unique-query-number option
 mysqslap, 338
 auto-generate-sql-unique-write-number option
 mysqslap, 338
 auto-generate-sql-write-number option
 mysqslap, 338
 AUTO-INCREMENT
 ODBC, 2639
 auto-increment, 4094
 auto-increment locking, 4095
 auto-rehash option
 mysql, 276
 auto-repair option
 mysqlcheck, 304
 auto-vertical-output option
 mysql, 276
 autocommit, 4095

autocommit session variable, 484
 automatic_sp_privileges system variable, 484
 AutoReconnect
 API and SQL nodes, 2093
 AUTO_INCREMENT, 226, 1044
 and NULL values, 4059
 and replication, 1938
 auto_increment_increment system variable, 1865
 auto_increment_offset system variable, 1868
 AVG(), 1203
 AVG(DISTINCT), 1203

B

B-tree, 4096
 B-tree indexes, 815, 1610
 background threads
 master, 1672, 1673
 read, 1671
 write, 1671
 backslash
 escape character, 937
 backspace (b), 938, 1348
 backticks, 4096
 backup, 4096
 backup identifiers
 native backup and restore, 2251
 backup option
 mysamchk, 350
 mysampack, 362
 BackupDataBufferSize, 2074, 2253
 BackupDataDir, 2042
 backupid option (ndb_restore), 2221
 BackupLogBufferSize, 2074, 2253
 BackupMaxWriteSize, 2076, 2253
 BackupMemory, 2075, 2253
 BackupReportFrequency, 2075
 backups, 771, 3864
 databases and tables, 307, 386
 in MySQL Cluster, 2217, 2249, 2250, 2250, 2253
 in MySQL Cluster replication, 2360
 InnoDB, 1592
 with mysqldump, 780
 backups, troubleshooting
 in MySQL Cluster, 2254
 BackupWriteSize, 2075, 2253
 backup_path option (ndb_restore), 2222
 back_log system variable, 485
 Barracuda, 4097
 Barracuda file format, 1651
 base64-output option
 mysqlbinlog, 372
 BaseCommandInterceptor class, 2792
 basedir option
 mysql.server, 260
 mysqld, 435
 mysqld_safe, 256
 mysql_install_db, 266
 mysql_plugin, 268
 mysql_upgrade, 271
 basedir system variable, 485
 BaseExceptionInterceptor class, 2793
 batch mode, 220
 batch option
 mysql, 276
 batch SQL files, 272
 BatchByteSize, 2092
 batched updates (MySQL Cluster Replication), 2357
 BatchSize, 2092
 BatchSizePerLocalScan, 2050
 Bazaar tree, 146
 BdMPolyFromText(), 1221
 BdMPolyFromWKB(), 1222
 BdPolyFromText(), 1221
 BdPolyFromWKB(), 1222
 BEGIN, 1390, 1419
 labels, 1420
 XA transactions, 1404
 BENCHMARK(), 1190
 benchmarks, 891, 892
 beta, 4097
 BETWEEN ... AND, 1091
 big-tables option
 mysqld, 435
 big5, 3957
 BIGINT data type, 1037
 big_tables session variable, 485
 BIN(), 1101
 BINARY, 1168
 BINARY data type, 1043, 1060
 binary distributions
 installing, 56
 binary log, 634, 4098
 event groups, 1413
 binary logging
 and MySQL Cluster, 1993
 bind-address option
 mysql, 277
 mysqladmin, 298
 mysqlbinlog, 373
 mysqlcheck, 304
 mysqld, 435
 mysqldump, 313
 mysqlexport, 327
 mysqlshow, 332
 bind-address option (ndb_mgmd), 2187
 BINLOG, 1511
 binlog, 4098

Binlog Dump
 thread command, 893
 BINLOG statement
 mysqlbinlog output, 381
 binlog-do-db option
 mysqld, 1894
 binlog-format option
 mysqld, 436
 binlog-ignore-db option
 mysqld, 1896
 binlog-row-event-max-size option
 mysqld, 1891
 binlog_cache_size system variable, 1898
 binlog_direct_non_transactional_updates system variable, 1898
 binlog_format system variable, 1899
 binlog_index table (OBSOLETE) (see MySQL Cluster replication)
 binlog_stmt_cache_size system variable, 1901
 BIT data type, 1036
 bit functions, 1182
 example, 226
 BIT_AND(), 1204
 BIT_COUNT, 226
 BIT_COUNT(), 1183
 BIT_LENGTH(), 1101
 BIT_OR, 226
 BIT_OR(), 1204
 BIT_XOR(), 1204
 BLACKHOLE storage engine, 1524, 1706
 blind query expansion, 4098
 BLOB
 inserting binary data, 939
 size, 1071
 BLOB columns
 default values, 1062
 indexing, 811, 1295
 with Connector/Net, 2759
 BLOB data type, 1043, 1061
 blob-info option
 ndb_desc, 2207
 Block Nested-Loop join algorithm, 917
 block-search option
 myisamchk, 352
 blocks
 ndbinfo table, 2287
 BOOL data type, 1036
 BOOLEAN data type, 1036
 boolean options, 243
 bootstrap option
 mysqld, 436
 Borland applications, 2643
 bottleneck, 4098
 bounce, 4099
 Boundary(), 1227
 brackets
 square, 1036
 brief option
 mysqlaccess, 368
 buddy allocator, 4099
 buffer, 4099
 buffer pool, 848, 1674, 4099
 buffer pool instance, 4100
 buffer sizes, 848
 client, 2565
 mysqld server, 868
 Buffer(), 1231
 bugs
 known, 4064
 MySQL Cluster reporting, 2209
 reporting, 2, 20
 bugs database, 20
 bugs.mysql.com, 20
 BuildIndexThreads, 2078
 building
 client programs, 3183
 BUILD_CONFIG option
 CMake, 150
 built-in, 4100
 bulk loading
 for InnoDB tables, 824
 for MyISAM tables, 830
 with Connector/Net, 2766
 bulk_insert_buffer_size system variable, 486
 business rules, 4100

C

C API
 data types, 3071
 functions, 3077
 linking problems, 3183
 C prepared statement API
 functions, 3139, 3140
 type codes, 3137
 C#, 2697, 2791
 C++ Builder, 2644
 C:\my.cnf file, 656
 cache, 4100
 CACHE INDEX, 1511
 and partitioning, 2430
 caches
 clearing, 1513
 calculating
 aggregate value for a set of rows, 1203
 cardinality, 1488
 dates, 210

calendar, 1152
CALL, 1327
calling sequences for aggregate functions
 UDF, 3846
calling sequences for simple functions
 UDF, 3844
can't create/write to file, 4045
cardinality, 801, 4101
carriage return (r), 938, 1348
CASE, 1097, 1423
case sensitivity
 in access checking, 703
 in identifiers, 946
 in names, 946
 in searches, 4055
 in string comparisons, 1111
 of database names, 26
 of replication filtering options, 1913
 of table names, 26
CAST, 1168
cast functions, 1168
cast operators, 1168
casts, 1083, 1089, 1168
catalog option
 MySQLInstallerConsole, 77
CC environment variable, 158, 193
CEIL(), 1124
CEILING(), 1125
Centroid(), 1230
certificate file, 2784
CFLAGS environment variable, 158, 193
cflags option
 mysql_config, 394
change buffer, 4101
change buffering, 4102
 disabling, 1668
CHANGE MASTER TO, 1408
 in MySQL Cluster, 2355
Change user
 thread command, 893
ChangeLog, 4069
changes
 release notes, 4069
changes to privileges, 710
changing
 column, 1257
 field, 1257
 socket location, 168, 4055
 table, 1253, 1258, 4063
Changing master
 thread state, 906
CHAR data type, 1042, 1058
CHAR VARYING data type, 1042
CHAR(), 1101
CHARACTER data type, 1042
character set repertoire, 995
character sets, 963
 adding, 1014
 and replication, 1938
 repertoire, 987
 restrictions, 4081
 with Connector/J, 2935
 with Connector/Net, 2774
CHARACTER VARYING data type, 1042
character-set-client-handshake option
 mysql, 437
character-set-filesystem option
 mysql, 437
character-set-server option
 mysql, 438
character-sets-dir option
 myisamchk, 350
 myisampack, 362
 mysql, 277
 mysqladmin, 298
 mysqlbinlog, 373
 mysqlcheck, 304
 mysql, 437
 mysqldump, 314
 mysqlimport, 327
 mysqlshow, 332
character-sets-dir option (MySQL Cluster programs),
2241, 2241
characters
 multi-byte, 1017
CHARACTER_LENGTH(), 1102
CHARACTER_SETS
 INFORMATION_SCHEMA table, 2472
character_sets_dir system variable, 488
character_set_client system variable, 486
character_set_connection system variable, 487
character_set_database system variable, 487
character_set_filesystem system variable, 487
character_set_results system variable, 488
character_set_server system variable, 488
character_set_system system variable, 488
charset command
 mysql, 284
charset option
 comp_err, 265
CHARSET(), 1191
CHAR_LENGTH(), 1102
check option
 myisamchk, 349
 mysqlcheck, 304
check options
 myisamchk, 349
CHECK TABLE, 1458

- and partitioning, 2421
- check-only-changed option
 - myisamchk, 349
 - mysqlcheck, 304
- check-upgrade option
 - mysqlcheck, 304
- checking
 - tables for errors, 790
- Checking master version
 - thread state, 904
- checking permissions
 - thread state, 895
- checking privileges on cached query
 - thread state, 903
- checking query cache for query
 - thread state, 903
- Checking table
 - thread state, 896
- checkpoint, 4103
- CHECKPOINT Events (MySQL Cluster), 2261
- checkpoint option
 - mysqlhotcopy, 388
- Checksum, 2097, 2103
- checksum, 4103
- Checksum (MySQL Cluster), 2100
- checksum errors, 135
- CHECKSUM TABLE, 1460
- child table, 4103
- Chinese, Japanese, Korean character sets
 - frequently asked questions, 3957
- choosing
 - a MySQL version, 49
 - data types, 1072
- chroot option
 - mysqld, 438
 - mysqlhotcopy, 388
- circular replication
 - in MySQL Cluster, 2346, 2366, 2371
- CJK (Chinese, Japanese, Korean)
 - Access, PHP, etc., 3957
 - availability of specific characters, 3957
 - big5, 3957
 - character sets available, 3957
 - characters displayed as question marks, 3957
 - CJKV, 3957
 - collations, 3957, 3957
 - conversion problems with Japanese character sets, 3957
 - data truncation, 3957
 - Database and table names, 3957
 - documentation in Chinese, 3957
 - documentation in Japanese, 3957
 - documentation in Korean, 3957
 - FAQ, 3957
 - gb2312, gbk, 3957
 - Japanese character sets, 3957
 - Korean character set, 3957
 - LIKE and FULLTEXT, 3957
 - MySQL 4.0 behavior, 3957
 - ORDER BY treatment, 3957, 3957
 - problems with Access, PHP, etc., 3957
 - problems with Big5 character sets (Chinese), 3957
 - problems with data truncation, 3957
 - problems with euckr character set (Korean), 3957
 - problems with GB character sets (Chinese), 3957
 - problems with LIKE and FULLTEXT, 3957
 - problems with Yen sign (Japanese), 3957
 - rejected characters, 3957
 - sort order problems, 3957, 3957
 - sorting problems, 3957, 3957
 - testing availability of characters, 3957
 - Unicode collations, 3957
 - Vietnamese, 3957
 - Yen sign, 3957
- clean page, 4104
- clean shutdown, 4104
- cleaning up
 - thread state, 896
- clear command
 - mysql, 284
- Clearing
 - thread state, 907
- clearing
 - cached, 1513
- client, 4104
- client connection threads, 882
- client programs, 233
 - building, 3183
- client tools, 2565
- clients
 - debugging, 3861
 - threaded, 3184
- CLOSE, 1428
- Close stmt
 - thread command, 893
- closing
 - tables, 820
- closing tables
 - thread state, 896
- cluster database (OBSOLETE) (see MySQL Cluster replication)
- cluster logs, 2258, 2260
- cluster.binlog_index table (OBSOLETE) (see MySQL Cluster replication)
- clustered index, 4104
 - InnoDB, 1610
- Clustering (see MySQL Cluster)
- CLUSTERLOG commands (MySQL Cluster), 2260

CLUSTERLOG STATISTICS command (MySQL Cluster), 2264
 cluster_operations
 ndbinfo table, 2287
 cluster_replication database (OBSOLETE) (see MySQL Cluster replication)
 cluster_transactions
 ndbinfo table, 2288
 CMake
 BUILD_CONFIG option, 150
 CMAKE_BUILD_TYPE option, 151
 CMAKE_INSTALL_PREFIX option, 151
 COMPILATION_COMMENT option, 154
 CPACK_MONOLITHIC_INSTALL option, 151
 DEFAULT_CHARSET option, 154
 DEFAULT_COLLATION option, 154
 DISABLE_GRANT_OPTIONS option, 156
 ENABLED_LOCAL_INFILE option, 154
 ENABLED_PROFILING option, 155
 ENABLE_DEBUG_SYNC option, 154
 ENABLE_DOWNLOADS option, 154
 ENABLE_DTRACE option, 154
 ENABLE_GCOV option, 154
 HAVE_EMBEDDED_PRIVILEGE_CONTROL option, 156
 INSTALL_BINDIR option, 151
 INSTALL_DOCDIR option, 151
 INSTALL_DOCREADMEDIR option, 151
 INSTALL_INCLUDEDIR option, 151
 INSTALL_INFODIR option, 151
 INSTALL_LAYOUT option, 151
 INSTALL_LIBDIR option, 152
 INSTALL_MANDIR option, 152
 INSTALL_MYSQLSHAREDIR option, 152
 INSTALL_MYSQLTESTDIR option, 152
 INSTALL_PLUGINDIR option, 152
 INSTALL_SBINDIR option, 152
 INSTALL_SCRIPTDIR option, 152
 INSTALL_SHAREDIR option, 152
 INSTALL_SQLBENCHDIR option, 152
 INSTALL_SUPPORTFILESDIR option, 152
 MEMCACHED_HOME option, 156
 MYSQL_DATADIR option, 152
 MYSQL_MAINTAINER_MODE option, 155
 MYSQL_PROJECT_NAME option, 155
 MYSQL_TCP_PORT option, 155
 MYSQL_UNIX_ADDR option, 155
 ODBC_INCLUDES option, 153
 ODBC_LIB_DIR option, 153
 options, 147
 running after prior invocation, 144, 158
 SYSCONFDIR option, 153
 VERSION file, 159
 WITH_BUNDLED_LIBEVENT option, 157
 WITH_BUNDLED_MEMCACHED option, 157
 WITH_CLASSPATH option, 157
 WITH_DEBUG option, 155
 WITH_EMBEDDED_SERVER option, 155
 WITH_ERROR_INSERT option, 157
 WITH_EXTRA_CHARSETS option, 155
 WITH_LIBWRAP option, 155
 WITH_NDBCLUSTER option, 157
 WITH_NDBCLUSTER_STORAGE_ENGINE option, 157
 WITH_NDBMTD option, 157
 WITH_NDB_BINLOG option, 157
 WITH_NDB_DEBUG option, 157
 WITH_NDB_JAVA option, 157
 WITH_NDB_PORT option, 158
 WITH_NDB_TEST option, 158
 WITH_READLINE option, 155
 WITH_SSL option, 156
 WITH_UNIXODBC option, 153
 WITH_ZLIB option, 156
 CMakeCache.txt file, 158
 CMAKE_BUILD_TYPE option
 CMake, 151
 CMAKE_INSTALL_PREFIX option
 CMake, 151
 COALESCE(), 1092
 COERCIBILITY(), 1191
 cold backup, 4105
 ColdFusion, 2644
 collating
 strings, 1017
 collation
 adding, 1017
 INFORMATION_SCHEMA, 986
 modifying, 1018
 names, 979
 COLLATION(), 1191
 collation-server option
 mysqld, 438
 COLLATIONS
 INFORMATION_SCHEMA table, 2473
 COLLATION_CHARACTER_SET_APPLICABILITY
 INFORMATION_SCHEMA table, 2473
 collation_connection system variable, 489
 collation_database system variable, 489
 collation_server system variable, 489
 column, 4105
 changing, 1257
 types, 1035
 column alias
 problems, 4059
 quoting, 944, 4059
 column comments, 1293
 Column Editor, 2667

- column format, 1293
- column index, 4105
- column names
 - case sensitivity, 946
- column prefix, 4105
- column storage, 1294
- column-names option
 - mysql, 277
- column-type-info option
 - mysql, 277
- columns
 - displaying, 330
 - indexes, 811
 - names, 943
 - other types, 1073
 - selecting, 208
 - storage requirements, 1069
- COLUMNS
 - INFORMATION_SCHEMA table, 2473
- columns option
 - mysqlimport, 327
- columns partitioning, 2393
- COLUMN_PRIVILEGES
 - INFORMATION_SCHEMA table, 2475
- comma-separated values data, reading, 1346, 1365
- command options
 - mysql, 273
 - mysqladmin, 296
 - mysqld, 433
- command options (MySQL Cluster)
 - mysqld, 2138
 - ndbd, 2176
 - ndbinfo_select_all, 2184
 - ndb_mgm, 2194
 - ndb_mgmd, 2186
- command syntax, 4
- command-line history
 - mysql, 289
- command-line options (MySQL Cluster), 2240
- command-line tool, 76, 272
- commands
 - for binary distribution, 57
- commands out of sync, 4045
- comment syntax, 961
- comments
 - adding, 961
 - starting, 32
- comments option
 - mysql, 277
 - mysqldump, 314
- COMMIT, 29, 1390
 - XA transactions, 1404
- commit, 4106
- commit option
 - mysqlaccess, 368
 - mysqlslap, 338
- Committing events to binlog
 - thread state, 906
- compact option
 - mysqldump, 314
- compact row format, 4106
- comparison operators, 1088
- compatibility
 - between MySQL versions, 177, 183
 - with mSQL, 1114
 - with ODBC, 554, 946, 1039, 1084, 1091, 1293, 1368
 - with Oracle, 27, 1206, 1257, 1519
 - with PostgreSQL, 28
 - with standard SQL, 24
- compatible option
 - mysqldump, 314
- COMPILATION_COMMENT option
 - CMake, 154
- compiling
 - optimizing, 868
 - problems, 158
 - user-defined functions, 3850
- compiling clients
 - on Unix, 3183
 - on Windows, 3183
- complete-insert option
 - mysqldump, 314
- completion_type system variable, 489
- composite index, 4106
- composite partitioning, 2406
- compound statements, 1419
- compress option
 - mysql, 277
 - mysqladmin, 298
 - mysqlcheck, 304
 - mysqldump, 314
 - mysqlimport, 327
 - mysqlshow, 332
 - mysqlslap, 338
- COMPRESS(), 1185
- compressed backup, 4106
- compressed row format, 4107
- compressed tables, 361, 1697
- CompressedBackup, 2059
- CompressedLCP, 2060
- compression, 4107
 - algorithms, 1649
 - application and schema design, 1646
 - BLOBs, VARCHAR and TEXT, 1650
 - buffer pool, 1651
 - compressed page size, 1648
 - configuration characteristics, 1647
 - data and indexes, 1649

- data characteristics, 1646
- enabling for a table, 1642
- implementation, 1648
- information schema, 1658, 1658
- innodb_strict_mode, 1681
- KEY_BLOCK_SIZE, 1648
- log files, 1651
- modification log, 1649
- monitoring, 1648
- overflow pages, 1650
- overview, 1642
- tuning, 1645
- with Connector/Net network traffic, 2784
- workload characteristics, 1647
- compression failure, 4108
- comp_err, 233, 264
 - charset option, 265
 - debug option, 265
 - debug-info option, 265
 - header_file option, 265
 - help option, 264
 - in_file option, 265
 - name_file option, 265
 - out_dir option, 265
 - out_file option, 265
 - statefile option, 265
 - version option, 265
- CONCAT(), 1102
- concatenation
 - string, 937, 1102
- CONCAT_WS(), 1102
- concurrency, 1529, 4109
 - of commits, 1551
 - of threads, 1573
 - tickets, 1552
- concurrency option
 - mysqslap, 338
- concurrent inserts, 863, 866
- concurrent_insert system variable, 490
- Conditions, 1430
- conditions, 1485, 1509
- cond_instances table
 - performance_schema, 2543
- config option
 - MySQLInstallerConsole, 77
- config-cache option (ndb_mgmd), 2188
- config-file option
 - mysqld_multi, 261
 - my_print_defaults, 396
 - ndb_config, 2197
- config-file option (ndb_mgmd), 2188
- config.ini (MySQL Cluster), 2014, 2025, 2026, 2193
- configdir option (ndb_mgmd), 2187
- configinfo option
 - ndb_config, 2200
- configuration
 - MySQL Cluster, 2104
- configuration file, 4109
- configuration files, 712
- configuring backups
 - in MySQL Cluster, 2253
- configuring MySQL Cluster, 1997, 2022, 2193, 2254
- Configuring MySQL Cluster (concepts), 1972
- config_from_node option
 - ndb_config, 2197
- config_params
 - ndbinfo table, 2289
- conflict resolution
 - and ndb_replication system table, 2373
 - enabling, 2373
 - in MySQL Cluster Replication, 2371
 - mysqld startup options, 2372
- Connect
 - thread command, 893
- connect command
 - mysql, 285
- Connect Out
 - thread command, 893
- connect-delay option (ndbd), 2181
- connect-delay option (ndbmtd), 2181
- connect-retries option (ndbd), 2181
- connect-retries option (ndbmtd), 2181
- connect-string option (MySQL Cluster programs), 2241
- ConnectCheckIntervalDelay, 2064
- connecting
 - remotely with SSH, 756
 - through Connector/Net, 2743
 - through JDBC and Connector/J, 2911
 - through Visual Studio, 2663
 - to the server, 197, 237
 - verification, 705
- Connecting to master
 - thread state, 904
- connection
 - aborted, 4043
- CONNECTION Events (MySQL Cluster), 2261
- connection pooling, 2749, 2784, 2954, 2969, 2969
- connection string, 2743, 2784
- connections option
 - ndb_config, 2199
- CONNECTION_ID(), 1191
- Connector/C, 2565, 3029
- Connector/C++, 2565, 2985
- Connector/J, 2565, 2903
 - reporting problems, 2984
 - troubleshooting, 2975
- Connector/Net, 2565, 2651
 - reporting problems, 2901

- Connector/ODBC, 2565, 2567
 - reporting problems, 2650, 2650
 - with Borland applications, 2643
- Connector/Python, 2565, 3034
- Connectors, 2565
- connectstring (see MySQL Cluster)
- connect_timeout system variable, 492
- connect_timeout variable, 283, 300
- consistent read, 4109
- consistent reads, 1599
- console option
 - mysqld, 438
- const table
 - optimizer, 837, 1363
- constant table, 800
- constraint, 4110
- constraints, 33
- CONSTRAINTS
 - INFORMATION_SCHEMA table, 2493
- Contains(), 1233
- contributing companies
 - list of, 43
- contributors
 - list of, 36
- control flow functions, 1097
- CONV(), 1125
- conventions
 - syntax, 3
 - typographical, 3
- CONVERT, 1168
- CONVERT TO, 1259
- converting HEAP to MyISAM
 - thread state, 896
- CONVERT_TZ(), 1134
- ConvexHull(), 1232
- copy option
 - mysqlaccess, 368
- copy to tmp table
 - thread state, 896
- copying databases, 191
- copying tables, 1307, 1308
- Copying to group table
 - thread state, 896
- Copying to tmp table
 - thread state, 896
- Copying to tmp table on disk
 - thread state, 896
- core-file option
 - mysqld, 439
- core-file option (MySQL Cluster programs), 2242
- core-file-size option
 - mysqld_safe, 256
- correct-checksum option
 - myisamchk, 350
- correlated subqueries, 1382
- COS(), 1125
- COT(), 1125
- count option
 - myisam_ftdump, 343
 - mysqladmin, 298
 - mysqlshow, 332
- COUNT(), 1204
- COUNT(DISTINCT), 1204
- counter, 4110
- counters
 - ndbinfo table, 2289
- counting
 - table rows, 215
- covering index, 4110
- CPACK_MONOLITHIC_INSTALL option
 - CMake, 151
- crash, 3854, 4111
 - recovery, 789
 - repeated, 4051
 - replication, 1952
- crash recovery, 4111
- crash-me, 891
- crash-me program, 891
- CrashOnCorruptedTuple, 2058
- CRC32(), 1125
- CREATE ... IF NOT EXISTS
 - and replication, 1939
- CREATE DATABASE, 1272
- Create DB
 - thread command, 893
- CREATE EVENT, 1272
 - and replication, 1947
- CREATE FUNCTION, 1283, 1465
- CREATE INDEX, 1277, 1638
- CREATE LOGFILE GROUP, 1281
 - (see also MySQL Cluster Disk Data)
- CREATE NODEGROUP command (MySQL Cluster), 2248
- create option
 - mysqlslap, 338
- CREATE PROCEDURE, 1283
- CREATE SCHEMA, 1272
- CREATE SERVER, 1288
- CREATE TABLE, 1289
 - DIRECTORY options
 - and replication, 1947
 - KEY_BLOCK_SIZE, 1648
 - options for table compression, 1642
 - ROW_FORMAT, 1657
- CREATE TABLE ... SELECT
 - and replication, 1939
- CREATE TABLESPACE, 1312
 - (see also MySQL Cluster Disk Data)

CREATE TRIGGER, 1313
CREATE USER, 1443
CREATE VIEW, 1317
create-and-drop-schema option
 mysqslap, 338
create-options option
 mysqldump, 314
create-schema option
 mysqslap, 338
creating
 bug reports, 20
 database, 1272
 databases, 201
 default startup options, 243
 function, 1465
 schema, 1272
 tables, 203
Creating delayed handler
 thread state, 902
Creating index
 thread state, 896
Creating sort index
 thread state, 896
creating table
 thread state, 896
Creating tmp table
 thread state, 896
creating user accounts, 1443
CROSS JOIN, 1366
Crosses(), 1233
CRUD, 4112
Crystal Reports, 2774
CR_SERVER_GONE_ERROR, 4041
CR_SERVER_LOST_ERROR, 4041
CSV data, reading, 1346, 1365
csv option
 mysqslap, 339
CSV storage engine, 1524, 1703
CURDATE(), 1135
CURRENT_DATE, 1135
CURRENT_TIME, 1135
CURRENT_TIMESTAMP, 1135
CURRENT_USER(), 1192
cursor, 4112
Cursors, 1427
CURTIME(), 1135
CXX environment variable, 158, 193
CXXFLAGS environment variable, 158, 193

D

Daemon
 thread command, 894
daemon option (ndb_mgmd), 2189

daemon plugins, 3799
data
 importing, 290, 325
 loading into tables, 204
 retrieving, 206
 size, 817
data dictionary, 4112
DATA DIRECTORY
 and replication, 1947
data directory, 4113
data files, 4113
data node (MySQL Cluster)
 defined, 1972
data nodes (MySQL Cluster), 2176, 2184
Data on disk (MySQL Cluster)
 and INFORMATION_SCHEMA.FILES table, 2505
Data truncation with CJK characters, 3957
data type
 BIGINT, 1037
 BINARY, 1043, 1060
 BIT, 1036
 BLOB, 1043, 1061
 BOOL, 1036, 1073
 BOOLEAN, 1036, 1073
 CHAR, 1042, 1058
 CHAR VARYING, 1042
 CHARACTER, 1042
 CHARACTER VARYING, 1042
 DATE, 1039, 1050
 DATETIME, 1039, 1050
 DEC, 1038
 DECIMAL, 1038, 1237
 DOUBLE, 1039
 DOUBLE PRECISION, 1039
 ENUM, 1044, 1062
 FIXED, 1038
 FLOAT, 1038, 1039, 1039
 GEOMETRY, 1220
 GEOMETRYCOLLECTION, 1220
 INT, 1037
 INTEGER, 1037
 LINestring, 1220
 LONG, 1061
 LONGBLOB, 1043
 LONGTEXT, 1044
 MEDIUMBLOB, 1043
 MEDIUMINT, 1037
 MEDIUMTEXT, 1043
 MULTILINESTRING, 1220
 MULTIPOINT, 1220
 MULTIPOLYGON, 1220
 NATIONAL CHAR, 1042
 NATIONAL VARCHAR, 1042
 NCHAR, 1042

NUMERIC, 1038
 NVARCHAR, 1042
 POINT, 1220
 POLYGON, 1220
 REAL, 1039
 SET, 1044, 1065
 SMALLINT, 1037
 TEXT, 1043, 1061
 TIME, 1040, 1051
 TIMESTAMP, 1039, 1050
 TINYBLOB, 1043
 TINYINT, 1036
 TINYTEXT, 1043
 VARBINARY, 1043, 1060
 VARCHAR, 1042, 1058
 VARCHARACTER, 1042
 YEAR, 1040, 1051
 data types, 1035
 C API, 3071
 overview, 1036
 data warehouse, 4113
 data-file-length option
 myisamchk, 350
 database, 4113
 altering, 1248
 creating, 1272
 deleting, 1321
 Database information
 obtaining, 1471
 database metadata, 2469
 database names
 case sensitivity, 26, 946
 database option
 mysql, 277
 mysqlbinlog, 373
 ndb_show_tables, 2234
 database option (ndb_index_stat), 2211
 DATABASE(), 1193
 databases
 backups, 771
 copying, 191
 creating, 201
 defined, 5
 displaying, 330
 dumping, 307, 386
 information about, 219
 names, 943
 replicating, 1839
 selecting, 202
 symbolic links, 877
 using, 201
 databases option
 mysqlcheck, 304
 mysqldump, 314
 DataDir, 2037, 2042
 datadir option
 mysql.server, 260
 mysqld, 439
 mysqld_safe, 256
 mysql_install_db, 266
 mysql_plugin, 268
 mysql_upgrade, 271
 datadir system variable, 492
 DataJunction, 2645
 DataMemory, 2043
 DATE, 4057
 date and time functions, 1131
 Date and Time types, 1048
 date calculations, 210
 DATE columns
 problems, 4057
 with Connector/Net, 2765
 DATE data type, 1039, 1050
 date literals, 940
 date types, 1070
 date values
 problems, 1050
 DATE(), 1135
 DATEDIFF(), 1135
 dates
 used with partitioning, 2386
 used with partitioning (examples), 2389, 2402, 2407, 2425
 DATETIME data type, 1039, 1050
 datetime_format system variable, 492
 DATE_ADD(), 1135
 date_format system variable, 492
 DATE_FORMAT(), 1138
 DATE_SUB(), 1135, 1139
 DAY(), 1139
 DAYNAME(), 1140
 DAYOFMONTH(), 1140
 DAYOFWEEK(), 1140
 DAYOFYEAR(), 1140
 db option
 mysqlaccess, 368
 db table
 sorting, 709
 DB2 SQL mode, 618
 DBI interface, 3789
 DBI->quote, 939
 DBI->trace, 3857
 DBI/DBD interface, 3789
 DBI_TRACE environment variable, 193, 3857
 DBI_USER environment variable, 193
 DEBUG package, 3861
 DCL, 1445, 1455, 4114
 DDL, 4114

deadlock, 4114
deadlock detection, 4115
DEALLOCATE PREPARE, 1415, 1419
Debug
 thread command, 894
debug option
 comp_err, 265
 myisamchk, 347
 myisampack, 362
 mysql, 277
 mysqlaccess, 368
 mysqladmin, 298
 mysqlbinlog, 374
 mysqlcheck, 304
 mysqld, 439
 mysqldump, 314
 mysqldumpslow, 385
 mysqlhotcopy, 388
 mysqlimport, 327
 mysqlshow, 332
 mysqslap, 339
 my_print_defaults, 396
debug option (MySQL Cluster programs), 2242
debug system variable, 492
debug-check option
 mysql, 277
 mysqladmin, 298
 mysqlbinlog, 374
 mysqlcheck, 304
 mysqldump, 315
 mysqlimport, 327
 mysqlshow, 332
 mysqslap, 339
 mysql_upgrade, 271
debug-info option
 comp_err, 265
 mysql, 277
 mysqladmin, 298
 mysqlbinlog, 374
 mysqlcheck, 304
 mysqldump, 315
 mysqlimport, 327
 mysqlshow, 332
 mysqslap, 339
 mysql_upgrade, 271
debug-sync-timeout option
 mysqld, 440
debugging
 client, 3861
 server, 3854
debugging support, 147
debug_sync system variable, 493
DEC data type, 1038
decimal arithmetic, 1237
DECIMAL data type, 1038, 1237
decimal point, 1036
DECLARE, 1421
DECODE(), 1186
decode_bits myisamchk variable, 347
DEFAULT
 constraint, 34
default
 privileges, 171
default accounts, 171
default host name, 237
default installation location, 55
default options, 243
DEFAULT value clause, 1068, 1293
default values, 1068, 1293, 1335
 BLOB and TEXT columns, 1062
 explicit, 1068
 implicit, 1068
 suppression, 34
DEFAULT(), 1198
default-auth option
 mysql, 277
 mysqladmin, 298
 mysqlbinlog, 374
 mysqlcheck, 305
 mysqldump, 315
 mysqlimport, 328
 mysqlshow, 333
 mysqslap, 339
 mysql_upgrade, 271
default-character-set option
 mysql, 277
 mysqladmin, 298
 mysqlcheck, 304
 mysqld, 440
 mysqldump, 315
 mysqlimport, 327
 mysqlshow, 333
default-collation option
 mysqld, 440
default-storage-engine option
 mysqld, 441
default-time-zone option
 mysqld, 441
DefaultOperationRedoProblemAction
 API and SQL nodes, 2094
defaults
 embedded, 3067
defaults-extra-file option, 248
 mysqld_multi, 261
 mysqld_safe, 256
 my_print_defaults, 396
defaults-file option, 248
 mysqld_multi, 261

- mysql_safe, 256
- my_print_defaults, 396
- defaults-group-suffix option, 248
 - my_print_defaults, 396
- DEFAULT_CHARSET option
 - CMake, 154
- DEFAULT_COLLATION option
 - CMake, 154
- default_storage_engine system variable, 494
- default_week_format system variable, 494
- DEGREES(), 1125
- delay option (ndbinfo_select_all), 2184
- delay-key-write option
 - mysqld, 441, 1694
- DELAYED, 1339
 - when ignored, 1337
- Delayed insert
 - thread command, 894
- delayed inserts
 - thread states, 902
- delayed-insert option
 - mysqldump, 315
- delayed_insert_limit, 1341
- delayed_insert_limit system variable, 495
- delayed_insert_timeout system variable, 496
- delayed_queue_size system variable, 496
- delay_key_write system variable, 494
- DELETE, 1329
 - and MySQL Cluster, 1988
- delete, 4116
- delete buffering, 4116
- delete option
 - mysqlimport, 328
- delete option (ndb_index_stat), 2212
- delete-master-logs option
 - mysqldump, 315
- deleting
 - database, 1321
 - foreign key, 1259, 1587
 - function, 1466
 - index, 1258, 1322
 - primary key, 1258
 - rows, 4060
 - schema, 1321
 - table, 1323
 - user, 721, 1444
 - users, 721, 1444
- deleting from main table
 - thread state, 896
- deleting from reference tables
 - thread state, 897
- deletion
 - mysql.sock, 4054
- delimiter command
 - mysql, 285
- delimiter option
 - mysql, 278
 - mysqlslap, 339
 - ndb_select_all, 2231
- Delphi, 2644
- denormalized, 4116
- derived tables, 1382
- des-key-file option
 - mysqld, 442
- DESC, 1519
- descending index, 4116
- descending option
 - ndb_select_all, 2231
- DESCRIBE, 219, 1519
- description option
 - myisamchk, 352
- design
 - issues, 4064
- DES_DECRYPT(), 1186
- DES_ENCRYPT(), 1186
- detach option
 - mysqlslap, 339
- development of MySQL Cluster, 1979
- development source tree, 146
- Difference(), 1232
- digits, 1036
- Dimension(), 1226
- directory structure
 - default, 55
- dirty page, 1547, 4116
- dirty read, 4117
- disable named command
 - mysql, 278
- disable option prefix, 243
- disable-indexes option (ndb_restore), 2228
- disable-keys option
 - mysqldump, 315
- disable-log-bin option
 - mysqlbinlog, 375
- DISABLE_GRANT_OPTIONS option
 - CMake, 156
- DISCARD TABLESPACE, 1259, 1539
- discard_or_import_tablespace
 - thread state, 897
- disconnect-slave-event-count option
 - mysqld, 1870
- disconnecting
 - from the server, 197
- Disjoint(), 1233
- Disk Data tables (MySQL Cluster) (see MySQL Cluster Disk Data)
- disk full, 4053
- disk option

- ndb_select_all, 2231
- disk performance, 875
- disk-based, 4117
- disk-bound, 4117, 4118
- DiskCheckpointSpeed, 2067
- DiskCheckpointSpeedInRestart, 2068
- DiskIOThreadPool, 2083, 2087
- Diskless, 2058
- diskpagebuffer
 - ndbinfo table, 2290
- DiskPageBufferMemory, 2083, 2087
- disks
 - splitting data across, 878
- DiskSyncSize, 2067
- display size, 1035
- display triggers, 1506
- display width, 1035
- displaying
 - database information, 330
 - information
 - Cardinality, 1488
 - Collation, 1488
 - SHOW, 1471, 1474, 1487, 1489, 1506
 - table status, 1504
- DISTINCT, 208, 931, 1363
 - AVG(), 1203
 - COUNT(), 1204
 - MAX(), 1205
 - MIN(), 1206
 - SUM(), 1206
- DISTINCTROW, 1363
- distributed privileges (MySQL Cluster), 2327
 - and NDB API applications, 2330
- DIV, 1122
- division (/), 1122
- div_precision_increment system variable, 497
- DML, 4118
- DNS, 883
- DO, 1333
- DocBook XML
 - documentation source format, 2
- document id, 4118
- Documentation
 - in Chinese, 3957
 - in Japanese, 3957
 - in Korean, 3957
- Documenters
 - list of, 41
- dont_ignore_systab_0 option (ndb_restore), 2224
- DOUBLE data type, 1039
- DOUBLE PRECISION data type, 1039
- double quote ("), 938
- doublewrite buffer, 4119
- downgrades

- MySQL Cluster, 2022, 2255
- downgrading, 175, 187, 1684
- downloading, 52
- drop, 4119
- DROP ... IF EXISTS
 - and replication, 1941
- DROP DATABASE, 1321
- Drop DB
 - thread command, 894
- DROP EVENT, 1322
- DROP FOREIGN KEY, 1259, 1587
- DROP FUNCTION, 1323, 1466
- DROP INDEX, 1258, 1322, 1638
- DROP LOGFILE GROUP, 1323
 - (see also MySQL Cluster Disk Data)
- DROP NODEGROUP command (MySQL Cluster), 2249
- DROP PREPARE, 1419
- DROP PRIMARY KEY, 1258
- DROP PROCEDURE, 1323
- DROP SCHEMA, 1321
- DROP SERVER, 1323
- DROP TABLE, 1323
 - and MySQL Cluster, 1988
- DROP TABLESPACE, 1324
 - (see also MySQL Cluster Disk Data)
- DROP TRIGGER, 1324
- DROP USER, 1444
- DROP VIEW, 1325
- dropping
 - user, 721, 1444
- dryrun option
 - mysqlhotcopy, 388
- DUAL, 1359
- dump option
 - myisam_ftdump, 343
- dump option (ndb_index_stat), 2212
- dump-date option
 - mysqldump, 315
- dump-slave option
 - mysqldump, 315
- DUMPFIL, 1365
- dumping
 - databases and tables, 307, 386
- dynamic row format, 4119
- dynamic table characteristics, 1696

E

- early adopter, 4120
- edit command
 - mysql, 285
- ego command
 - mysql, 285
- Eiffel Wrapper, 3791

ELT(), 1102
 email lists, 16
 embedded MySQL server library, 3066
 embedded option
 mysql_config, 394
 --enable option prefix, 243
 enable-cleartext-plugin option
 mysql, 278
 mysqladmin, 298
 mysqlslap, 339
 enable-named-pipe option
 mysqld, 442
 enable-pstack option
 mysqld, 442
 ENABLED_LOCAL_INFILE option
 CMake, 154
 ENABLED_PROFILING option
 CMake, 155
 ENABLE_DEBUG_SYNC option
 CMake, 154
 ENABLE_DOWNLOADS option
 CMake, 154
 ENABLE_DTRACE option
 CMake, 154
 ENABLE_GCOV option
 CMake, 154
 ENCODE(), 1187
 ENCRYPT(), 1187
 encryption, 746
 encryption functions, 1183
 end
 thread state, 897
 END, 1419
 EndPoint(), 1227
 engine option
 mysqlslap, 339
 ENGINES
 INFORMATION_SCHEMA table, 2475
 engine_condition_pushdown system variable, 497
 ENTER SINGLE USER MODE command (MySQL Cluster), 2248
 entering
 queries, 198
 enterprise components
 MySQL Enterprise Audit, 757, 3865
 MySQL Enterprise Backup, 3864
 MySQL Enterprise Monitor, 3863
 MySQL Enterprise Security, 729, 736, 3865
 MySQL Thread Pool, 884, 3865
 ENUM
 size, 1072
 ENUM data type, 1044, 1062
 Envelope(), 1226
 environment variable
 AUTHENTICATION_PAM_LOG, 735
 CC, 158, 193
 CFLAGS, 158, 193
 CXX, 158, 193
 CXXFLAGS, 158, 193
 DBI_TRACE, 193, 3857
 DBI_USER, 193
 HOME, 193, 289
 LD_LIBRARY_PATH, 196
 LD_RUN_PATH, 193, 196
 LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN, 193
 LIBMYSQL_PLUGINS, 3171
 MYSQL_DEBUG, 193, 236, 3861
 MYSQL_GROUP_SUFFIX, 193
 MYSQL_HISTFILE, 193, 289
 MYSQL_HOME, 193
 MYSQL_HOST, 193, 240
 MYSQL_PS1, 193
 MYSQL_PWD, 193, 236, 240
 MYSQL_TCP_PORT, 193, 236, 656, 656
 MYSQL_UNIX_PORT, 166, 193, 236, 656, 656
 PATH, 161, 193, 237
 TMPDIR, 166, 193, 236, 4054
 TZ, 193, 4055
 UMASK, 193, 4048
 UMASK_DIR, 193, 4048
 USER, 193, 240
 environment variables, 236, 253, 712
 list of, 192
 equal (=), 1089
 Equals(), 1234
 eq_ref join type
 optimizer, 837
 Errcode, 397
 errno, 397
 Error
 thread command, 894
 error codes, 2941
 ERROR Events (MySQL Cluster), 2264
 error log, 4120
 error logs (MySQL Cluster), 2182
 error messages
 can't find file, 4048
 displaying, 397
 languages, 1013, 1013
 errors
 access denied, 4035
 and replication, 1955
 checking tables for, 790
 common, 4033
 directory checksum, 135
 handling for UDFs, 3849
 in subqueries, 1384
 known, 4064

linking, 3183
 list of, 4035
 lost connection, 4038
 reporting, 20, 20
 sources of information, 3977
 error_count session variable, 498
 ERROR_FOR_DIVISION_BY_ZERO SQL mode, 613
 ER_ABORTING_CONNECTION, 2941
 ER_ACCESS_DENIED_ERROR, 2941
 ER_BAD_FIELD_ERROR, 2941
 ER_BAD_HOST_ERROR, 2941
 ER_BAD_TABLE_ERROR, 2941
 ER_BLOBS_AND_NO_TERMINATED, 2941
 ER_BLOB_CANT_HAVE_DEFAULT, 2941
 ER_BLOB_KEY_WITHOUT_LENGTH, 2941
 ER_BLOB_USED_AS_KEY, 2941
 ER_CANT_DO_THIS_DURING_AN_TRANSACTION, 2941
 ER_CANT_DROP_FIELD_OR_KEY, 2941
 ER_CANT_REMOVE_ALL_FIELDS, 2941
 ER_CANT_USE_OPTION_HERE, 2941
 ER_CHECK_NOT_IMPLEMENTED, 2941
 ER_CHECK_NO_SUCH_TABLE, 2941
 ER_COLLATION_CHARSET_MISMATCH, 2941
 ER_COLUMNACCESS_DENIED_ERROR, 2941
 ER_CONNECT_TO_MASTER, 2941
 ER_CON_COUNT_ERROR, 2941
 ER_DBACCESS_DENIED_ERROR, 2941
 ER_DERIVED_MUST_HAVE_ALIAS, 2941
 ER_DUP_ENTRY, 2941
 ER_DUP_FIELDNAME, 2941
 ER_DUP_KEY, 2941
 ER_DUP_KEYNAME, 2941
 ER_DUP_UNIQUE, 2941
 ER_EMPTY_QUERY, 2941
 ER_FIELD_SPECIFIED_TWICE, 2941
 ER_FORCING_CLOSE, 2941
 ER_GRANT_WRONG_HOST_OR_USER, 2941
 ER_HANDSHAKE_ERROR, 2941
 ER_HOST_IS_BLOCKED, 2941
 ER_HOST_NOT_PRIVILEGED, 2941
 ER_ILLEGAL_GRANT_FOR_TABLE, 2941
 ER_ILLEGAL_REFERENCE, 2941
 ER_INVALID_DEFAULT, 2941
 ER_INVALID_USE_OF_NULL, 2941
 ER_IPSOCK_ERROR, 2941
 ER_KEY_COLUMN_DOES_NOT_EXITS, 2941
 ER_LOCK_DEADLOCK, 2941
 ER_LOCK_WAIT_TIMEOUT, 2941
 ER_MASTER_NET_READ, 2941
 ER_MASTER_NET_WRITE, 2941
 ER_MIX_OF_GROUP_FUNC_AND_FIELDS, 2941
 ER_MULTIPLE_PRI_KEY, 2941
 ER_NET_ERROR_ON_WRITE, 2941
 ER_NET_FCNTL_ERROR, 2941
 ER_NET_PACKETS_OUT_OF_ORDER, 2941
 ER_NET_PACKET_TOO_LARGE, 2941
 ER_NET_READ_ERROR, 2941
 ER_NET_READ_ERROR_FROM_PIPE, 2941
 ER_NET_READ_INTERRUPTED, 2941
 ER_NET_UNCOMPRESS_ERROR, 2941
 ER_NET_WRITE_INTERRUPTED, 2941
 ER_NEW_ABORTING_CONNECTION, 2941
 ER_NONEXISTING_GRANT, 2941
 ER_NONEXISTING_TABLE_GRANT, 2941
 ER_NONUNIQ_TABLE, 2941
 ER_NON_UNIQ_ERROR, 2941
 ER_NOT_ALLOWED_COMMAND, 2941
 ER_NOT_SUPPORTED_AUTH_MODE, 2941
 ER_NOT_SUPPORTED_YET, 2941
 ER_NO_DEFAULT, 2941
 ER_NO_PERMISSION_TO_CREATE_USER, 2941
 ER_NO_REFERENCED_ROW, 2941
 ER_NO_SUCH_INDEX, 2941
 ER_NO_SUCH_TABLE, 2941
 ER_NULL_COLUMN_IN_INDEX, 2941
 ER_OPERAND_COLUMNS, 2941
 ER_OUTOFMEMORY, 2941
 ER_OUT_OF_SORTMEMORY, 2941
 ER_PARSE_ERROR, 2941
 ER_PASSWORD_ANONYMOUS_USER, 2941
 ER_PASSWORD_NOT_ALLOWED, 2941
 ER_PASSWORD_NO_MATCH, 2941
 ER_PRIMARY_CANT_HAVE_NULL, 2941
 ER_READ_ONLY_TRANSACTION, 2941
 ER_REGEX_ERROR, 2941
 ER_REQUIRES_PRIMARY_KEY, 2941
 ER_ROW_IS_REFERENCED, 2941
 ER_SELECT_REDUCED, 2941
 ER_SERVER_SHUTDOWN, 2941
 ER_SPATIAL_CANT_HAVE_NULL, 2941
 ER_SUBQUERY_NO_1_ROW, 2941
 ER_SYNTAX_ERROR, 2941
 ER_TABLEACCESS_DENIED_ERROR, 2941
 ER_TABLENAME_NOT_ALLOWED_HERE, 2941
 ER_TABLE_CANT_HANDLE_AUTO_INCREMENT, 2941
 ER_TABLE_CANT_HANDLE_BLOB, 2941
 ER_TABLE_EXISTS_ERROR, 2941
 ER_TABLE_MUST_HAVE_COLUMNS, 2941
 ER_TOO_BIG_FIELDLENGTH, 2941
 ER_TOO_BIG_ROWSIZE, 2941
 ER_TOO_BIG_SELECT, 2941
 ER_TOO_LONG_IDENT, 2941
 ER_TOO_LONG_KEY, 2941
 ER_TOO_LONG_STRING, 2941
 ER_TOO_MANY_KEYS, 2941
 ER_TOO_MANY_KEY_PARTS, 2941

ER_TOO_MANY_ROWS, 2941
 ER_TOO_MANY_USER_CONNECTIONS, 2941
 ER_UNKNOWN_CHARACTER_SET, 2941
 ER_UNKNOWN_COM_ERROR, 2941
 ER_UNKNOWN_PROCEDURE, 2941
 ER_UNKNOWN_STORAGE_ENGINE, 2941
 ER_UNKNOWN_TABLE, 2941
 ER_UNSUPPORTED_EXTENSION, 2941
 ER_USER_LIMIT_REACHED, 2941
 ER_WARN_DATA_OUT_OF_RANGE, 2941
 ER_WARN_DATA_TRUNCATED, 2941
 ER_WARN_NULL_TO_NOTNULL, 2941
 ER_WARN_TOO_FEW_RECORDS, 2941
 ER_WARN_TOO_MANY_RECORDS, 2941
 ER_WRONG_AUTO_KEY, 2941
 ER_WRONG_COLUMN_NAME, 2941
 ER_WRONG_DB_NAME, 2941
 ER_WRONG_FIELD_SPEC, 2941
 ER_WRONG_FIELD_TERMINATORS, 2941
 ER_WRONG_FIELD_WITH_GROUP, 2941
 ER_WRONG_FK_DEF, 2941
 ER_WRONG_GROUP_FIELD, 2941
 ER_WRONG_KEY_COLUMN, 2941
 ER_WRONG_NAME_FOR_CATALOG, 2941
 ER_WRONG_NAME_FOR_INDEX, 2941
 ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT, 2941
 ER_WRONG_OUTER_JOIN, 2941
 ER_WRONG_PARAMCOUNT_TO_PROCEDURE, 2941
 ER_WRONG_SUM_SELECT, 2941
 ER_WRONG_TABLE_NAME, 2941
 ER_WRONG_TYPE_FOR_VAR, 2941
 ER_WRONG_VALUE_COUNT, 2941
 ER_WRONG_VALUE_COUNT_ON_ROW, 2941
 ER_WRONG_VALUE_FOR_VAR, 2941
 escape (\), 938
 escape sequences
 option files, 246
 strings, 937
 estimating
 query performance, 846
 event
 restrictions, 4073
 event groups, 1413
 event log format (MySQL Cluster), 2261
 event logs (MySQL Cluster), 2258, 2260, 2260
 event scheduler, 2441
 thread states, 906
 Event Scheduler, 2448
 altering events, 1249
 and MySQL privileges, 2454
 and mysqladmin debug, 2453
 and replication, 1947, 1947
 and SHOW PROCESSLIST, 2450
 concepts, 2449
 creating events, 1272
 dropping events, 1322
 enabling and disabling, 2450
 event metadata, 2452
 obtaining status information, 2453
 SQL statements, 2452
 starting and stopping, 2450
 time representation, 2453
 event severity levels (MySQL Cluster), 2260
 event types (MySQL Cluster), 2259, 2261
 event-scheduler option
 mysqld, 443
 events, 2441, 2448
 altering, 1249
 creating, 1272
 dropping, 1322
 metadata, 2452
 status variables, 2456
 EVENTS
 INFORMATION_SCHEMA table, 2455, 2476
 events option
 mysqldump, 316
 events_waits_current table
 performance_schema, 2546
 events_waits_history table
 performance_schema, 2548
 events_waits_history_long table
 performance_schema, 2548
 events_waits_summary_by_instance table
 performance_schema, 2549
 events_waits_summary_by_thread_by_event_name table
 performance_schema, 2549
 events_waits_summary_global_by_event_name table
 performance_schema, 2549
 event_scheduler system variable, 498
 eviction, 4120
 exact-value literals, 1237
 example option
 mysqld_multi, 261
 EXAMPLE storage engine, 1524, 1719
 examples
 compressed tables, 362
 myisamchk output, 352
 queries, 221
 exclude-databases option (ndb_restore), 2226
 exclude-missing-columns option (ndb_restore), 2227
 exclude-tables option (ndb_restore), 2226
 exclusive lock, 4120
 Execute
 thread command, 894
 EXECUTE, 1415, 1418
 execute option

- mysql, 278
- execute option (ndb_mgm), 2195
- ExecuteNonQuery method, 2698
- ExecuteOnComputer, 2035, 2039, 2091
- ExecuteReader method, 2698
- ExecuteScalar method, 2698
- executing
 - thread state, 897
- executing SQL statements from text files, 220, 290
- Execution of init_command
 - thread state, 897
- execution plan, 833
- execution threads (MySQL Cluster), 2079
- EXISTS
 - with subqueries, 1381
- exit command
 - mysql, 285
- EXIT command (MySQL Cluster), 2248
- EXIT SINGLE USER MODE command (MySQL Cluster), 2248
- exit-info option
 - mysqld, 443
- EXP(), 1126
- expire_logs_days system variable, 498
- EXPLAIN, 833, 1519
- EXPLAIN PARTITIONS, 2422, 2423
- EXPLAIN used with partitioned tables, 2422
- explicit default values, 1068
- EXPORT_SET(), 1103
- expression aliases, 1210, 1360
- expression syntax, 959
- expressions
 - extended, 213
- extend-check option
 - myisamchk, 349, 350
- extended option
 - mysqlcheck, 304
- extended-insert option
 - mysqldump, 316
- extensions
 - to standard SQL, 24
- extent, 4121
- ExteriorRing(), 1229
- external locking, 444, 460, 552, 789, 867, 900
- external-locking option
 - mysqld, 444
- external_user session variable, 499
- extra-file option
 - my_print_defaults, 396
- extra-node-info option
 - ndb_desc, 2207
- extra-partition-info option
 - ndb_desc, 2207
- EXTRACT(), 1140

- extracting
 - dates, 210
- ExtractValue(), 1173
- ExtraSendBufferMemory
 - data nodes, 2087

F

- failover
 - in MySQL Cluster replication, 2359
 - Java clients, 1973, 2912
- FALSE, 940, 942
 - testing for, 1090, 1090
- FAQs
 - Connector/ODBC, 2639
- Fast Index Creation, 4121
 - concurrency, 1640
 - crash recovery, 1640
 - examples, 1639
 - implementation, 1640
 - limitations, 1641
 - overview, 1638
- fast option
 - myisamchk, 349
 - mysqlcheck, 305
- fast shutdown, 4121
- features of MySQL, 6
- FEDERATED storage engine, 1524, 1714
- Fetch
 - thread command, 894
- FETCH, 1428
- field
 - changing, 1257
- Field List
 - thread command, 894
- FIELD(), 1103
- fields option
 - ndb_config, 2200
- fields-enclosed-by option
 - mysqldump, 316, 328
- fields-enclosed-by option (ndb_restore), 2223
- fields-escaped-by option
 - mysqldump, 316, 328
- fields-optionally-enclosed-by option
 - mysqldump, 316, 328
- fields-optionally-enclosed-by option (ndb_restore), 2223
- fields-terminated-by option
 - mysqldump, 316, 328
- fields-terminated-by option (ndb_restore), 2223, 2224

FILE, 1105

- file format, 1651, 4122
 - Antelope, 1650
 - Barracuda, 1642
 - downgrading, 1656

- identifying, 1656
- file format management
 - downgrading, 1684
 - enabling new file formats, 1678
- file per table, 1679
- file-per-table, 4122
- files
 - binary log, 634
 - error messages, 1013
 - general query log, 633
 - log, 648
 - my.cnf, 1937
 - not found message, 4048
 - permissions, 4048
 - repairing, 350
 - script, 220
 - size limits, 4085
 - slow query log, 647
 - text, 290, 325
 - tmp, 166
- FILES
 - INFORMATION_SCHEMA table, 2505
- filesort optimization, 928
- FileSystemPath, 2042
- FileSystemPathDataFiles, 2084
- FileSystemPathDD, 2084
- FileSystemPathUndoFiles, 2085
- file_instances table
 - performance_schema, 2544
- file_summary_by_event_name table
 - performance_schema, 2550
- file_summary_by_instance table
 - performance_schema, 2550
- fill factor, 4123
- FIND_IN_SET(), 1103
- Finished reading one binlog; switching to next binlog
 - thread state, 904
- firewalls (software)
 - and MySQL Cluster, 2304, 2305
- first-slave option
 - mysqldump, 316
- fix-db-names option
 - mysqlcheck, 305
- fix-table-names option
 - mysqlcheck, 305
- FIXED data type, 1038
- fixed row format, 4123
- fixed-point arithmetic, 1237
- FLOAT data type, 1038, 1039, 1039
- floating-point number, 1039
- floating-point values
 - and replication, 1948
- floats, 940
- FLOOR(), 1126
- FLUSH, 1513
 - and replication, 1949
- flush, 4123
- flush list, 4124
- flush list mutex, 1678
- flush option
 - mysqld, 444
- flush system variable, 499
- flush tables, 296
- flush-logs option
 - mysqldump, 316
- flush-privileges option
 - mysqldump, 316
- Flushing tables
 - thread state, 897
- flushlog option
 - mysqlhotcopy, 388
- flush_time system variable, 500
- FOR UPDATE, 1363
- FORCE INDEX, 1373, 4063
- FORCE KEY, 1373
- force option
 - myisamchk, 349, 350
 - myisampack, 362
 - mysql, 278
 - mysqladmin, 298
 - mysqlcheck, 305
 - mysqldump, 316
 - mysqlimport, 328
 - mysql_convert_table_format, 390
 - mysql_install_db, 266
 - mysql_upgrade, 272
- force-if-open option
 - mysqlbinlog, 375
- force-read option
 - mysqlbinlog, 375
- foreign key, 4124
 - constraint, 34, 1584
 - deleting, 1259, 1587
- FOREIGN KEY constraint, 4125
- FOREIGN KEY constraints
 - and fast index creation, 1641
 - and TRUNCATE TABLE, 1680
- foreign keys, 31, 224, 1259, 2668
- foreign_key_checks session variable, 500
- FORMAT(), 1103
- Forums, 19
- FOUND_ROWS(), 1193
- FragmentLogFileSize, 2052
- FreeBSD troubleshooting, 159
- freeing items
 - thread state, 897
- frequently-asked questions about MySQL Cluster, 3945
- FROM, 1360

FROM_DAYS(), 1140
FROM_UNIXTIME(), 1140
FTS, 4126
ft_boolean_syntax system variable, 500
ft_max_word_len myisamchk variable, 347
ft_max_word_len system variable, 501
ft_min_word_len myisamchk variable, 347
ft_min_word_len system variable, 501
ft_query_expansion_limit system variable, 502
ft_stopword_file myisamchk variable, 347
ft_stopword_file system variable, 502
full backup, 4126
full disk, 4053
full table scan, 4126
full table scans
 avoiding, 801
full-text parser plugins, 3798
full-text search, 1153, 4126
FULLTEXT, 1153
fulltext
 stopword list, 1165
FULLTEXT index, 4126
FULLTEXT initialization
 thread state, 897
fulltext join type
 optimizer, 838
function
 creating, 1465
 deleting, 1466
function names
 parsing, 950
 resolving ambiguity, 950
functions, 1076
 and replication, 1949
 arithmetic, 1182
 bit, 1182
 C API, 3077
 C prepared statement API, 3139, 3140
 cast, 1168
 control flow, 1097
 date and time, 1131
 encryption, 1183
 for SELECT and WHERE clauses, 1076
 GROUP BY, 1203
 grouping, 1088
 information, 1190
 mathematical, 1123
 miscellaneous, 1198
 native
 adding, 3852
 new, 3841
 stored, 2443
 string, 1099
 string comparison, 1111

 user-defined, 1465, 1466, 3841
 adding, 3842
fuzzy checkpointing, 4127

G

GA, 4127
gap, 4127
gap event, 2345
gap lock, 4127
 InnoDB, 1561, 1597, 1602, 1603
gb2312, gbk, 3957
gci option
 ndb_select_all, 2231
GCP Stop errors (MySQL Cluster), 2087
gdb
 using, 3856
gdb option
 mysqld, 444
general information, 1
General Public License, 5
general query log, 633, 4128
general-log option
 mysqld, 445
general_log system variable, 502
general_log_file system variable, 503
geographic feature, 1211
GeomCollFromText(), 1220
GeomCollFromWKB(), 1221
geometry, 1212
GEOMETRY data type, 1220
GEOMETRYCOLLECTION data type, 1220
GeometryCollection(), 1222
GeometryCollectionFromText(), 1220
GeometryCollectionFromWKB(), 1221
GeometryFromText(), 1220
GeometryFromWKB(), 1221
GeometryN(), 1231
GeometryType(), 1226
GeomFromText(), 1220, 1225
GeomFromWKB(), 1221, 1225
geospatial feature, 1212
getting MySQL, 52
GET_FORMAT(), 1141
GET_LOCK(), 1199
GIS, 1211, 1211
Glassfish application server, 2969
GLength(), 1228, 1229
global privileges, 1445, 1455
globalization, 963
GLOBAL_STATUS
 INFORMATION_SCHEMA table, 2479
global_transaction, 4128
GLOBAL_VARIABLES

INFORMATION_SCHEMA table, 2479
 go command
 mysql, 285
 Google Test, 154
 got handler lock
 thread state, 902
 got old table
 thread state, 902
 GRANT, 1445
 GRANT statement, 718
 grant table distribution (MySQL Cluster), 2327
 grant tables
 re-creating, 166
 sorting, 707, 709
 structure, 698
 granting
 privileges, 1445
 GRANTS, 1487
 greater than (>), 1090
 greater than or equal (>=), 1090
 greatest timestamp wins (conflict resolution), 2374
 greatest timestamp, delete wins (conflict resolution), 2375
 GREATEST(), 1092
 GROUP BY, 929
 aliases in, 1210
 extensions to standard SQL, 1209, 1361
 GROUP BY functions, 1203
 group commit, 1672, 4128
 grouping
 expressions, 1088
 GROUP_CONCAT(), 1205
 group_concat_max_len system variable, 503

H

HANDLER, 1333
 Handlers, 1431
 handling
 errors, 3849
 hash index, 4129
 hash indexes, 815
 hash partitioning, 2401
 hash partitions
 managing, 2420
 splitting and merging, 2420
 have_compress system variable, 504
 have_crypt system variable, 504
 have_csv system variable, 504
 have_dynamic_loading system variable, 504
 HAVE_EMBEDDED_PRIVILEGE_CONTROL option
 CMake, 156
 have_geometry system variable, 504
 have_innodb system variable, 504

have_openssl system variable, 504
 have_partitioning system variable, 504
 have_profiling system variable, 504
 have_query_cache system variable, 505
 have_rtree_keys system variable, 505
 have_ssl system variable, 505
 have_symlink system variable, 505
 HAVING, 1361
 HDD, 4129
 header option
 ndb_select_all, 2231
 header_file option
 comp_err, 265
 HEAP storage engine, 1524, 1699
 heartbeat, 4129
 HeartbeatIntervalDbApi, 2062
 HeartbeatIntervalDbDb, 2062
 HeartbeatOrder, 2063
 HeartbeatThreadPriority, 2037, 2092
 help command
 mysql, 284
 HELP command (MySQL Cluster), 2246
 help option
 comp_err, 264
 myisamchk, 347
 myisampack, 361
 myisam_ftdump, 343
 mysql, 276
 mysqlaccess, 368
 mysqladmin, 297
 mysqlbinlog, 372
 mysqlcheck, 303
 mysqld, 434
 mysqldump, 313
 mysqldumpslow, 385
 mysqld_multi, 261
 mysqld_safe, 256
 mysqlhotcopy, 388
 mysqlimport, 327
 MySQLInstallerConsole, 77
 mysqlshow, 332
 mysqlslap, 337
 mysql_convert_table_format, 390
 mysql_find_rows, 391
 mysql_plugin, 268
 mysql_setpermission, 392
 mysql_upgrade, 271
 mysql_waitpid, 392
 my_print_defaults, 395
 perror, 397
 resolveip, 398
 resolve_stack_dump, 396
 HELP option
 myisamchk, 347

help option (MySQL Cluster programs), 2241
HELP statement, 1520
hex option (ndb_restore), 2224
HEX(), 1104, 1126
hex-blob option
 mysqldump, 316
hexadecimal literals, 942
hexdump option
 mysqlbinlog, 375
high-water mark, 4129
HIGH_NOT_PRECEDENCE SQL mode, 613
HIGH_PRIORITY, 1363
hints, 26
 index, 1360, 1373
history list, 4129
history of MySQL, 9
HOME environment variable, 193, 289
host name
 default, 237
host name caching, 883
host name resolution, 883
host names
 in account names, 703
 in default accounts, 171
host option, 239
 mysql, 278
 mysqlaccess, 368
 mysqladmin, 298
 mysqlbinlog, 375
 mysqlcheck, 305
 mysqldump, 316
 mysqlhotcopy, 388
 mysqlimport, 328
 mysqlshow, 333
 mysqslap, 339
 mysql_convert_table_format, 390
 mysql_setpermission, 392
 ndb_config, 2198
host table, 710
 sorting, 709
Host*Scild* parameters, 2101
host.frm
 problems finding, 162
HostName, 2035, 2040, 2091
HostName (MySQL Cluster), 2303
hostname system variable, 505
HostName1, 2095, 2099, 2102
HostName2, 2096, 2099, 2102
hot, 4130
hot backup, 4130
HOUR(), 1142
howto option
 mysqlaccess, 368
html option

mysql, 278

I

i-am-a-dummy option
 mysql, 281
ib-file set, 1652, 4130
ibbackup command, 4131
ibbackup_logfile, 4131
ibdata file, 4132
ib_logfile, 4130
icc
 and MySQL Cluster support>, 3854
 MySQL builds, 56
Id, 2034, 2039, 2090
ID
 unique, 3173
id option
 ndb_config, 2199
identifiers, 943
 case sensitivity, 946
 quoting, 943
identity session variable, 505
IF, 1424
IF(), 1097
IFNULL(), 1098
IGNORE
 with partitioned tables, 1338
IGNORE INDEX, 1373
IGNORE KEY, 1373
ignore option
 mysqlimport, 328
ignore-builtin-innodb option
 mysqld, 1546
ignore-lines option
 mysqlimport, 328
ignore-spaces option
 mysql, 278
ignore-table option
 mysqldump, 317
ignore_builtin_innodb system variable, 1547
IGNORE_SPACE SQL mode, 613
ilist, 4133
implicit default values, 1068
implicit row lock, 4133
IMPORT TABLESPACE, 1259, 1539
importing
 data, 290, 325
IN, 1092, 1379
in-memory database, 4133
include option
 mysql_config, 394
include-databases option (ndb_restore), 2225
include-master-host-port option

- mysqldump, 317
- include-tables option (ndb_restore), 2225
- increasing with replication
 - speed, 1839
- incremental backup, 4134
- incremental recovery, 786
 - using MySQL Cluster replication, 2365
- index, 4134
 - deleting, 1258, 1322
 - rebuilding, 190
- index cache, 4135
- INDEX DIRECTORY
 - and replication, 1947
- index dives (for statistics estimation), 1681
- index hint, 4135
- index hints, 1360, 1373
- index join type
 - optimizer, 839
- index prefix, 4135
- index-record lock
 - InnoDB, 1561, 1597, 1602, 1603
- indexes, 1277
 - and BLOB columns, 811, 1295
 - and IS NULL, 816
 - and LIKE, 815
 - and NULL values, 1295
 - and TEXT columns, 811, 1295
 - assigning to key cache, 1511
 - block size, 508
 - columns, 811
 - creating and dropping, 1640
 - leftmost prefix of, 813
 - multi-column, 812
 - multiple-part, 1277
 - names, 943
 - primary (clustered) and secondary, 1640
 - use of, 810
- Indexes/Keys dialog, 2667
- IndexMemory, 2044
- index_merge join type
 - optimizer, 838
- index_subquery join type
 - optimizer, 838
- INET_ATON(), 1199
- INET_NTOA(), 1200
- infimum record, 4135
- INFO Events (MySQL Cluster), 2264
- information functions, 1190
- information option
 - myisamchk, 349
- information schema tables, 1658
 - INNODB_CMP, 1658
 - INNODB_CMPMEM, 1658
 - INNODB_CMPMEM_RESET, 1658
 - INNODB_CMP_RESET, 1658
 - INNODB_LOCKS, 1660
 - INNODB_LOCK_WAITS, 1660
 - INNODB_TRX, 1660
- INFORMATION_SCHEMA, 2469, 4136
 - and security issues, 2308
 - collation and searching, 986
- INFORMATION_SCHEMA plugins, 3799
- INFORMATION_SCHEMA.ENGINES table
 - and MySQL Cluster, 2280
- INFORMATION_SCHEMA.GLOBAL_STATUS table
 - and MySQL Cluster, 2282
- INFORMATION_SCHEMA.GLOBAL_VARIABLES table
 - and MySQL Cluster, 2280
- init
 - thread state, 897
- Init DB
 - thread command, 894
- init-command option
 - mysql, 278
- init-file option
 - mysqld, 445
- InitFragmentLogFiles, 2052
- initial option (ndbd), 2178
- initial option (ndbmtd), 2178
- initial option (ndb_mgmd), 2190
- initial-start option (ndbd), 2179
- initial-start option (ndbmtd), 2179
- Initialized
 - thread state, 907
- InitialLogFileGroup, 2085
- InitialNoOfOpenFiles, 2053
- InitialTablespace, 2086
- init_connect system variable, 505
- init_file system variable, 506
- init_slave system variable, 1885
- INNER JOIN, 1366
- innobackup command, 4136
- innochecksum, 234, 341
- InnoDB, 1529, 4136
 - adaptive hash index, 1611
 - and application feature requirements, 1985
 - applications supported, 1984
 - auto-increment columns, 1578
 - autocommit mode, 1577, 1607
 - availability, 1982
 - backups, 1592
 - buffer pool, 848
 - checkpoints, 1595
 - clustered index, 1610
 - compared to MySQL Cluster, 1982, 1982, 1984, 1985
 - configuration parameters, 1542
 - configuring data files and memory allocation, 1534
 - considerations as default storage engine, 1531

- consistent reads, 1599
- crash recovery, 1594
- data files, 1591
- deadlock detection, 1607
- disk I/O, 1613
- file space management, 1614
- file-per-table setting, 1538
- foreign key constraints, 1584
- gap lock, 1561, 1597, 1602, 1603
- index-record lock, 1561, 1597, 1602, 1603
- indexes, 1610
- insert buffering, 1611
- limits and restrictions, 1632
- lock modes, 1597
- locking, 1596
- locking reads, 1601
- log files, 1591
- migrating tables, 1596
- Monitors, 1594, 1615, 1620, 1630, 1631
- multi-versioning, 1609
- next-key lock, 1561, 1597, 1602, 1603
- NFS, 1535, 1632
- page size, 1610, 1633
- raw devices, 1540
- record-level locks, 1561, 1597, 1602, 1603
- replication, 1589
- row structure, 1612
- secondary index, 1610
- semi-consistent read, 1561
- Solaris 10 x86_64 issues, 135
- system tablespace setup, 1541
- system variables, 1542
- tables, 1576, 1610
 - converting from other storage engines, 1578
- transaction model, 1596
- troubleshooting, 1630
 - data dictionary problems, 1631
 - deadlocks, 1607
 - defragmenting tables, 1615
 - fast index creation, 1641
 - I/O problems, 1542
 - InnoDB error codes, 1616
 - OS error codes, 1617
 - performance problems, 823
 - recovery problems, 1594
 - SQL errors, 1615
- InnoDB Hot Backup, 4137
- innodb option
 - mysqld, 1546
- InnoDB parameters, deprecated, 1690
 - innodb_file_io_threads, 1671
- InnoDB parameters, new, 1688
 - innodb_adaptive_flushing, 1673
 - innodb_change_buffering, 1668
- innodb_file_format, 1678
- innodb_file_format_check, 1653
- innodb_io_capacity, 1672
- innodb_large_prefix, 1560
- innodb_read_ahead_threshold, 1670
- innodb_read_io_threads, 1671
- innodb_spin_wait_delay, 1673
- innodb_stats_sample_pages, 1681
- innodb_strict_mode, 1681
- innodb_use_sys_malloc, 1667
- innodb_write_io_threads, 1671
- InnoDB parameters, with new defaults, 1690
 - innodb_additional_mem_pool_size, 1690
 - innodb_buffer_pool_size, 1690
 - innodb_change_buffering, 1690
 - innodb_file_format_check, 1690
 - innodb_log_buffer_size, 1690
 - innodb_max_dirty_pages_pct, 1673, 1690
 - innodb_sync_spin_loops, 1690
 - innodb_thread_concurrency, 1690
- InnoDB storage engine, 1524, 1529
 - compatibility, 1635
 - downloading, 1637
 - features, 1635
 - installing, 1637
 - restrictions, 1638
- InnoDB tables, 29
- innodb-status-file option
 - mysqld, 1546
- innodb_system variable
 - innodb_rollback_segments, 1570
- innodb_adaptive_flushing, 1673
- innodb_adaptive_flushing system variable, 1547
- innodb_adaptive_hash_index, 1669
 - and innodb_thread_concurrency, 1669
 - dynamically changing, 1680
- innodb_adaptive_hash_index system variable, 1547
- innodb_additional_mem_pool_size
 - and innodb_use_sys_malloc, 1667
- innodb_additional_mem_pool_size system variable, 1548
- innodb_autoextend_increment system variable, 1548
- innodb_autoinc_lock_mode, 4137
- innodb_autoinc_lock_mode system variable, 1548
- INNODB_BUFFER_PAGE table, 2502
- INNODB_BUFFER_PAGE_LRU table, 2503
- innodb_buffer_pool_instances system variable, 1549
- innodb_buffer_pool_size system variable, 1549
- INNODB_BUFFER_POOL_STATS table, 2503
- innodb_change_buffering, 1668
- innodb_change_buffering system variable, 1550
- innodb_checksums system variable, 1551
- INNODB_CMP
 - INFORMATION_SCHEMA table, 2498
- INNODB_CMPMEM

INFORMATION_SCHEMA table, 2498
 INNODB_CMPMEM_RESET
 INFORMATION_SCHEMA table, 2498
 INNODB_CMP_RESET
 INFORMATION_SCHEMA table, 2498
 innodb_commit_concurrency system variable, 1551
 innodb_concurrency_tickets, 1669
 innodb_concurrency_tickets system variable, 1552
 innodb_data_file_path system variable, 1552
 innodb_data_home_dir system variable, 1552
 innodb_doublewrite system variable, 1553
 innodb_fast_shutdown system variable, 1553
 innodb_file_format, 1651, 4137
 Antelope, 1650
 Barracuda, 1642
 downgrading, 1684
 enabling new file formats, 1678
 identifying, 1656
 innodb_file_format system variable, 1554
 innodb_file_format_check, 1653
 innodb_file_format_check system variable, 1554
 innodb_file_format_max system variable, 1555
 innodb_file_io_threads, 1671
 innodb_file_per_table, 1642, 4138
 dynamically changing, 1679
 innodb_file_per_table system variable, 1555
 innodb_flush_log_at_trx_commit system variable, 1556
 innodb_flush_method system variable, 1557
 innodb_force_load_corrupted system variable, 1558
 innodb_force_recovery system variable, 1558
 innodb_io_capacity, 1672
 innodb_io_capacity system variable, 1559
 innodb_large_prefix system variable, 1560
 INNODB_LOCKS
 INFORMATION_SCHEMA table, 2500
 innodb_locks_unsafe_for_binlog system variable, 1561
 INNODB_LOCK_WAITS
 INFORMATION_SCHEMA table, 2501
 innodb_lock_wait_timeout, 4138
 dynamically changing, 1680
 innodb_lock_wait_timeout system variable, 1560
 innodb_log_buffer_size system variable, 1563
 innodb_log_files_in_group system variable, 1564
 innodb_log_file_size system variable, 1564
 innodb_log_group_home_dir system variable, 1564
 innodb_max_dirty_pages_pct, 1673
 innodb_max_dirty_pages_pct system variable, 1565
 innodb_max_purge_lag system variable, 1565
 innodb_mirrored_log_groups system variable, 1566
 innodb_old_blocks_pct, 1674
 innodb_old_blocks_pct system variable, 1566
 innodb_old_blocks_time, 1674
 innodb_old_blocks_time system variable, 1566
 innodb_open_files system variable, 1567
 innodb_purge_batch_size system variable, 1567
 innodb_purge_threads system variable, 1567
 innodb_read_ahead_threshold, 1670
 innodb_read_ahead_threshold system variable, 1568
 innodb_read_io_threads, 1671
 innodb_read_io_threads system variable, 1568
 innodb_replication_delay system variable, 1569
 innodb_rollback_on_timeout system variable, 1569
 innodb_rollback_segments system variable, 1570
 innodb_spin_wait_delay, 1673
 innodb_spin_wait_delay system variable, 1570
 innodb_stats_method system variable, 1570
 innodb_stats_on_metadata
 dynamically changing, 1679
 innodb_stats_on_metadata system variable, 1571
 innodb_stats_sample_pages, 1681
 innodb_stats_sample_pages system variable, 1571
 innodb_strict_mode, 1681, 4138
 innodb_strict_mode system variable, 1572
 innodb_support_xa system variable, 1572
 innodb_sync_spin_loops system variable, 1573
 innodb_table_locks system variable, 1573
 innodb_thread_concurrency, 1669
 innodb_thread_concurrency system variable, 1573
 innodb_thread_sleep_delay, 1669
 innodb_thread_sleep_delay system variable, 1574
 INNODB_TRX
 INFORMATION_SCHEMA table, 2499
 innodb_use_native_aio system variable, 1574
 innodb_use_sys_malloc, 1667
 and innodb_thread_concurrency, 1669
 innodb_use_sys_malloc system variable, 1575
 innodb_version system variable, 1575
 innodb_write_io_threads, 1671
 innodb_write_io_threads system variable, 1575
 INSERT, 803, 1334
 insert, 4139
 thread state, 903
 INSERT ... SELECT, 1338
 insert buffer, 4139
 insert buffering, 1611, 4139
 disabling, 1668
 INSERT DELAYED, 1339, 1339
 INSERT statement
 grant privileges, 719
 INSERT(), 1104
 insert-ignore option
 mysqldump, 317
 insertable views
 insertable, 2459
 inserting
 speed of, 803
 inserts
 concurrent, 863, 866

- insert_id session variable, 506
- install option
 - mysqld, 445
- install option (ndbd), 2180
- install option (ndbmtd), 2180
- install option (ndb_mgmd), 2193
- INSTALL_PLUGIN, 1466
- install-manual option
 - mysqld, 445
- Installation, 76
- installation layouts, 55
- installation overview, 140
- installing
 - binary distribution, 56
 - installing on HP-UX, 138
 - Linux RPM packages, 127
 - Mac OS X PKG packages, 117
 - overview, 46
 - Perl, 194
 - Perl on Windows, 195
 - Solaris PKG packages, 135
 - source distribution, 140
 - user-defined functions, 3850
- installing MySQL Cluster, 1997
 - Linux, 2000
 - Linux binary release, 2000
 - Linux RPM, 2002
 - Linux source release, 2004
 - Windows, 2005
 - Windows binary release, 2006
 - Windows source, 2009
- installing plugins, 618, 1466
- INSTALL_BINDIR option
 - CMake, 151
- INSTALL_DOCDIR option
 - CMake, 151
- INSTALL_DOCREADMEDIR option
 - CMake, 151
- INSTALL_INCLUDEDIR option
 - CMake, 151
- INSTALL_INFODIR option
 - CMake, 151
- INSTALL_LAYOUT option
 - CMake, 151
- INSTALL_LIBDIR option
 - CMake, 152
- INSTALL_MANDIR option
 - CMake, 152
- INSTALL_MYSQLSHAREDIR option
 - CMake, 152
- INSTALL_MYSQLTESTDIR option
 - CMake, 152
- INSTALL_PLUGINDIR option
 - CMake, 152
- INSTALL_SBINDIR option
 - CMake, 152
- INSTALL_SCRIPTDIR option
 - CMake, 152
- INSTALL_SHAREDIR option
 - CMake, 152
- INSTALL_SQLBENCHDIR option
 - CMake, 152
- INSTALL_SUPPORTFILESDIR option
 - CMake, 152

- instance, 4140
- INSTR(), 1104
- instrumentation, 4140
- INT data type, 1037
- integer arithmetic, 1237
- INTEGER data type, 1037
- integers, 940
- intention lock, 4140
- interactive option (ndb_mgmd), 2189
- interactive_timeout system variable, 506
- interceptors, 2763, 2784
- InteriorRingN(), 1229
- internal locking, 862
- internal memory allocator
 - disabling, 1667
- internals, 3793
- internationalization, 963
- Internet Relay Chat, 19
- Intersection(), 1232
- Intersects(), 1234
- INTERVAL(), 1093
- INTO
 - SELECT, 1364
- introducer
 - string literal, 938, 970
- invalid data
 - constraint, 34
- invalidating query cache entries
 - thread state, 903
- inverted index, 4140
- in_file option
 - comp_err, 265
- IOPS, 4141
- IP addresses
 - in account names, 703
 - in default accounts, 171
- IPv6 addresses
 - for JDBC URLs, 2911
 - in account names, 703
 - in default accounts, 171
- IPv6 connections, 172
- IRC, 19
- IS boolean_value, 1090
- IS NOT boolean_value, 1090

IS NOT NULL, 1091
IS NULL, 915, 1091
 and indexes, 816
isamlog, 234, 360
IsClosed(), 1229
IsEmpty(), 1227
ISNULL(), 1093
ISOLATION LEVEL, 1400
isolation level, 1596, 4141, 4483
IsRing(), 1228
IsSimple(), 1227
IS_FREE_LOCK(), 1200
IS_USED_LOCK(), 1200
ITERATE, 1425
iterations option
 mysqslap, 339

J

J2EE
 connection pooling, 2954
 load balancing, 2958
Japanese character sets
 conversion, 3957
Japanese, Korean, Chinese character sets
 frequently asked questions, 3957
JBoss application server, 2964
JDBC, 2565
 and MySQL data types, 2933
 background information for Connector/J, 2945
 character sets, 2935
 CLASSPATH, 2906
 code examples, 2910
 compatibility, 2930
 configuration properties, 2911
 driver for MySQL, 2904
 SQLState codes, 2941
 troubleshooting, 2907, 2975
 versions supported, 2904
jdbc:mysql:loadbalance://, 1973
join, 4142
 nested-loop algorithm, 920
JOIN, 1366
join algorithm
 Block Nested-Loop, 917
 Nested-Loop, 917
join option
 mysampack, 362
join type, 4335
 ALL, 839
 const, 837
 eq_ref, 837
 fulltext, 838
 index, 839

 index_merge, 838
 index_subquery, 838
 range, 838
 ref, 837
 ref_or_null, 838
 system, 837
 unique_subquery, 838
join_buffer_size system variable, 507

K

keepold option
 mysqlhotcopy, 388
keep_files_on_create system variable, 507
Key cache
 MyISAM, 851
key cache
 assigning indexes to, 1511
key partitioning, 2404
key partitions
 managing, 2420
 splitting and merging, 2420
key space
 MyISAM, 1695
key-value store, 816
keys, 811
 foreign, 31, 224
 multi-column, 812
 searching on two, 226
keys option
 mysqlshow, 333
keys-used option
 myisamchk, 350
keywords, 953
KEY_BLOCK_SIZE, 1642, 1648, 4142
key_buffer_size myisamchk variable, 347
key_buffer_size system variable, 508
key_cache_age_threshold system variable, 509
key_cache_block_size system variable, 509
key_cache_division_limit system variable, 510
KEY_COLUMN_USAGE
 INFORMATION_SCHEMA table, 2480
Kill
 thread command, 894
KILL, 1516
Killed
 thread state, 897
Killing slave
 thread state, 906
known errors, 4064
Korean, 3957
Korean, Chinese, Japanese character sets
 frequently asked questions, 3957

L

- labels
 - stored program block, 1420
- language option
 - mysqld, 446
- language support
 - error messages, 1013
- language system variable, 510
- large page support, 881
- large tables
 - and MySQL Cluster, 1300
 - creating in NDB, 1300
- large-pages option
 - mysqld, 446
- large_files_support system variable, 511
- large_pages system variable, 511
- large_page_size system variable, 511
- last row
 - unique ID, 3173
- LAST_DAY(), 1142
- last_insert_id session variable, 511
- LAST_INSERT_ID(), 1194, 1337
 - and replication, 1938
 - and stored routines, 2445
 - and triggers, 2445
 - locking considerations, 31
- latch, 4142
- layout of installation, 55
- lc-messages option
 - mysqld, 447
- lc-messages-dir option
 - mysqld, 447
- LCASE(), 1104
- lc_messages system variable, 512
- lc_messages_dir system variable, 512
- lc_time_names system variable, 512
- ldata option
 - mysql_install_db, 266
- LDML syntax, 1024
- LD_LIBRARY_PATH environment variable, 196
- LD_RUN_PATH environment variable, 193, 196
- LEAST(), 1093
- LEAVE, 1425
- ledir option
 - mysqld_safe, 256
- LEFT JOIN, 916, 1366
- LEFT OUTER JOIN, 1366
- LEFT(), 1104
- leftmost prefix of indexes, 813
- legal names, 943
- length option
 - myisam_ftdump, 343
- LENGTH(), 1104
- less than (<), 1090
- less than or equal (<=), 1090
- libmysqlclient.a, 3004
- libmysqlcppconn-static.a, 3004
- libmysqlcppconn.so, 3004
- libmysqld, 3066
 - options, 3067
- libmysqld-libs option
 - mysql_config, 394
- LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN environment variable, 193
- LIBMYSQL_PLUGINS environment variable, 3171
- library
 - mysqlclient, 2565
 - mysqld, 2565
- libs option
 - mysql_config, 394
- libs_r option
 - mysql_config, 394
- license system variable, 513
- LIKE, 1111
 - and indexes, 815
 - and wildcards, 815
- LIMIT, 801, 1193, 1362
 - and replication, 1951
- limitations
 - MySQL Limitations, 4084
 - replication, 1937
- limitations of MySQL Cluster, 1986
- limits
 - file-size, 4085
 - InnoDB, 1632
 - MySQL Limits, limits in MySQL, 4084
- line-numbers option
 - mysql, 278
- linear hash partitioning, 2403
- linear key partitioning, 2406
- linefeed (\n), 938, 1348
- LineFromText(), 1220
- LineFromWKB(), 1221
- lines-terminated-by option
 - mysqldump, 317, 328
- LINestring data type, 1220
- LineString(), 1222
- LineStringFromText(), 1220
- LineStringFromWKB(), 1221
- linking, 3183
 - errors, 3183
 - problems, 3183
- links
 - symbolic, 876
- LINQ, 2731
- list, 4143
- list partitioning, 2391, 2393

- list partitions
 - adding and dropping, 2415
 - managing, 2415
- literals, 937
- LN(), 1126
- load balancing
 - with Connector/J, 2940, 2958
- LOAD DATA
 - and replication, 1951
- LOAD DATA INFILE, 1342, 2766, 4059
- load emulation, 334
- LOAD INDEX INTO CACHE
 - and partitioning, 2430
- LOAD XML, 1351
- loading
 - tables, 204
- LOAD_FILE(), 1105
- local option
 - mysqlimport, 328
- local-infile option
 - mysql, 278
- local-load option
 - mysqlbinlog, 375
- localhost
 - special treatment of, 238
- localization, 963
- LOCALTIME, 1142
- LOCALTIMESTAMP, 1142
- local_infile system variable, 513
- LOCATE(), 1105
- lock, 4143
- lock escalation, 4143
- LOCK IN SHARE MODE, 1363
- lock mode, 4143
- Lock Monitor
 - InnoDB, 1620
- lock option
 - ndb_select_all, 2230
- LOCK TABLES, 1394
- lock wait timeout, 1680
- lock-all-tables option
 - mysqldump, 317
- lock-tables option
 - mysqldump, 317
 - mysqlimport, 328
- Locked
 - thread state, 897
- locked_in_memory system variable, 514
- LockExecuteThreadToCPU, 2076
- locking, 868, 4144
 - external, 444, 460, 552, 789, 867, 900
 - information schema, 1658, 1660, 1664
 - internal, 862
 - metadata, 866
 - row-level, 31, 862
 - table-level, 862
- locking methods, 862
- locking read, 4144
- LockMaintThreadsToCPU, 2077
- LockPagesInMainMemory, 2057
- lock_wait_timeout system variable, 513
- log, 4144
 - log buf mutex, 1678
 - log buffer, 4145
 - log file, 4145
 - log files
 - maintaining, 648
 - log files (MySQL Cluster), 2182
 - ndbmysqld, 2185
 - log group, 4145
 - log option
 - mysqld, 447
 - mysqld_multi, 261
 - log sys mutex, 1678
 - log system variable, 514
- LOG(), 1126
- log-bin option
 - mysqld, 1892
- log-bin-index option
 - mysqld, 1892
- log-bin-trust-function-creators option
 - mysqld, 1892
- log-bin-use-v1-row-events option
 - mysqld, 1893
- log-error option
 - mysqld, 448
 - mysqldump, 317
 - mysqld_safe, 256
- log-isam option
 - mysqld, 448
- log-long-format option
 - mysqld, 448
- log-name option (ndb_mgmd), 2190
- log-output option
 - mysqld, 449
- log-queries-not-using-indexes option
 - mysqld, 449
- log-short-format option
 - mysqld, 450, 1894
- log-slave-updates option
 - mysqld, 1870
- log-slow-admin-statements option
 - mysqld, 450
- log-slow-queries option
 - mysqld, 450
- log-slow-slave-statements option
 - mysqld, 1871
- log-tc option

- mysqld, 450
- log-tc-size option
 - mysqld, 451
- log-warnings option
 - mysqld, 451, 1871
- LOG10(), 1127
- LOG2(), 1127
- logbuffers
 - ndbinfo table, 2291
- LogDestination, 2036
- logging
 - passwords, 681
- logging commands (MySQL Cluster), 2260
- logging slow query
 - thread state, 898
- logical, 4145
- logical backup, 4145
- logical operators, 1094
- login
 - thread state, 898
- LogLevelCheckpoint, 2072
- LogLevelCongestion, 2072
- LogLevelConnection, 2072
- LogLevelError, 2072
- LogLevelInfo, 2073
- LogLevelNodeRestart, 2072
- LogLevelShutdown, 2071
- LogLevelStartup, 2071
- LogLevelStatistic, 2071
- logs
 - and TIMESTAMP, 181
 - flushing, 629
 - server, 629
- logspaces
 - ndbinfo table, 2292
- log_bin system variable, 514
- log_bin_trust_function_creators system variable, 514
- log_bin_use_v1_row_events system variable, 1901
- log_error system variable, 515
- log_output system variable, 515
- log_queries_not_using_indexes system variable, 515
- log_slave_updates system variable, 1897
- log_slow_queries system variable, 516
- log_warnings system variable, 516
- Long Data
 - thread command, 894
- LONG data type, 1061
- LONGBLOB data type, 1043
- LongMessageBuffer, 2050
- LONGTEXT data type, 1044
- long_query_time system variable, 517
- LOOP, 1425
 - labels, 1420
- loops option

- ndb_show_tables, 2234
- loops option (ndbinfo_select_all), 2184
- loops option (ndb_index_stat), 2214
- loose option prefix, 243
- loose_, 4146
- lossy-conversions option (ndb_restore), 2221
- lost connection errors, 4038
- low-priority option
 - mysqlimport, 328
- low-priority-updates option
 - mysqld, 452
- low-water mark, 4146
- LOWER(), 1105
- lower_case_file_system system variable, 518
- lower_case_table_names system variable, 518
- low_priority_updates system variable, 517
- LPAD(), 1106
- LRU, 4146
- LRU page replacement, 1674
- LSN, 4146
- LTRIM(), 1106
- Lua, 1799

M

- Mac OS X, 2567
 - installation, 117
- mailing lists, 16
 - archive location, 16
 - guidelines, 19
- main features of MySQL, 6
- maintaining
 - log files, 648
 - tables, 794
- maintenance
 - tables, 300
- MAKEDATE(), 1142
- MAKETIME(), 1142
- MAKE_SET(), 1106
- Making temp file
 - thread state, 905
- malicious SQL statements
 - and MySQL Cluster, 2307
- manage keys
 - thread state, 898
- management client (MySQL Cluster), 2194
 - (see also mgm)
- management node (MySQL Cluster)
 - defined, 1972
- management nodes (MySQL Cluster), 2186
 - (see also mgmd)
- managing MySQL Cluster, 2243
- managing MySQL Cluster processes, 2176
- manual

available formats, 2
 online location, 2
 syntax conventions, 3
 typographical conventions, 3
 Master has sent all binlog to slave; waiting for binlog to be updated
 thread state, 904
 master server, 4147
 master thread, 4147
 master-data option
 mysqldump, 317
 master-info-file option
 mysqld, 1872
 master-retry-count option
 mysqld, 1872
 MASTER_POS_WAIT(), 1200, 1412
 MATCH ... AGAINST(), 1153
 matching
 patterns, 213
 math, 1237
 mathematical functions, 1123
 MAX(), 1205
 MAX(DISTINCT), 1205
 max-binlog-dump-events option
 mysqld, 1897
 max-record-length option
 myisamchk, 351
 max-relay-log-size option
 mysqld, 1873
 MaxBufferedEpochs, 2066
 MAXDB SQL mode, 618
 MaxDMLOperationsPerTransaction, 2048
 maximum memory used, 296
 --maximum option prefix, 243
 maximums
 maximum columns per table, 4086
 maximum number of databases, 4084
 maximum number of tables, 4084
 maximum tables per join, 4084
 table size, 4085
 MaxLCPStartDelay, 2053
 MaxNoOfAttributes, 2054
 MaxNoOfConcurrentIndexOperations, 2048
 MaxNoOfConcurrentOperations, 2047
 MaxNoOfConcurrentScans, 2049
 MaxNoOfConcurrentSubOperations, 2057
 MaxNoOfConcurrentTransactions, 2046
 MaxNoOfExecutionThreads
 ndbmtd, 2079
 MaxNoOfFiredTriggers, 2049
 MaxNoOfIndexes, 2056
 MaxNoOfLocalOperations, 2048
 MaxNoOfLocalScans, 2050
 MaxNoOfOpenFiles, 2053
 MaxNoOfOrderedIndexes, 2055
 MaxNoOfSavedMessages, 2053
 MaxNoOfSubscribers, 2057
 MaxNoOfSubscriptions, 2056
 MaxNoOfTables, 2054
 MaxNoOfTriggers, 2056
 MaxNoOfUniqueHashIndexes, 2055
 MaxParallelScansPerFragment, 2051
 MaxScanBatchSize, 2093
 MaxStartFailRetries, 2089
 max_allowed_packet
 and replication, 1952
 max_allowed_packet system variable, 519
 max_allowed_packet variable, 283
 max_binlog_cache_size system variable, 1902
 max_binlog_size system variable, 1903
 max_binlog_stmt_cache_size system variable, 1903
 max_connections system variable, 519
 MAX_CONNECTIONS_PER_HOUR, 721
 max_connect_errors system variable, 519
 max_delayed_threads system variable, 520
 max_error_count system variable, 520
 max_heap_table_size system variable, 521
 max_insert_delayed_threads system variable, 521
 max_join_size system variable, 522
 max_join_size variable, 283
 max_length_for_sort_data system variable, 522
 max_long_data_size system variable, 523
 max_prepared_stmt_count system variable, 523
 MAX_QUERIES_PER_HOUR, 721
 max_relay_log_size system variable, 523
 MAX_ROWS
 and DataMemory (MySQL Cluster), 2043
 and MySQL Cluster, 1300, 2384
 and NDB, 1300
 max_seeks_for_key system variable, 524
 max_sort_length system variable, 524
 max_sp_recursion_depth system variable, 525
 max_tmp_tables system variable, 525
 MAX_UPDATES_PER_HOUR, 721
 MAX_USER_CONNECTIONS, 721
 max_user_connections system variable, 525
 max_write_lock_count system variable, 526
 MBR, 1232
 MBRContains(), 1232
 MBRDisjoint(), 1232
 MBREqual(), 1232
 MBRIntersects(), 1232
 MBROverlaps(), 1232
 MBRTouches(), 1233
 MBRWithin(), 1233
 MD5(), 1187
 MDL, 4147
 medium trust, 2781

medium-check option
 mysamchk, 350
 mysqlcheck, 305
 MEDIUMBLOB data type, 1043
 MEDIUMINT data type, 1037
 MEDIUMTEXT data type, 1043
 membership
 ndbinfo table, 2292
 memcached, 4147
 MEMCACHED_HOME option
 CMake, 156
 memlock option
 mysqld, 453
 memory allocator
 innodb_use_sys_malloc, 1667
 MEMORY storage engine, 1524, 1699
 and replication, 1953
 memory usage
 mysamchk, 359
 memory use, 296, 879
 in MySQL Cluster, 1988
 Performance Schema, 2528
 memoryusage
 ndbinfo table, 2294
 MemReportFrequency, 2073
 merge, 4148
 MERGE storage engine, 1524, 1709
 MERGE tables
 defined, 1709
 metadata
 database, 2469
 stored routines, 2445
 triggers, 2448
 views, 2461
 metadata lock, 4148
 metadata locking
 transactions, 866
 metadata_locks_cache_size system variable, 526
 method option
 mysqlhotcopy, 388
 methods
 locking, 862
 metrics counter, 4148
 mgmd (MySQL Cluster)
 defined, 1972
 (see also management node (MySQL Cluster))
 MICROSECOND(), 1142
 Microsoft Access, 2641
 Microsoft ADO, 2642, 2687
 Microsoft Excel, 2642
 Microsoft Visual Basic, 2642
 Microsoft Visual InterDev, 2642
 Microsoft Visual Studio, 2662
 MID(), 1106
 midpoint insertion, 1674
 midpoint insertion strategy, 4148
 MIN(), 1206
 MIN(DISTINCT), 1206
 min-examined-row-limit option
 mysqld, 452
 MinFreePct, 2043, 2045
 mini-transaction, 4149
 Minimum Bounding Rectangle, 1232
 minus
 unary (-), 1122
 MINUTE(), 1142
 min_examined_row_limit system variable, 527
 mirror sites, 52
 miscellaneous functions, 1198
 mixed statements (Replication), 1957
 mixed-mode insert, 4149
 MLineFromText(), 1220
 MLineFromWKB(), 1221
 MOD (modulo), 1127
 MOD(), 1127
 modes
 batch, 220
 modulo (%), 1127
 modulo (MOD), 1127
 monitor
 terminal, 197
 monitoring, 3863
 threads, 892
 Monitors
 InnoDB, 1594, 1615, 1620, 1630, 1631
 Mono, 2651
 MONTH(), 1143
 MONTHNAME(), 1143
 MPointFromText(), 1221
 MPointFromWKB(), 1222
 MPolyFromText(), 1221
 MPolyFromWKB(), 1222
 mSQL compatibility, 1114
 msq2mysql, 394
 MSSQL SQL mode, 618
 multi mysqld, 260
 multi-byte character sets, 4046
 multi-byte characters, 1017
 multi-column indexes, 812
 multi-core, 4150
 multi-master replication
 in MySQL Cluster, 2366, 2371
 MULTILINESTRING data type, 1220
 MultiLineString(), 1222
 MultiLineStringFromText(), 1220
 MultiLineStringFromWKB(), 1221
 multiple buffer pools, 1677
 multiple rollback segments, 1677

multiple servers, 650
 multiple-part index, 1277
 multiplication (*), 1122
 MULTIPOINT data type, 1220
 MultiPoint(), 1222
 MultiPointFromText(), 1221
 MultiPointFromWKB(), 1222
 MULTIPOLYGON data type, 1220
 MultiPolygon(), 1223
 MultiPolygonFromText(), 1221
 MultiPolygonFromWKB(), 1222
 mutex, 4150
 mutex_instances table
 performance_schema, 2544
 MVCC, 4150
 MVCC (multi-version concurrency control), 1609
 My
 derivation, 9
 my-print-defaults option
 mysql_plugin, 268
 my.cnf, 4150
 and MySQL Cluster, 2014, 2025, 2026
 in MySQL Cluster, 2254
 my.cnf file, 1937
 my.ini, 4151
 mycnf option
 ndb_config, 2198
 ndb_mgmd, 2189
 MyISAM
 compressed tables, 361, 1697
 converting tables to InnoDB, 1578
 MyISAM key cache, 851
 MyISAM storage engine, 1524, 1691
 myisam-block-size option
 mysqld, 454
 myisam-recover option
 mysqld, 454
 myisam-recover-options option
 mysqld, 454, 1694
 myisamchk, 234, 343
 analyze option, 352
 backup option, 350
 block-search option, 352
 character-sets-dir option, 350
 check option, 349
 check-only-changed option, 349
 correct-checksum option, 350
 data-file-length option, 350
 debug option, 347
 description option, 352
 example output, 352
 extend-check option, 349, 350
 fast option, 349
 force option, 349, 350
 help option, 347
 HELP option, 347
 information option, 349
 keys-used option, 350
 max-record-length option, 351
 medium-check option, 350
 no-symlinks option, 351
 options, 347
 parallel-recover option, 351
 quick option, 351
 read-only option, 350
 recover option, 351
 safe-recover option, 351
 set-auto-increment[option, 352
 set-character-set option, 351
 set-collation option, 351
 silent option, 347
 sort-index option, 352
 sort-records option, 352
 sort-recover option, 351
 tmpdir option, 351
 unpack option, 352
 update-state option, 350
 verbose option, 347
 version option, 347
 wait option, 347
 myisamlog, 234, 360
 myisampack, 234, 361, 1311, 1697
 backup option, 362
 character-sets-dir option, 362
 debug option, 362
 force option, 362
 help option, 361
 join option, 362
 silent option, 362
 test option, 362
 tmpdir option, 362
 verbose option, 362
 version option, 362
 wait option, 362
 myisam_block_size myisamchk variable, 347
 myisam_data_pointer_size system variable, 527
 myisam_ftdump, 234, 342
 count option, 343
 dump option, 343
 help option, 343
 length option, 343
 stats option, 343
 verbose option, 343
 myisam_max_sort_file_size system variable, 528
 myisam_mmap_size system variable, 528
 myisam_recover_options system variable, 529
 myisam_repair_threads system variable, 529
 myisam_sort_buffer_size myisamchk variable, 347

myisam_sort_buffer_size system variable, 530

myisam_stats_method system variable, 530

myisam_use_mmap system variable, 531

MyODBC, 2567

MySQL

defined, 4

introduction, 4

pronunciation, 6

upgrading, 270

mysql, 233, 272, 4151

auto-rehash option, 276

auto-vertical-output option, 276

batch option, 276

bind-address option, 277

character-sets-dir option, 277

charset command, 284

clear command, 284

column-names option, 277

column-type-info option, 277

comments option, 277

compress option, 277

connect command, 285

database option, 277

debug option, 277

debug-check option, 277

debug-info option, 277

default-auth option, 277

default-character-set option, 277

delimiter command, 285

delimiter option, 278

disable named commands, 278

edit command, 285

ego command, 285

enable-clear-text-plugin option, 278

execute option, 278

exit command, 285

force option, 278

go command, 285

help command, 284

help option, 276

host option, 278

html option, 278

i-am-a-dummy option, 281

ignore-spaces option, 278

init-command option, 278

line-numbers option, 278

local-infile option, 278

named-commands option, 279

no-auto-rehash option, 279

no-beep option, 279

no-named-commands option, 279

no-pager option, 279

no-tee option, 279

nopager command, 285

notee command, 285

nowarning command, 286

one-database option, 279

pager command, 286

pager option, 280

password option, 280

pipe option, 280

plugin-dir option, 280

port option, 280

print command, 286

prompt command, 286

prompt option, 280

protocol option, 280

quick option, 280

quit command, 286

raw option, 280

reconnect option, 281

rehash command, 286

safe-updates option, 281

secure-auth option, 281

show-warnings option, 281

sigint-ignore option, 281

silent option, 281

skip-column-names option, 282

skip-line-numbers option, 282

socket option, 282

source command, 286

SSL options, 282

status command, 286

system command, 286

table option, 282

tee command, 287

tee option, 282

unbuffered option, 282

use command, 287

user option, 282

verbose option, 282

version option, 282

vertical option, 282

wait option, 282

warnings command, 287

xml option, 282

MySQL binary distribution, 49

MySQL C type, 3072

MySQL Cluster, 1968

"quick" configuration, 2023

administration, 2138, 2176, 2186, 2194, 2194, 2240, 2246, 2264

and application feature requirements, 1985

and DNS, 1998

and INFORMATION_SCHEMA, 2308

and IP addressing, 1998

and MySQL privileges, 2306

and MySQL root user, 2307, 2309

and networking, 1977
and transactions, 2044
API node, 1972, 2090
applications supported, 1984
availability, 1982
available platforms, 1968
backups, 2217, 2249, 2250, 2250, 2253, 2254
benchmarks, 2175
CHECKPOINT Events, 2261
cluster logs, 2258, 2260
CLUSTERLOG commands, 2260
CLUSTERLOG STATISTICS command, 2264
commands, 2138, 2176, 2186, 2194, 2246
compared to InnoDB, 1982, 1982, 1984, 1985
compared to standalone MySQL Server, 1981, 1982, 1984, 1985
compiling with icc, 3854
concepts, 1972
configuration, 1997, 2022, 2023, 2033, 2034, 2038, 2089, 2193, 2254
configuration (example), 2026
configuration changes, 2256
configuration files, 2014, 2025
configuration parameters, 2104, 2105, 2117, 2119, 2121
configuring, 2253
CONNECTION Events, 2261
connectstring, 2032
CREATE NODEGROUP command, 2248
creating large tables, 1300
data node, 1972, 2038
data nodes, 2176, 2184
defining node hosts, 2033
direct connections between nodes, 2097
Disk Data tables (see MySQL Cluster Disk Data)
DROP NODEGROUP command, 2249
ENTER SINGLE USER MODE command, 2248
ERROR Events, 2264
error logs, 2182
event log format, 2261
event logging thresholds, 2260
event logs, 2258, 2260
event severity levels, 2260
event types, 2259, 2261
execution threads, 2079
EXIT command, 2248
EXIT SINGLE USER MODE command, 2248
FAQ, 3945
GCP Stop errors, 2087
general description, 1970
HELP command, 2246
HostName parameter
 and security, 2303
INFO Events, 2264
information sources, 1970
insecurity of communication protocols, 2302
installation, 1997
installation (Linux), 2000
installation (Windows), 2005
installing binary (Windows), 2006
installing binary release (Linux), 2000
installing from source (Linux), 2004
installing from source (Windows), 2009
installing RPM (Linux), 2002
interconnects, 2174
Java clients, 1973
log files, 2182, 2185
logging commands, 2260
management client (ndb_mgm), 2194
management commands, 2264
management node, 1972, 2034
management nodes, 2186
managing, 2243
MAX_ROWS, 1300
memory usage and recovery, 1988, 2256
mgm, 2240
mgm client, 2246
mgm management client, 2264
mgm process, 2194
mgmd, 2240
mgmd process, 2186
monitoring, 2330
multiple management servers, 2258
mysqld options and variables for, 2126
mysqld process, 2138, 2254
ndbd, 2176, 2240
ndbd process, 2176, 2266
ndbinfo_select_all, 2183
ndbmtd, 2184
ndb_mgm, 2017, 2194
ndb_mgmd process, 2186
network configuration
 and security, 2303
network transporters, 2174, 2175
networking, 2097, 2098, 2100
node failure (single user mode), 2279
node identifiers, 2098, 2099, 2101, 2101
node logs, 2258
NODERESTART Events, 2262
nodes and node groups, 1975
nodes and types, 1972
partitioning support, 1988
partitions, 1975
performance, 2175
performing queries, 2017
preparing for replication, 2355
process management, 2176
QUIT command, 2248

- replicas, 1975
- replication, 2342
 - (see also MySQL Cluster replication)
- REPORT command, 2247
- requirements, 1977
- resetting, 2256
- RESTART command, 2247
- restarting, 2021
- restoring backups, 2217
- rolling restarts (multiple management servers), 2258
- runtime statistics, 2264
- SCI (Scalable Coherent Interface), 2100, 2175
- security, 2302
 - and firewalls, 2304, 2305
 - and HostName parameter, 2303
 - and network configuration, 2303
 - and network ports, 2306
 - and remote administration, 2306
 - networking, 2302
- security procedures, 2308
- shared memory transport, 2098
- SHOW command, 2246
- SHUTDOWN command, 2248
- shutting down, 2021
- single user mode, 2248, 2278
- SQL node, 1972, 2090
- SQL nodes, 2254
- SQL statements for monitoring, 2279
- START BACKUP command, 2361
- START command, 2246
- start phases (summary), 2244
- starting, 2023
- starting nodes, 2010, 2016
- starting or restarting, 2244
- STARTUP Events, 2262
- STATISTICS Events, 2264
- STATUS command, 2247
- status variables, 2161
- STOP command, 2247
- storage requirements, 1069
- thread states, 906
- trace files, 2182
- transaction handling, 1991
- transporters
 - Scalable Coherent Interface (SCI), 2100
 - shared memory (SHM), 2098
 - TCP/IP, 2097
- troubleshooting backups, 2254
- upgrades and downgrades, 2022, 2255
- using tables and data, 2017
- MySQL Cluster Disk Data, 2310
 - creating log file groups, 2311
 - creating tables, 2310, 2312
 - creating tablespaces, 2312
 - dropping Disk Data objects, 2313
 - storage requirements, 2316
- MySQL Cluster How-To, 1997
- MySQL Cluster limitations, 1986
 - and differences from standard MySQL limits, 1988
- binary logging, 1993
- CREATE TABLE statements, 1996
- database objects, 1992
- Disk Data storage, 1994
- error handling and reporting, 1991
- geometry data types, 1987
- implementation, 1993
- imposed by configuration, 1989
- INSERT IGNORE, UPDATE IGNORE, and REPLACE statements, 1996
- memory usage and transaction handling, 1991
- multiple data nodes, 1996
- multiple management servers, 1994
- multiple MySQL servers, 1994
- partitioning, 1988
- performance, 1993
- replication, 1988, 1995
 - resolved in current version from previous versions, 1995
- syntax, 1987
- transactions, 1989
- unsupported features, 1992
- MySQL Cluster NDB 7.2, 1979
- MySQL Cluster processes, 2176
- MySQL Cluster programs, 2176
- MySQL Cluster replication, 2342
 - and --initial option, 2349
 - and circular replication, 2346
 - and primary key, 2348
 - and single point of failure, 2358
 - and unique keys, 2348
 - backups, 2360
 - circular replication, 2366
 - concepts, 2343, 2344
 - conflict resolution, 2371
 - failover, 2358, 2359
 - gap event, 2345
 - known issues, 2345
 - loss of connection, 2345
 - multi-master replication, 2366
 - point-in-time recovery, 2365
 - preparing, 2355
 - requirements, 2344
 - reset-slave.pl backup automation script, 2363
 - restoring from backup, 2360
 - starting, 2356
 - storage engines other than NDB on slave, 2350
 - synchronization of master and slave, 2363
 - system tables used, 2352

- MySQL Cluster Replication
 - and NDB API `_slave` status variables, 2343
- mysql command options, 273
- mysql commands
 - list of, 284
- MySQL Dolphin name, 9
- MySQL Enterprise Audit, 757, 3865
- MySQL Enterprise Backup, 3864, 4151
- MySQL Enterprise Monitor, 3863
- MySQL Enterprise Security, 729, 736, 3865
- MySQL history, 9
- mysql history file, 289
- MySQL Installer, 61, 61
- MySQL mailing lists, 16
- MySQL name, 9
- MySQL Notifier, 85
- MySQL privileges
 - and MySQL Cluster, 2307
- mysql prompt command, 287
- MySQL server
 - mysqld, 254, 401
- mysql source (command for reading from text files), 221, 290
- MySQL source distribution, 49
- MySQL storage engines, 1524
- MySQL system tables
 - and MySQL Cluster, 2307
 - and replication, 1954
- MySQL Thread Pool, 884, 3865
- MySQL version, 52
- mysql \. (command for reading from text files), 221, 290
- MySql.Data class, 2791
- MySql.Data.dll library, 2660, 2660
- mysql.data.ef.dll library, 2662
- mysql.event table, 2456
- mysql.ndb_binlog_index table, 2352
 - (see also MySQL Cluster replication)
- mysql.server, 232, 259
 - basedir option, 260
 - datadir option, 260
 - pid-file option, 260
 - service-startup-timeout option, 260
 - use-mysqld_safe option, 260
 - user option, 260
- mysql.sock
 - protection, 4054
- mysql.visualstudio.dll library, 2662
- MYSQL323 SQL mode, 618
- MYSQL40 SQL mode, 618
- mysqlaccess, 234, 367
 - brief option, 368
 - commit option, 368
 - copy option, 368
 - db option, 368
 - debug option, 368
 - help option, 368
 - host option, 368
 - howto option, 368
 - old_server option, 369
 - password option, 369
 - plan option, 369
 - preview option, 369
 - relnotes option, 369
 - rhost option, 369
 - rollback option, 369
 - spassword option, 369
 - superuser option, 369
 - table option, 369
 - user option, 369
 - version option, 369
- mysqladmin, 233, 293, 1272, 1321, 1503, 1508, 1513, 1516
 - bind-address option, 298
 - character-sets-dir option, 298
 - compress option, 298
 - count option, 298
 - debug option, 298
 - debug-check option, 298
 - debug-info option, 298
 - default-auth option, 298
 - default-character-set option, 298
 - enable-clear-text-plugin option, 298
 - force option, 298
 - help option, 297
 - host option, 298
 - no-beep option, 299
 - password option, 299
 - pipe option, 299
 - plugin-dir option, 299
 - port option, 299
 - protocol option, 299
 - relative option, 299
 - silent option, 299
 - sleep option, 299
 - socket option, 299
 - SSL options, 299
 - user option, 300
 - verbose option, 300
 - version option, 300
 - vertical option, 300
 - wait option, 300
- mysqladmin command options, 296
- mysqladmin option
 - mysqld_multi, 261
- mysqlbackup command, 4151
- mysqlbinlog, 234, 370
 - base64-output option, 372
 - bind-address option, 373

- character-sets-dir option, 373
- database option, 373
- debug option, 374
- debug-check option, 374
- debug-info option, 374
- default-auth option, 374
- disable-log-bin option, 375
- force-if-open option, 375
- force-read option, 375
- help option, 372
- hexdump option, 375
- host option, 375
- local-load option, 375
- offset option, 375
- password option, 375
- plugin-dir option, 375
- port option, 376
- position option, 376
- protocol option, 376
- read-from-remote-server option, 376
- result-file option, 376
- server-id option, 376
- server-id-bits option, 376
- set-charset option, 376
- short-form option, 376
- socket option, 376
- start-datetime option, 377
- start-position option, 377
- stop-datetime option, 377
- stop-position option, 377
- to-last-log option, 377
- user option, 377
- verbose option, 377
- version option, 378
- mysqlbug, 265
- MySqlBulkLoader class, 2766
- mysqlcheck, 233, 300
 - all-databases option, 303
 - all-in-1 option, 303
 - analyze option, 303
 - auto-repair option, 304
 - bind-address option, 304
 - character-sets-dir option, 304
 - check option, 304
 - check-only-changed option, 304
 - check-upgrade option, 304
 - compress option, 304
 - databases option, 304
 - debug option, 304
 - debug-check option, 304
 - debug-info option, 304
 - default-auth option, 305
 - default-character-set option, 304
 - extended option, 304
 - fast option, 305
 - fix-db-names option, 305
 - fix-table-names option, 305
 - force option, 305
 - help option, 303
 - host option, 305
 - medium-check option, 305
 - optimize option, 305
 - password option, 305
 - pipe option, 305
 - plugin-dir option, 306
 - port option, 306
 - protocol option, 306
 - quick option, 306
 - repair option, 306
 - silent option, 306
 - socket option, 306
 - SSL options, 306
 - tables option, 306
 - use-frm option, 306
 - user option, 306
 - verbose option, 306
 - version option, 307
 - write-binlog option, 307
- mysqlclient library, 2565
- MySQLClient namespace, 2791
- MySQLCommand class, 2698
- MySQLCommand class, 2748
- mysqlcppconn-static.lib, 2996
- mysqlcppconn.dll, 2996
- mysqld, 232, 4152
 - abort-slave-event-count option, 1870
 - allow-suspicious-udfs option, 434
 - ansi option, 434
 - as MySQL Cluster process, 2138, 2254
 - audit-log option, 764
 - basedir option, 435
 - big-tables option, 435
 - bind-address option, 435
 - binlog-do-db option, 1894
 - binlog-format option, 436
 - binlog-ignore-db option, 1896
 - binlog-row-event-max-size option, 1891
 - bootstrap option, 436
 - character-set-client-handshake option, 437
 - character-set-filesystem option, 437
 - character-set-server option, 438
 - character-sets-dir option, 437
 - chroot option, 438
 - collation-server option, 438
 - command options, 433
 - console option, 438
 - core-file option, 439
 - datadir option, 439

debug option, 439
debug-sync-timeout option, 440
default-character-set option, 440
default-collation option, 440
default-storage-engine option, 441
default-time-zone option, 441
delay-key-write option, 441, 1694
des-key-file option, 442
disconnect-slave-event-count option, 1870
enable-named-pipe option, 442
enable-pstack option, 442
event-scheduler option, 443
exit-info option, 443
external-locking option, 444
flush option, 444
gdb option, 444
general-log option, 445
help option, 434
ignore-builtin-innodb option, 1546
init-file option, 445
innodb option, 1546
innodb-status-file option, 1546
install option, 445
install-manual option, 445
language option, 446
large-pages option, 446
lc-messages option, 447
lc-messages-dir option, 447
log option, 447
log-bin option, 1892
log-bin-index option, 1892
log-bin-trust-function-creators option, 1892
log-bin-use-v1-row-events option, 1893
log-error option, 448
log-isam option, 448
log-long-format option, 448
log-output option, 449
log-queries-not-using-indexes option, 449
log-short-format option, 450, 1894
log-slave-updates option, 1870
log-slow-admin-statements option, 450
log-slow-queries option, 450
log-slow-slave-statements option, 1871
log-tc option, 450
log-tc-size option, 451
log-warnings option, 451, 1871
low-priority-updates option, 452
master-info-file option, 1872
master-retry-count option, 1872
max-binlog-dump-events option, 1897
max-relay-log-size option, 1873
memlock option, 453
min-examined-row-limit option, 452
myisam-block-size option, 454
myisam-recover option, 454
myisam-recover-options option, 454, 1694
MySQL server, 254, 401
ndb-batch-size option, 2138
ndb-blob-read-batch-bytes option, 2140
ndb-blob-write-batch-bytes option, 2140
ndb-cluster-connection-pool option, 2139
ndb-connectstring option, 2141
ndb-log-apply-status, 2142
ndb-log-empty-epochs, 2144
ndb-log-transaction-id, 2143
ndb-nodeid, 2143
ndbcluster option, 2142
old-alter-table option, 455
old-style-user-limits option, 455
one-thread option, 456
open-files-limit option, 456
partition option, 456
pid-file option, 456
plugin option prefix, 457
plugin-load option, 457
port option, 458
port-open-timeout option, 458
read-only option, 1873
relay-log option, 1873
relay-log-index option, 1874
relay-log-info-file option, 1874
relay-log-purge option, 1874
relay-log-recovery option, 1875
relay-log-space-limit option, 1875
remove option, 458
replicate-do-db option, 1875
replicate-do-table option, 1878
replicate-ignore-db option, 1877
replicate-ignore-table option, 1878
replicate-rewrite-db option, 1878
replicate-same-server-id option, 1879
replicate-wild-do-table option, 1879
replicate-wild-ignore-table option, 1880
report-host option, 1880
report-password option, 1880
report-port option, 1881
report-user option, 1881
role in MySQL Cluster (see SQL Node (MySQL Cluster))
safe-mode option, 459
safe-show-database option, 459
safe-user-create option, 459
secure-auth option, 460
secure-file-priv option, 460
server-id option, 1858
server-id-bits option, 2144
shared-memory option, 460
shared-memory-base-name option, 460

show-slave-auth-info option, 1882
 skip-concurrent-insert option, 460
 skip-event-scheduler option, 461
 skip-external-locking option, 460
 skip-grant-tables option, 461
 skip-host-cache option, 461
 skip-innodb option, 461, 1547
 skip-name-resolve option, 461
 skip-ndbcluster option, 2145
 skip-networking option, 461
 skip-partition option, 462
 skip-safemalloc option, 462
 skip-show-database option, 463
 skip-slave-start option, 1882
 skip-stack-trace option, 463
 skip-symbolic-links option, 462
 skip-thread-priority option, 463
 slave-load-tmpdir option, 1882
 slave-max-allowed-packet, 1869
 slave-net-timeout option, 1883
 slave-skip-errors option, 1883
 slave_compressed_protocol option, 1882
 slow-query-log option, 463
 slow-start-timeout option, 464
 socket option, 464
 sporadic-binlog-dump-fail option, 1897
 sql-mode option, 464
 SSL options, 462
 standalone option, 462
 starting, 690
 super-large-pages option, 462
 symbolic-links option, 462
 sysdate-is-now option, 465
 tc-heuristic-recover option, 465
 temp-pool option, 465
 tmpdir option, 466
 transaction-isolation option, 465
 user option, 466
 verbose option, 467
 version option, 467
 mysqld (MySQL Cluster), 2176
 mysqld library, 2565
 mysqld option
 malloc-lib, 256
 mysqld_multi, 262
 mysqld_safe, 257
 mysql_plugin, 268
 mysqld options, 868
 mysqld options and variables
 MySQL Cluster, 2126
 mysqld server
 buffer sizes, 868
 mysqld-version option
 mysqld_safe, 257
 mysqldump, 192, 233, 307, 4152
 add-drop-database option, 313
 add-drop-table option, 313
 add-drop-trigger option, 313
 add-locks option, 313
 all-databases option, 313
 all-tablespaces option, 313
 allow-keywords option, 313
 apply-slave-statements option, 313
 bind-address option, 313
 character-sets-dir option, 314
 comments option, 314
 compact option, 314
 compatible option, 314
 complete-insert option, 314
 compress option, 314
 create-options option, 314
 databases option, 314
 debug option, 314
 debug-check option, 315
 debug-info option, 315
 default-auth option, 315
 default-character-set option, 315
 delayed-insert option, 315
 delete-master-logs option, 315
 disable-keys option, 315
 dump-date option, 315
 dump-slave option, 315
 events option, 316
 extended-insert option, 316
 fields-enclosed-by option, 316, 328
 fields-escaped-by option, 316, 328
 fields-optionally-enclosed-by option, 316, 328
 fields-terminated-by option, 316, 328
 first-slave option, 316
 flush-logs option, 316
 flush-privileges option, 316
 force option, 316
 help option, 313
 hex-blob option, 316
 host option, 316
 ignore-table option, 317
 include-master-host-port option, 317
 insert-ignore option, 317
 lines-terminated-by option, 317, 328
 lock-all-tables option, 317
 lock-tables option, 317
 log-error option, 317
 master-data option, 317
 no-autocommit option, 318
 no-create-db option, 318
 no-create-info option, 318
 no-data option, 319
 no-set-names option, 319

no-tablespaces option, 319
opt option, 319
order-by-primary option, 319
password option, 319
pipe option, 319
plugin-dir option, 319
port option, 320
problems, 325, 4080
protocol option, 320
quick option, 320
quote-names option, 320
replace option, 320
result-file option, 320
routines option, 320
set-charset option, 320
single-transaction option, 321
skip-comments option, 321
skip-opt option, 321
socket option, 321
SSL options, 321
tab option, 321
tables option, 322
triggers option, 322
tz-utc option, 322
user option, 322
using for backups, 780
verbose option, 322
version option, 322
views, 325, 4080
where option, 322
workarounds, 325, 4080
xml option, 322

mysqldumpslow, 234, 384
 debug option, 385
 help option, 385
 verbose option, 386

mysqld_multi, 232, 260
 config-file option, 261
 defaults-extra-file option, 261
 defaults-file option, 261
 example option, 261
 help option, 261
 log option, 261
 mysqladmin option, 261
 mysqld option, 262
 no-defaults option, 261
 no-log option, 262
 password option, 262
 silent option, 262
 tcp-ip option, 262
 user option, 262
 verbose option, 262
 version option, 262

mysqld_safe, 232, 254
 basedir option, 256
 core-file-size option, 256
 datadir option, 256
 defaults-extra-file option, 256
 defaults-file option, 256
 help option, 256
 ledir option, 256
 log-error option, 256
 malloc-lib option, 256
 mysqld option, 257
 mysqld-version option, 257
 nice option, 257
 no-defaults option, 257
 open-files-limit option, 257
 pid-file option, 257
 port option, 257
 skip-kill-mysqld option, 257
 skip-syslog option, 258
 socket option, 258
 syslog option, 258
 syslog-tag option, 258
 timezone option, 258
 user option, 258

mysqlhotcopy, 235, 386
 addtodest option, 388
 allowold option, 388
 checkpoint option, 388
 chroot option, 388
 debug option, 388
 dryrun option, 388
 flushlog option, 388
 help option, 388
 host option, 388
 keepold option, 388
 method option, 388
 noindices option, 388
 old_server option, 389
 password option, 388
 port option, 389
 quiet option, 389
 record_log_pos option, 389
 regexp option, 389
 resetmaster option, 389
 resetslave option, 389
 socket option, 389
 suffix option, 389
 tmpdir option, 389
 user option, 389

mysqlimport, 192, 234, 325, 1343
 bind-address option, 327
 character-sets-dir option, 327
 columns option, 327
 compress option, 327
 debug option, 327

- debug-check option, 327
- debug-info option, 327
- default-auth option, 328
- default-character-set option, 327
- delete option, 328
- force option, 328
- help option, 327
- host option, 328
- ignore option, 328
- ignore-lines option, 328
- local option, 328
- lock-tables option, 328
- low-priority option, 328
- password option, 328
- pipe option, 329
- plugin-dir option, 329
- port option, 329
- protocol option, 329
- replace option, 329
- silent option, 329
- socket option, 329
- SSL options, 329
- use-threads option, 329
- user option, 329
- verbose option, 330
- version option, 330
- MySQLInstallerConsole, 76
 - action option, 77
 - catalog option, 77
 - config option, 77
 - help option, 77
 - install action option, 77
 - list action option, 77
 - nowait option, 77
 - product option, 77
 - remove action option, 77
 - status action option, 77
 - type option, 77
 - updates option, 77
 - upgrade action option, 77
 - user option, 77
- mysqlshow, 234, 330
 - bind-address option, 332
 - character-sets-dir option, 332
 - compress option, 332
 - count option, 332
 - debug option, 332
 - debug-check option, 332
 - debug-info option, 332
 - default-auth option, 333
 - default-character-set option, 333
 - help option, 332
 - host option, 333
 - keys option, 333
 - password option, 333
 - pipe option, 333
 - plugin-dir option, 333
 - port option, 333
 - protocol option, 333
 - show-table-type option, 333
 - socket option, 333
 - SSL options, 334
 - status option, 334
 - user option, 334
 - verbose option, 334
 - version option, 334
- mysqlslap, 234, 334
 - auto-generate-sql option, 337
 - auto-generate-sql-add-autoincrement option, 337
 - auto-generate-sql-execute-number option, 337
 - auto-generate-sql-guid-primary option, 338
 - auto-generate-sql-load-type option, 338
 - auto-generate-sql-secondary-indexes option, 338
 - auto-generate-sql-unique-query-number option, 338
 - auto-generate-sql-unique-write-number option, 338
 - auto-generate-sql-write-number option, 338
 - commit option, 338
 - compress option, 338
 - concurrency option, 338
 - create option, 338
 - create-and-drop-schema option, 338
 - create-schema option, 338
 - csv option, 339
 - debug option, 339
 - debug-check option, 339
 - debug-info option, 339
 - default-auth option, 339
 - delimiter option, 339
 - detach option, 339
 - enable-clear-text-plugin option, 339
 - engine option, 339
 - help option, 337
 - host option, 339
 - iterations option, 339
 - number-char-cols option, 339
 - number-int-cols option, 339
 - number-of-queries option, 339
 - only-print option, 340
 - password option, 340
 - pipe option, 340
 - plugin-dir option, 340
 - port option, 340
 - post-query option, 340
 - post-system option, 340
 - pre-query option, 340
 - pre-system option, 341
 - protocol option, 341
 - query option, 341

shared-memory-base-name option, 340
 silent option, 341
 socket option, 341
 SSL options, 341
 user option, 341
 verbose option, 341
 version option, 341
 mysqltest
 MySQL Test Suite, 3794
 mysql_affected_rows(), 3082
 mysql_autocommit(), 3083
 MYSQL_BIND C type, 3134
 mysql_change_user(), 3083
 mysql_character_set_name(), 3084
 mysql_client_find_plugin(), 3169
 mysql_client_register_plugin(), 3170
 mysql_close(), 3085
 mysql_cluster_move_privileges, 2328
 mysql_cluster_privileges_are_distributed, 2328
 mysql_commit(), 3085
 mysql_config, 394
 cflags option, 394
 embedded option, 394
 include option, 394
 libmysqld-libs option, 394
 libs option, 394
 libs_r option, 394
 plugindir option, 394
 port option, 395
 socket option, 395
 version option, 395
 mysql_connect(), 3085
 mysql_convert_table_format, 235, 389
 force option, 390
 help option, 390
 host option, 390
 password option, 390
 port option, 390
 socket option, 390
 type option, 390
 user option, 390
 verbose option, 390
 version option, 390
 mysql_create_db(), 3086
 MYSQL_DATADIR option
 CMake, 152
 mysql_data_seek(), 3087
 MYSQL_DEBUG environment variable, 193, 236, 3861
 mysql_debug(), 3087
 mysql_drop_db(), 3087
 mysql_dump_debug_info(), 3088
 mysql_eof(), 3089
 mysql_errno(), 3090
 mysql_error(), 3090
 mysql_escape_string(), 3091
 mysql_fetch_field(), 3091
 mysql_fetch_fields(), 3092
 mysql_fetch_field_direct(), 3092
 mysql_fetch_lengths(), 3093
 mysql_fetch_row(), 3093
 MYSQL_FIELD C type, 3072
 mysql_field_count(), 3094, 3108
 MYSQL_FIELD_OFFSET C type, 3073
 mysql_field_seek(), 3095
 mysql_field_tell(), 3095
 mysql_find_rows, 235, 390
 help option, 391
 regexp option, 391
 rows option, 391
 skip-use-db option, 391
 start_row option, 391
 mysql_fix_extensions, 235, 391
 mysql_free_result(), 3096
 mysql_get_character_set_info(), 3096
 mysql_get_client_info(), 3097
 mysql_get_client_version(), 3097
 mysql_get_host_info(), 3097
 mysql_get_proto_info(), 3097
 mysql_get_server_info(), 3098
 mysql_get_server_version(), 3098
 mysql_get_ssl_cipher(), 3098
 MYSQL_GROUP_SUFFIX environment variable, 193
 mysql_hex_string(), 3099
 MYSQL_HISTFILE environment variable, 193, 289
 MYSQL_HOME environment variable, 193
 MYSQL_HOST environment variable, 193, 240
 mysql_info(), 1260, 1337, 1351, 1389, 3099
 mysql_init(), 3100
 mysql_insert_id(), 31, 1337, 3100
 mysql_install_db, 165, 233, 265
 basedir option, 266
 datadir option, 266
 force option, 266
 ldata option, 266
 rpm option, 266
 skip-name-resolve option, 266
 srcdir option, 266
 user option, 267
 verbose option, 267
 windows option, 267
 mysql_kill(), 3102
 mysql_library_end(), 3102
 mysql_library_init(), 3103
 mysql_list_dbs(), 3104
 mysql_list_fields(), 3105
 mysql_list_processes(), 3105
 mysql_list_tables(), 3106
 mysql_load_plugin(), 3170

mysql_load_plugin_v(), 3171
MYSQL_MAINTAINER_MODE option
 CMake, 155
mysql_more_results(), 3107
mysql_next_result(), 3107
mysql_num_fields(), 3108
mysql_num_rows(), 3109
mysql_options(), 3110
mysql_ping(), 3114
mysql_plugin, 233, 267
 basedir option, 268
 datadir option, 268
 help option, 268
 my-print-defaults option, 268
 mysqld option, 268
 no-defaults option, 268
 plugin-dir option, 268
 plugin-ini option, 268
 print-defaults option, 268
 verbose option, 269
 version option, 269
mysql_plugin_options(), 3171
MYSQL_PROJECT_NAME option
 CMake, 155
MYSQL_PS1 environment variable, 193
MYSQL_PWD environment variable, 193, 236, 240
mysql_query(), 3115, 3172
mysql_real_connect(), 3115
mysql_real_escape_string(), 939, 1107, 3119
mysql_real_query(), 3120
mysql_refresh(), 3121
mysql_reload(), 3122
MYSQL_RES C type, 3072
mysql_rollback(), 3123
MYSQL_ROW C type, 3072
mysql_row_seek(), 3123
mysql_row_tell(), 3123
mysql_secure_installation, 233, 269
mysql_select_db(), 3124
MYSQL_SERVER_AUTH_INFO plugin structure, 3834
mysql_server_end(), 3168
mysql_server_init(), 3168
mysql_setpermission, 235, 391
 help option, 392
 host option, 392
 password option, 392
 port option, 392
 socket option, 392
 user option, 392
mysql_set_character_set(), 3124
mysql_set_local_infile_default(), 3125, 3125
mysql_set_server_option(), 3126
mysql_shutdown(), 3127
mysql_sqlstate(), 3128
mysql_ssl_set(), 3128
mysql_stat(), 3129
MYSQL_STMT C type, 3134
mysql_stmt_affected_rows(), 3142
mysql_stmt_attr_get(), 3143
mysql_stmt_attr_set(), 3143
mysql_stmt_bind_param(), 3144
mysql_stmt_bind_result(), 3145
mysql_stmt_close(), 3146
mysql_stmt_data_seek(), 3146
mysql_stmt_errno(), 3147
mysql_stmt_error(), 3147
mysql_stmt_execute(), 3148
mysql_stmt_fetch(), 3151
mysql_stmt_fetch_column(), 3156
mysql_stmt_field_count(), 3157
mysql_stmt_free_result(), 3157
mysql_stmt_init(), 3157
mysql_stmt_insert_id(), 3157
mysql_stmt_next_result(), 3158
mysql_stmt_num_rows(), 3159
mysql_stmt_param_count(), 3159
mysql_stmt_param_metadata(), 3160
mysql_stmt_prepare(), 3160
mysql_stmt_reset(), 3161
mysql_stmt_result_metadata, 3162
mysql_stmt_row_seek(), 3162
mysql_stmt_row_tell(), 3163
mysql_stmt_send_long_data(), 3163
mysql_stmt_sqlstate(), 3165
mysql_stmt_store_result(), 3166
mysql_store_result(), 3129, 3172
MYSQL_TCP_PORT environment variable, 193, 236, 656, 656
MYSQL_TCP_PORT option
 CMake, 155
mysql_thread_end(), 3167
mysql_thread_id(), 3131
mysql_thread_init(), 3167
mysql_thread_safe(), 3168
MYSQL_TIME C type, 3136
mysql_tzinfo_to_sql, 233, 269
MYSQL_UNIX_ADDR option
 CMake, 155
MYSQL_UNIX_PORT environment variable, 166, 193, 236, 656, 656
mysql_upgrade, 233, 270, 712
 basedir option, 271
 datadir option, 271
 debug-check option, 271
 debug-info option, 271
 default-auth option, 271
 force option, 272
 help option, 271

- mysql_upgrade_info file, 271
- plugin-dir option, 272
- tmpdir option, 272
- upgrade-system-tables option, 272
- user option, 272
- verbose option, 272
- write-binlog option, 272
- mysql_upgrade_info file
 - mysql_upgrade, 271
- mysql_use_result(), 3131
- mysql_waitpid, 235, 392
 - help option, 392
 - verbose option, 393
 - version option, 393
- mysql_warning_count(), 3132
- mysql_zap, 235, 393
- my_bool C type, 3073
- my_bool values
 - printing, 3073
- my_init(), 3167
- my_print_defaults, 235, 395
 - config-file option, 396
 - debug option, 396
 - defaults-extra-file option, 396
 - defaults-file option, 396
 - defaults-group-suffix option, 396
 - extra-file option, 396
 - help option, 395
 - no-defaults option, 396
 - verbose option, 396
 - version option, 396
- my_ulonglong C type, 3073
- my_ulonglong values
 - printing, 3073

N

- named pipes, 107, 112
- named-commands option
 - mysql, 279
- named_pipe system variable, 531
- names, 943
 - case sensitivity, 946
 - variables, 956
- NAME_CONST(), 1201, 2467
- name_file option
 - comp_err, 265
- naming
 - releases of MySQL, 49
- NATIONAL CHAR data type, 1042
- NATIONAL VARCHAR data type, 1042
- native backup and restore
 - backup identifiers, 2251
- native functions

- adding, 3852
- native thread support, 48
- natural key, 4152
- NATURAL LEFT JOIN, 1366
- NATURAL LEFT OUTER JOIN, 1366
- NATURAL RIGHT JOIN, 1366
- NATURAL RIGHT OUTER JOIN, 1366
- NCHAR data type, 1042
- NDB
 - creating large tables using, 1300
 - MAX_ROWS, 1300
- NDB API
 - and distributed grant tables, 2330
 - and distributed privileges, 2330
- NDB API counters (MySQL Cluster), 2330
 - scope, 2333
 - status variables associated with, 2335
 - types, 2334
- NDB API _slave status variables
 - and MySQL Cluster Replication, 2343
- ndb option
 - perror, 397
- NDB statistics variables
 - and NDB API counters, 2335
- NDB statistics variables (MySQL Cluster), 2330
 - scope, 2333
 - types, 2334
- NDB storage engine (see MySQL Cluster)
 - FAQ, 3945
- NDB tables
 - and MySQL root user, 2307
- NDB utilities
 - security issues, 2309
- NDB\$MAX(), 2374
- NDB\$MAX_DELETE_WIN(), 2375
- NDB\$OLD, 2374
- ndb-batch-size option
 - mysqld, 2138
- ndb-blob-read-batch-bytes option
 - mysqld, 2140
- ndb-blob-write-batch-bytes option
 - mysqld, 2140
- ndb-cluster-connection-pool option
 - mysqld, 2139
- ndb-connectstring option
 - mysqld, 2141
 - ndb_config, 2197
- ndb-connectstring option (MySQL Cluster programs), 2241
- ndb-deferred-constraints option (MySQL Cluster), 2141
- ndb-distribution option (MySQL Cluster), 2141
- ndb-log-apply-status option
 - mysqld, 2142
- ndb-log-empty-epochs option

- mysqld, 2144
- ndb-log-transaction-id option
 - mysqld, 2143
- ndb-log-update-as-write (mysqld option), 2372
- ndb-mgmd-host option (MySQL Cluster programs), 2242
- ndb-mgmd-host option (MySQL Cluster), 2142
- ndb-nodegroup-map option (ndb_restore), 2225
- ndb-nodeid option
 - mysqld, 2143
- ndb-nodeid option (MySQL Cluster programs), 2242
- ndb-optimized-node-selection option (MySQL Cluster), 2243
- ndbcluster option
 - mysqld, 2142
- NDBCLUSTER storage engine (see MySQL Cluster)
- ndbd, 2176, 2176
- ndbd (MySQL Cluster)
 - defined, 1972
 - (see also data node (MySQL Cluster))
- ndbd_redo_log_reader, 2216
- ndbinfo database, 2282
 - and query cache, 2284
 - basic usage, 2285
 - determining support for, 2282
- ndbinfo_select_all, 2176, 2183
- ndbmtd, 2176, 2184
 - configuration, 2080, 2081
 - MaxNoOfExecutionThreads, 2079
 - trace files, 2185, 2185
- ndb_apply_status table (MySQL Cluster replication), 2353, 2353, 2359
 - (see also MySQL Cluster replication)
- ndb_binlog_index table (MySQL Cluster replication), 2352, 2359
 - (see also MySQL Cluster replication)
- ndb_config, 2176, 2195
 - config-file option, 2197
 - configinfo option, 2200
 - config_from_node option, 2197
 - connections option, 2199
 - fields option, 2200
 - host option, 2198
 - id option, 2199
 - mycnf option, 2198
 - ndb-connectstring option, 2197
 - nodeid option, 2199
 - nodes option, 2199
 - query option, 2198, 2198
 - rows option, 2200
 - system option, 2199
 - type option, 2199
 - usage option, 2197
 - version option, 2197
 - xml option, 2201
- ndb_cpcd, 2176, 2203
- ndb_delete_all, 2176, 2203
 - transactional option, 2204
- ndb_desc, 2176, 2204
 - blob-info option, 2207
 - extra-node-info option, 2207
 - extra-partition-info option, 2207
- ndb_dist_priv.sql, 2327
- ndb_drop_index, 2176, 2208
- ndb_drop_table, 2176, 2209
- ndb_error_reporter, 2176, 2209
- ndb_index_stat, 2176, 2210
- ndb_log_empty_epochs system variable, 2160
- ndb_log_transaction_id system variable, 2154
- ndb_mgm, 2176, 2194 (see mgm)
- ndb_mgm (MySQL Cluster management node client), 2017
- ndb_mgmd, 2176 (see mgmd)
 - mycnf option, 2189
- ndb_mgmd (MySQL Cluster process), 2186
- ndb_mgmd (MySQL Cluster)
 - defined, 1972
 - (see also management node (MySQL Cluster))
- ndb_print_backup_file, 2176, 2215
- ndb_print_schema_file, 2176, 2215
- ndb_print_sys_file, 2176, 2215
- ndb_replication system table, 2373
- ndb_restore, 2217
 - and circular replication, 2369
 - append option, 2224
 - attribute promotion, 2221
 - backup_path option, 2222
 - disable-indexes option, 2228
 - dont_ignore_systab_0 option, 2224
 - errors, 2229
 - exclude-databases option, 2226
 - exclude-missing-columns option, 2227
 - exclude-tables option, 2226
 - fields-enclosed-by option, 2223
 - fields-optionally-enclosed-by option, 2223
 - fields-terminated-by option, 2223, 2224
 - hex option, 2224
 - include-databases option, 2225
 - include-tables option, 2225
 - lossy-conversions option, 2221
 - ndb-nodegroup-map option, 2225
 - no-binlog option, 2225
 - no-restore-disk-objects option, 2225
 - no-upgrade option, 2223
 - parallelism option, 2225
 - print option, 2224
 - print_data option, 2223
 - print_log option, 2224
 - print_meta option, 2224

progress-frequency option, 2225
 rebuild-indexes option, 2228
 restore-privilege-tables option, 2222
 rewrite-database option, 2228
 skip-broken-objects option, 2228
 skip-unknown-objects option, 2228
 tab option, 2223
 verbose option, 2225
 ndb_schema table (MySQL Cluster replication), 2354
 (see also MySQL Cluster replication)
 ndb_select_all, 2176, 2229
 database option, 2230
 delimiter option, 2231
 descending option, 2231
 disk option, 2231
 gci option, 2231
 header option, 2231
 lock option, 2230
 nodata option, 2231
 order option, 2230
 parallelism option, 2230
 rowid option, 2231
 tupscan option, 2231
 useHexFormat option, 2231
 ndb_select_count, 2176, 2232
 ndb_show_tables, 2176, 2233
 database option, 2234
 loops option, 2234
 parsable option, 2234
 show-temp-status option, 2234
 type option, 2234
 unqualified option, 2234
 ndb_size.pl, 2176, 2234
 ndb_waiter, 2176, 2237
 no-contact option, 2238
 not-started option, 2238
 nowait-nodes option, 2238
 single-user option, 2238
 timeout option, 2238
 wait-nodes option, 2239
 negative values, 940
 neighbor page, 4152
 nested queries, 1377
 Nested-Loop join algorithm, 917
 nested-loop join algorithm, 920
 net etiquette, 19
 NetBeans, 3004
 netmask notation
 in account names, 705
 network ports
 and MySQL Cluster, 2306
 net_buffer_length system variable, 531
 net_buffer_length variable, 283
 net_read_timeout system variable, 532
 net_retry_count system variable, 532
 net_write_timeout system variable, 533
 new features in MySQL 5.5, 9
 new features in MySQL Cluster, 1979
 new system variable, 533
 new users
 adding, 165
 newline (\n), 938, 1348
 next-key lock, 4153
 InnoDB, 1561, 1597, 1602, 1603
 NFS
 InnoDB, 1535, 1632
 nice option
 mysqld_safe, 257
 no matching rows, 4061
 no-auto-rehash option
 mysql, 279
 no-autocommit option
 mysqldump, 318
 no-beep option
 mysql, 279
 mysqladmin, 299
 no-binlog option (ndb_restore), 2225
 no-contact option
 ndb_waiter, 2238
 no-create-db option
 mysqldump, 318
 no-create-info option
 mysqldump, 318
 no-data option
 mysqldump, 319
 no-defaults option, 248
 mysqld_multi, 261
 mysqld_safe, 257
 mysql_plugin, 268
 my_print_defaults, 396
 no-log option
 mysqld_multi, 262
 no-named-commands option
 mysql, 279
 no-nodeid-checks option (ndb_mgmd), 2187
 no-pager option
 mysql, 279
 no-restore-disk-objects option (ndb_restore), 2225
 no-set-names option
 mysqldump, 319
 no-symlinks option
 myisamchk, 351
 no-tablespaces option
 mysqldump, 319
 no-tee option
 mysql, 279
 no-upgrade option (ndb_restore), 2223
 nodaemon option (ndb_mgmd), 2190

nodata option
 ndb_select_all, 2231
 node groups (MySQL Cluster), 1975
 node logs (MySQL Cluster), 2258
 NodeGroup, 2040
 NodeId, 2035, 2039, 2090
 nodeid option
 ndb_config, 2199
 NodeId1, 2095, 2098, 2101
 NodeId2, 2095, 2099, 2101
 NODERESTART Events (MySQL Cluster), 2262
 nodes
 ndbinfo table, 2295
 nodes option
 ndb_config, 2199
 noindices option
 mysqlhotcopy, 388
 non-blocking I/O, 4153
 non-locking read, 4153
 non-repeatable read, 4153
 nondelimited strings, 941
 Nontransactional tables, 4060
 NoOfDiskPagesToDiskAfterRestartACC (DEPRECATED), 2068
 NoOfDiskPagesToDiskAfterRestartTUP (DEPRECATED), 2068
 NoOfDiskPagesToDiskDuringRestartACC, 2068
 NoOfDiskPagesToDiskDuringRestartTUP, 2068
 NoOfFragmentLogFiles, 2051
 NoOfFragmentLogParts, 2080
 NoOfReplicas, 2041
 nopager command
 mysql, 285
 normalized, 4154
 NoSQL, 4154
 nostart option (ndbd), 2180
 nostart option (ndbmttd), 2180
 NOT
 logical, 1094
 NOT BETWEEN, 1092
 not equal (!=), 1090
 not equal (<>), 1090
 NOT EXISTS
 with subqueries, 1381
 NOT IN, 1093
 NOT LIKE, 1113
 NOT NULL
 constraint, 34
 NOT NULL constraint, 4155
 NOT REGEXP, 1114
 not-started option
 ndb_waiter, 2238
 notee command
 mysql, 285
 Notifier, 85
 NOW(), 1143
 NOWAIT (START BACKUP command), 2250
 nowait option
 MySQLInstallerConsole, 77
 nowait-nodes option
 ndb_waiter, 2238
 nowait-nodes option (ndbd), 2180
 nowait-nodes option (ndbmttd), 2180
 nowait-nodes option (ndb_mgmd), 2191
 nowarning command
 mysql, 286
 NO_AUTO_CREATE_USER SQL mode, 613
 NO_AUTO_VALUE_ON_ZERO SQL mode, 613
 NO_BACKSLASH_ESCAPES SQL mode, 614
 NO_DIR_IN_CREATE SQL mode, 614
 NO_ENGINE_SUBSTITUTION SQL mode, 614
 NO_FIELD_OPTIONS SQL mode, 614
 NO_KEY_OPTIONS SQL mode, 614
 NO_TABLE_OPTIONS SQL mode, 614
 NO_UNSIGNED_SUBTRACTION SQL mode, 614
 NO_ZERO_DATE SQL mode, 615
 NO_ZERO_IN_DATE SQL mode, 615
 NUL, 938, 1348
 NULL, 212, 4058, 4155
 ORDER BY, 927, 1361
 testing for null, 1090, 1091, 1091, 1092, 1098
 thread state, 898
 NULL value, 212, 943
 NULL values
 and AUTO_INCREMENT columns, 4059
 and indexes, 1295
 and TIMESTAMP columns, 4059
 vs. empty values, 4058
 NULLIF(), 1099
 Numa, 2078
 number-char-cols option
 mysqlslap, 339
 number-int-cols option
 mysqlslap, 339
 number-of-queries option
 mysqlslap, 339
 numbers, 940
 NUMERIC data type, 1038
 numeric precision, 1035
 numeric scale, 1036
 numeric types, 1070
 numeric-dump-file option
 resolve_stack_dump, 396
 NumGeometries(), 1231
 NumInteriorRings(), 1230
 NumPoints(), 1228
 NVARCHAR data type, 1042

O

- obtaining information about partitions, 2422
- OCT(), 1106
- OCTET_LENGTH(), 1106
- ODBC, 2567
- ODBC compatibility, 554, 946, 1039, 1084, 1091, 1293, 1368
- ODBC.NET, 2628
- ODBC_INCLUDES= option
 - CMake, 153
- ODBC_LIB_DIR option
 - CMake, 153
- ODirect, 2059
- off-page column, 4155
- offset option
 - mysqlbinlog, 375
- OLAP, 1207
- old system variable, 534
- old-alter-table option
 - mysqld, 455
- old-style-user-limits option
 - mysqld, 455
- old_alter_table system variable, 534
- OLD_PASSWORD(), 1187
- old_passwords system variable, 534
- old_server option
 - mysqlaccess, 369
 - mysqlhotcopy, 389
- OLTP, 4156
- ON DUPLICATE KEY UPDATE, 1334
- one-database option
 - mysql, 279
- one-thread option
 - mysqld, 456
- one_shot system variable, 535
- online, 4156
- online DDL, 4156
- online location of manual, 2
- online upgrades and downgrades (MySQL Cluster), 2255
 - order of node updates, 2258
- only-print option
 - mysqlslap, 340
- ONLY_FULL_GROUP_BY
 - SQL mode, 1209
- ONLY_FULL_GROUP_BY SQL mode, 615
- OPEN, 1428
- Open Source
 - defined, 5
- open tables, 296, 820
- open-files-limit option
 - mysqld, 456
 - mysqld_safe, 257
- OpenGIS, 1211
- opening
 - tables, 820
- Opening master dump table
 - thread state, 906
- Opening mysql.ndb_apply_status
 - thread state, 906
- Opening table
 - thread state, 898
- Opening tables
 - thread state, 898
- opens, 295
- OpenSSL, 745, 746
- open_files_limit system variable, 535
- open_files_limit variable, 378
- operating systems
 - file-size limits, 4085
 - supported, 48
- operations
 - arithmetic, 1121
- operators, 1076
 - assignment, 956, 1095
 - cast, 1121, 1168
 - logical, 1094
 - precedence, 1087
- opt option
 - mysqldump, 319
- optimistic, 4157
- optimization, 796
 - benchmarking, 890
 - BLOB types, 819
 - buffering and caching, 848
 - character and string types, 819
 - data size, 817
 - DELETE statements, 803
 - disk I/O, 875
 - DML statements, 802
 - foreign keys, 811
 - full table scans, 801
 - indexes, 810
 - INFORMATION_SCHEMA queries, 804
 - InnoDB tables, 823
 - INSERT statements, 803
 - internal details, 907
 - locking, 862
 - many tables, 820
 - MEMORY tables, 833
 - memory usage, 879
 - MyISAM tables, 828
 - network usage, 882
 - numeric types, 819
 - PERFORMANCE_SCHEMA, 892
 - primary keys, 811
 - privileges, 804
 - SELECT statements, 798

- SQL statements, 798
- subquery, 932
- tips, 809
- UPDATE statements, 803
- WHERE clauses, 799
- optimizations, 910
 - LIMIT clause, 801
- optimize option
 - mysqlcheck, 305
- OPTIMIZE TABLE, 1461
 - and partitioning, 2421
- optimizer, 4157
 - and replication, 1954
 - controlling, 846
 - query plan evaluation, 846
 - switchable optimizations, 847
- optimizer statistics estimation, 1679, 1681
- optimizer_prune_level system variable, 535
- optimizer_search_depth system variable, 536
- optimizer_switch system variable, 536, 847
- optimizing
 - DISTINCT, 931
 - filesort, 928
 - GROUP BY, 929
 - LEFT JOIN, 916
 - server configuration, 868
 - tables, 793
 - thread state, 898
- option, 4158
- option file, 4158
- option files, 243, 712
 - escape sequences, 246
- option prefix
 - disable, 243
 - enable, 243
 - loose, 243
 - maximum, 243
 - skip, 243
- options
 - boolean, 243
 - CMake, 147
 - command-line
 - mysql, 273
 - mysqladmin, 296
 - embedded server, 3067
 - libmysqld, 3067
 - myisamchk, 347
 - provided by MySQL, 197
 - replication, 1937
- OR, 226, 910
 - bitwise, 1182
 - logical, 1095
- OR Index Merge optimization, 910
- Oracle compatibility, 27, 1206, 1257, 1519

- ORACLE SQL mode, 618
- ORD(), 1106
- ORDER BY, 209, 1258, 1361
 - NULL, 927, 1361
- order option
 - ndb_select_all, 2230
- order-by-primary option
 - mysqldump, 319
- Out of resources error
 - and partitioned tables, 2430
- out-of-range handling, 1047
- OUTFILE, 1364
- out_dir option
 - comp_err, 265
- out_file option
 - comp_err, 265
- overflow handling, 1047
- overflow page, 4158
- Overlaps(), 1234
- OverloadLimit, 2096, 2103
- overview, 1

P

- packages
 - list of, 42
- PAD_CHAR_TO_FULL_LENGTH SQL mode, 616
- page, 4158
- page cleaner, 4159
- page size, 4159
 - InnoDB, 1610, 1633
- pager command
 - mysql, 286
- pager option
 - mysql, 280
- PAM
 - pluggable authentication, 729
- PAM authentication
 - with Connector/J, 2939
- parallel-recover option
 - myisamchk, 351
- parallelism option (ndb_restore), 2225
- parameters
 - server, 868
- PARAMETERS
 - INFORMATION_SCHEMA table, 2481
- parent table, 4160
- parentheses (and), 1088
- parsable option
 - ndb_show_tables, 2234
- partial backup, 4160
- partial index, 4160
- partial trust, 2781
- partial updates

- and replication, 1955
- PARTITION, 2381
- PARTITION BY LIST COLUMNS, 2393
- PARTITION BY RANGE COLUMNS, 2393
- partition management, 2414
- partition option
 - mysqld, 456
- partition pruning, 2424
- partitioning, 2381
 - advantages, 2385
 - and dates, 2386
 - and foreign keys, 2431
 - and FULLTEXT indexes, 2431
 - and key cache, 2430
 - and query cache, 2430
 - and replication, 1955
 - and SQL mode, 1955, 2428
 - and subqueries, 2432
 - and temporary tables, 2431, 2433
 - by hash, 2401
 - by key, 2404
 - by linear hash, 2403
 - by linear key, 2406
 - by list, 2391
 - by range, 2387
 - COLUMNS, 2393
 - concepts, 2383
 - data type of partitioning key, 2431
 - enabling, 2381
 - functions supported in partitioning expressions, 2437
 - limitations, 2427
 - operators not permitted in partitioning expressions, 2428
 - operators supported in partitioning expressions, 2428
 - optimization, 2423, 2424
 - resources, 2382
 - storage engines (limitations), 2437
 - subpartitioning, 2432
 - support, 2381
 - support in MySQL Cluster, 1988
 - types, 2385
- Partitioning
 - maximum number of partitions, 2430
- partitioning information statements, 2422
- partitioning keys and primary keys, 2433
- partitioning keys and unique keys, 2433
- partitions
 - adding and dropping, 2414
 - analyzing, 2421
 - checking, 2421
 - managing, 2414
 - modifying, 2414
 - optimizing, 2421
 - repairing, 2421
 - splitting and merging, 2414
 - truncating, 2414
- PARTITIONS
 - INFORMATION_SCHEMA table, 2481
- partitions (MySQL Cluster), 1975
- password
 - root user, 171
- password encryption
 - reversibility of, 1188
- password option, 239
 - mysql, 280
 - mysqlaccess, 369
 - mysqladmin, 299
 - mysqlbinlog, 375
 - mysqlcheck, 305
 - mysqldump, 319
 - mysqld_multi, 262
 - mysqlhotcopy, 388
 - mysqlimport, 328
 - mysqlshow, 333
 - mysqslap, 340
 - mysql_convert_table_format, 390
 - mysql_setpermission, 392
- PASSWORD(), 706, 723, 1188, 4046
- passwords
 - administrator guidelines, 681
 - for users, 716
 - forgotten, 4048
 - hashing, 681
 - logging, 681
 - lost, 4048
 - resetting, 4048
 - security, 679, 692
 - setting, 723, 1451, 1456
 - user guidelines, 679
- PATH environment variable, 161, 193, 237
- path name separators
 - Windows, 246
- pattern matching, 213, 1114
- PEP-249 specification for Python, 3034
- performance, 796
 - benchmarks, 892
 - disk I/O, 875
 - estimating, 846
- Performance Schema, 1676, 2519, 4160
 - event filtering, 2531
 - memory use, 2528
- performance_schema
 - cond_instances table, 2543
 - events_waits_current table, 2546
 - events_waits_history table, 2548
 - events_waits_history_long table, 2548
 - events_waits_summary_by_instance table, 2549

events_waits_summary_by_thread_by_event_name table, 2549
 events_waits_summary_global_by_event_name table, 2549
 file_instances table, 2544
 file_summary_by_event_name table, 2550
 file_summary_by_instance table, 2550
 mutex_instances table, 2544
 performance_timers table, 2551
 rlock_instances table, 2545
 setup_consumers table, 2541
 setup_instruments table, 2541
 setup_timers table, 2542
 threads table, 2552
 performance_schema database, 2519
 restrictions, 4082
 TRUNCATE TABLE, 2540, 4082
 PERFORMANCE_SCHEMA storage engine, 2519
 performance_schema system variable, 2554
 performance_schema_events_waits_history_long_size system variable, 2554
 performance_schema_events_waits_history_size system variable, 2554
 performance_schema_max_cond_classes system variable, 2554
 performance_schema_max_cond_instances system variable, 2555
 performance_schema_max_file_classes system variable, 2555
 performance_schema_max_file_handles system variable, 2555
 performance_schema_max_file_instances system variable, 2556
 performance_schema_max_mutex_classes system variable, 2556
 performance_schema_max_mutex_instances system variable, 2556
 performance_schema_max_rlock_classes system variable, 2557
 performance_schema_max_rlock_instances system variable, 2557
 performance_schema_max_table_handles system variable, 2557
 performance_schema_max_table_instances system variable, 2558
 performance_schema_max_thread_classes system variable, 2558
 performance_schema_max_thread_instances system variable, 2558
 performance_timers table
 performance_schema, 2551
 PERIOD_ADD(), 1144
 PERIOD_DIFF(), 1144
 Perl
 installing, 194
 installing on Windows, 195
 Perl API, 3789
 Perl DBI/DBD
 installation problems, 195
 permission checks
 effect on speed, 804
 perror, 236, 397
 help option, 397
 ndb option, 397
 silent option, 397
 verbose option, 397
 version option, 397
 persistent statistics, 4160
 pessimistic, 4160
 phantom, 4161
 phantom rows, 1603
 PHP API, 3185
 physical, 4161
 physical backup, 4161
 PI(), 1127
 pid-file option
 mysql.server, 260
 mysqld, 456
 mysqld_safe, 257
 pid_file system variable, 537
 Ping
 thread command, 894
 pipe option, 239
 mysql, 280, 305
 mysqladmin, 299
 mysqldump, 319
 mysqlimport, 329
 mysqlshow, 333
 mysqlslap, 340
 PIPES_AS_CONCAT SQL mode, 616
 PITR, 4162
 plan option
 mysqlaccess, 369
 plan stability, 4162
 pluggable authentication
 PAM, 729
 restrictions, 4082
 Windows, 736
 plugin, 4162
 audit log, 757
 plugin API, 618, 3795
 plugin option prefix
 mysqld, 457
 plugin services, 3840
 plugin-dir option
 mysql, 280
 mysqladmin, 299
 mysqlbinlog, 375

- mysqlcheck, 306
- mysqldump, 319
- mysqlimport, 329
- mysqlshow, 333
- mysqslap, 340
- mysql_plugin, 268
- mysql_upgrade, 272
- plugin-ini option
 - mysql_plugin, 268
- plugin-load option
 - mysqld, 457
- plugindir option
 - mysql_config, 394
- plugins
 - activating, 619
 - adding, 3795
 - audit, 3799
 - authentication, 3800
 - daemon, 3799
 - full-text parser, 3798
 - INFORMATION_SCHEMA, 3799
 - installing, 619, 1466
 - semisynchronous replication, 3799
 - server, 618
 - storage engine, 3798
 - uninstalling, 619, 1467
- PLUGINS
 - INFORMATION_SCHEMA table, 2484
- plugin_dir system variable, 537
- POINT data type, 1220
- Point(), 1223
- point-in-time recovery, 786, 4162
 - using MySQL Cluster replication, 2365
- PointFromText(), 1221
- PointFromWKB(), 1222
- PointN(), 1228
- PointOnSurface(), 1230
- PolyFromText(), 1221
- PolyFromWKB(), 1222
- POLYGON data type, 1220
- Polygon(), 1223
- PolygonFromText(), 1221
- PolygonFromWKB(), 1222
- port option, 239
 - mysql, 280
 - mysqladmin, 299
 - mysqlbinlog, 376
 - mysqlcheck, 306
 - mysqld, 458
 - mysqldump, 320
 - mysqld_safe, 257
 - mysqlhotcopy, 389
 - mysqlimport, 329
 - mysqlshow, 333
 - mysqslap, 340
 - mysql_config, 395
 - mysql_convert_table_format, 390
 - mysql_setpermission, 392
- port system variable, 538
- port-open-timeout option
 - mysqld, 458
- portability, 798
 - types, 1073
- porting
 - to other systems, 3854
- PortNumber, 2035
- PortNumber (OBSOLETE), 2097
- ports, 678
- position option
 - mysqlbinlog, 376
- POSITION(), 1107
- post-filtering
 - Performance Schema, 2531
- post-query option
 - mysqslap, 340
- post-system option
 - mysqslap, 340
- PostgreSQL compatibility, 28
- POSTGRESQL SQL mode, 618
- postinstall
 - multiple servers, 650
- postinstallation
 - setup and testing, 159
- POW(), 1128
- POWER(), 1128
- pre-filtering
 - Performance Schema, 2531
- pre-query option
 - mysqslap, 340
- pre-system option
 - mysqslap, 341
- precedence
 - operator, 1087
- precision
 - arithmetic, 1237
 - numeric, 1035
- precision math, 1237
- preload_buffer_size system variable, 538
- Prepare
 - thread command, 894
- PREPARE, 1415, 1418
 - XA transactions, 1404
- prepared backup, 4163
- prepared statements, 1415, 1418, 1418, 1419, 3132
 - reparation, 1419
 - with Connector/Net, 2754
- preparing
 - thread state, 898

preserve-trailing-spaces option (ndb_restore), 2221
preview option
 mysqlaccess, 369
primary key, 4163
 constraint, 34
 deleting, 1258
PRIMARY KEY, 1258, 1294
primary keys
 and partitioning keys, 2433
print command
 mysql, 286
print option (ndb_restore), 2224
print-defaults option, 248
 mysql_plugin, 268
print-full-config option (ndb_mgmd), 2190
print_data option (ndb_restore), 2223
print_log option (ndb_restore), 2224
print_meta option (ndb_restore), 2224
privilege
 changes, 710
privilege checks
 effect on speed, 804
privilege information
 location, 698
privilege system, 692
privileges
 access, 692
 adding, 718
 and replication, 1954
 default, 171
 deleting, 721, 1444
 display, 1487
 dropping, 721, 1444
 granting, 1445
 revoking, 1455
problems
 access denied errors, 4035
 common errors, 4033
 compiling, 158
 DATE columns, 4057
 date values, 1050
 installing on Solaris, 135
 installing Perl, 195
 linking, 3183
 lost connection errors, 4038
 ODBC, 2650
 reporting, 2, 20
 starting the server, 169
 table locking, 864
 time zone, 4055
PROCEDURE, 1363
PROCEDURE ANALYSE(), 820
procedures
 stored, 2443
process, 4163
process management (MySQL Cluster), 2176
process support, 48
processes
 display, 1492
processing
 arguments, 3847
Processing events
 thread state, 906
Processing events from schema table
 thread state, 906
Processlist
 thread command, 894
PROCESSLIST, 1492
 INFORMATION_SCHEMA table, 2486
 possible inconsistency with INFORMATION_SCHEMA
 tables, 1665
product option
 MySQLInstallerConsole, 77
PROFILING
 INFORMATION_SCHEMA table, 2486
profiling session variable, 539
profiling_history_size session variable, 539
program options (MySQL Cluster), 2240
program variables
 setting, 248
program-development utilities, 235
programs
 administrative, 234
 client, 233, 3183
 stored, 1419, 2441
 utility, 234
progress-frequency option (ndb_restore), 2225
promote-attributes option (ndb_restore), 2221
prompt command
 mysql, 286
prompt option
 mysql, 280
prompts
 meanings, 200
pronunciation
 MySQL, 6
protocol option, 239
 mysql, 280
 mysqladmin, 299
 mysqlbinlog, 376
 mysqlcheck, 306
 mysqldump, 320
 mysqlexport, 329
 mysqlshow, 333
 mysqlslap, 341
protocol_version system variable, 539
Proxy, 1799
proxy_user session variable, 539

pseudo-record, 4163
pseudo_thread_id system variable, 539
Pthreads, 4164
purge, 4164
PURGE BINARY LOGS, 1406
purge buffering, 4164
purge lag, 4164
PURGE MASTER LOGS, 1406
purge scheduling, 1677
purge thread, 4164
Purging old relay logs
 thread state, 898
Python, 3034
 third-party driver, 3790

Q

QUARTER(), 1144
queries
 entering, 198
 estimating performance, 846
 examples, 221
 speed of, 798
Query
 thread command, 894
query, 4164
Query Cache, 855
query cache
 and ndbinfo database tables, 2284
 and partitioned tables, 2430
 thread states, 903
query end
 thread state, 898
query execution plan, 833, 4165
query option
 mysqlslap, 341
 ndb_config, 2198, 2198
query option (ndb_index_stat), 2212
query_alloc_block_size system variable, 539
query_cache_limit system variable, 540
query_cache_min_res_unit system variable, 541
query_cache_size system variable, 541
query_cache_type system variable, 542
query_cache_wlock_invalidate system variable, 542
query_prealloc_size system variable, 543
questions, 295
 answering, 19
Queueing master event to the relay log
 thread state, 905
quick option
 myisamchk, 351
 mysql, 280
 mysqlcheck, 306
 mysqldump, 320

quiesce, 4165
quiet option
 mysqlhotcopy, 389
Quit
 thread command, 895
quit command
 mysql, 286
QUIT command (MySQL Cluster), 2248
quotation marks
 in strings, 939
QUOTE(), 939, 1107, 3119
quote-names option
 mysqldump, 320
quoting, 939
 column alias, 944, 4059
quoting binary data, 939
quoting of identifiers, 943

R

RADIANS(), 1128
RAID, 4165
RAND(), 1128
random dive, 4166
rand_seed1 session variable, 543
rand_seed2 session variable, 543
range join type
 optimizer, 838
range partitioning, 2387, 2393
range partitions
 adding and dropping, 2415
 managing, 2415
range_alloc_block_size system variable, 544
raw backup, 4166
raw option
 mysql, 280
re-creating
 grant tables, 166
read ahead, 1683
READ COMMITTED, 4166
 transaction isolation level, 1402
READ UNCOMMITTED, 4167
 transaction isolation level, 1402
read view, 4167
read-ahead, 4167
 linear, 1670
 random, 1670
read-from-remote-server option
 mysqlbinlog, 376
read-only option
 myisamchk, 350
 mysqld, 1873
read-only transaction, 4168
Reading event from the relay log

- thread state, 905
- Reading from net
 - thread state, 898
- Reading master dump table data
 - thread state, 906
- read_buffer_size myisamchk variable, 347
- read_buffer_size system variable, 544
- read_only system variable, 544
- read_rnd_buffer_size system variable, 545
- REAL data type, 1039
- RealtimeScheduler, 2077
- REAL_AS_FLOAT SQL mode, 616
- rebuild-indexes option (ndb_restore), 2228
- Rebuilding the index on master dump table
 - thread state, 906
- ReceiveBufferMemory, 2097
- reconfiguring, 158
- reconnect option
 - mysql, 281
- Reconnecting after a failed binlog dump request
 - thread state, 904
- Reconnecting after a failed master event read
 - thread state, 905
- reconnection
 - automatic, 3174
- record lock, 4168
- record-level locks
 - InnoDB, 1561, 1597, 1602, 1603
- record_log_pos option
 - mysqlhotcopy, 389
- RECOVER
 - XA transactions, 1404
- recover option
 - myisamchk, 351
- recovery
 - from crash, 789
 - incremental, 786
 - point in time, 786
- redo, 4168
- redo log, 4168
- RedoBuffer, 2070
- RedoOverCommitCounter
 - data nodes, 2088
- RedoOverCommitLimit
 - data nodes, 2088
- reducing
 - data size, 817
- redundant row format, 4169
- ref join type
 - optimizer, 837
- references, 1259
- referential integrity, 1529, 1531, 4169
- REFERENTIAL_CONSTRAINTS
 - INFORMATION_SCHEMA table, 2487
- Refresh
 - thread command, 895
- ref_or_null, 915
- ref_or_null join type
 - optimizer, 838
- REGEXP, 1114
- REGEXP operator, 1114
- regexp option
 - mysqlhotcopy, 389
 - mysql_find_rows, 391
- Register Slave
 - thread command, 895
- Registering slave on master
 - thread state, 904
- regular expression syntax, 1114
- rehash command
 - mysql, 286
- relational, 4169
- relational databases
 - defined, 5
- relative option
 - mysqladmin, 299
- relay-log option
 - mysqld, 1873
- relay-log-index option
 - mysqld, 1874
- relay-log-info-file option
 - mysqld, 1874
- relay-log-purge option
 - mysqld, 1874
- relay-log-recovery option
 - mysqld, 1875
- relay-log-space-limit option
 - mysqld, 1875
- relay_log system variable, 1885
- relay_log_index system variable, 1885
- relay_log_info_file system variable, 1886
- relay_log_purge system variable, 546
- relay_log_recovery system variable, 1886
- relay_log_space_limit system variable, 546
- release notes, 4069
- release numbers, 49
- RELEASE SAVEPOINT, 1393
- releases
 - naming scheme, 49
 - testing, 50
 - updating, 51
- RELEASE_LOCK(), 1201
- relevance, 4170
- relnotes option
 - mysqlaccess, 369
- reload option (ndb_mgmd), 2191
- remote administration (MySQL Cluster)
 - and security issues, 2306

- remove option
 - mysqld, 458
- remove option (ndbd), 2181
- remove option (ndbmtd), 2181
- remove option (ndb_mgmd), 2194
- Removing duplicates
 - thread state, 898
- removing tmp table
 - thread state, 898
- rename
 - thread state, 899
- rename result table
 - thread state, 899
- RENAME TABLE, 1325
- RENAME USER, 1455
- renaming user accounts, 1455
- Reopen tables
 - thread state, 899
- repair
 - tables, 300
- Repair by sorting
 - thread state, 899
- Repair done
 - thread state, 899
- repair option
 - mysqlcheck, 306
- repair options
 - myisamchk, 350
- REPAIR TABLE, 1463
 - and partitioning, 2421
 - and replication, 1465, 1952
- Repair with keycache
 - thread state, 899
- repairing
 - tables, 791
- REPEAT, 1426
 - labels, 1420
- REPEAT(), 1107
- REPEATABLE READ, 4170
 - transaction isolation level, 1401
- repertoire
 - character set, 987, 995
- replace, 236
- REPLACE, 1357
- replace option
 - mysqldump, 320
 - mysqlexport, 329
- replace utility, 397
- REPLACE(), 1107
- replicas (MySQL Cluster), 1975
- replicate-do-db option
 - mysqld, 1875
- replicate-do-table option
 - mysqld, 1878
- replicate-ignore-db option
 - mysqld, 1877
- replicate-ignore-table option
 - mysqld, 1878
- replicate-rewrite-db option
 - mysqld, 1878
- replicate-same-server-id option
 - mysqld, 1879
- replicate-wild-do-table option
 - mysqld, 1879
- replicate-wild-ignore-table option
 - mysqld, 1880
- replication, 1839, 4171
 - and AUTO_INCREMENT, 1938
 - and character sets, 1938
 - and CREATE ... IF NOT EXISTS, 1939
 - and CREATE TABLE ... SELECT, 1939
 - and DATA DIRECTORY, 1947
 - and DROP ... IF EXISTS, 1941
 - and errors on slave, 1955
 - and floating-point values, 1948
 - and FLUSH, 1949
 - and functions, 1949
 - and INDEX DIRECTORY, 1947
 - and invoked features, 1947
 - and LAST_INSERT_ID(), 1938
 - and LIMIT, 1951
 - and LOAD DATA, 1951
 - and max_allowed_packet, 1952
 - and MEMORY tables, 1953
 - and mysql (system) database, 1954
 - and partial updates, 1955
 - and partitioning, 1955
 - and privileges, 1954
 - and query optimizer, 1954
 - and REPAIR TABLE statement, 1465, 1952
 - and reserved words, 1954
 - and scheduled events, 1947, 1947
 - and slow query log, 1952
 - and SQL mode, 1955
 - and stored routines, 1947
 - and temporary tables, 1953
 - and time zones, 1956
 - and TIMESTAMP, 181, 1938, 1956
 - and transactions, 1956, 1956
 - and triggers, 1947, 1958
 - and TRUNCATE TABLE, 1958
 - and variables, 1959
 - and views, 1958
 - attribute demotion, 1943
 - attribute promotion, 1943
 - between MySQL server versions, 181, 1956
 - circular, 2346
 - crashes, 1952

- in MySQL Cluster, 2342
 - (see also MySQL Cluster replication)
- row-based vs statement-based, 1851
- safe and unsafe statements, 1856
- semisynchronous, 1932
- shutdown and restart, 1952, 1953
- statements incompatible with STATEMENT format, 1852
- timeouts, 1956
 - with differing tables on master and slave, 1941
 - with ZFS, 1734
- replication filtering options
 - and case sensitivity, 1913
- replication formats
 - compared, 1851
- replication implementation, 1907
- replication limitations, 1937
- replication master
 - thread states, 903
- replication masters
 - statements, 1406
- replication options, 1937
- replication slave
 - thread states, 904, 905, 905
- replication slaves
 - statements, 1408
- replication, asynchronous (see MySQL Cluster replication)
- REPORT command (MySQL Cluster), 2247
- report-host option
 - mysqld, 1880
- report-password option
 - mysqld, 1880
- report-port option
 - mysqld, 1881
- report-user option
 - mysqld, 1881
- reporting
 - bugs, 2, 20
 - Connector/Net problems, 2901
 - Connector/ODBC problems, 2650, 2650
 - errors, 20
 - problems, 2
- report_host system variable, 547
- report_password system variable, 547
- report_port system variable, 547
- report_user system variable, 548
- Requesting binlog dump
 - thread state, 904
- REQUIRE GRANT option, 1452
- reschedule
 - thread state, 903
- reserved words, 953
 - and replication, 1954
- ReservedSendBufferMemory, 2088
- RESET MASTER, 1407
- RESET SLAVE, 1412
- RESET SLAVE ALL, 1412
- Reset stmt
 - thread command, 895
- reset-slave.pl (see MySQL Cluster replication)
- resetmaster option
 - mysqlhotcopy, 389
- resetslave option
 - mysqlhotcopy, 389
- RESIGNAL, 1434
- resolveip, 236, 398
 - help option, 398
 - silent option, 399
 - version option, 399
- resolve_stack_dump, 235, 396
 - help option, 396
 - numeric-dump-file option, 396
 - symbols-file option, 396
 - version option, 396
- resource limits
 - user accounts, 525, 721, 1452
- resources
 - ndbinfo table, 2296
- RESTART command (MySQL Cluster), 2247
- restarting
 - the server, 163
- RestartOnErrorInsert, 2059
- restore, 4171
- restore-privilege-tables option (ndb_restore), 2222
- restore_connect option (ndb_restore), 2220
- restore_data option (ndb_restore), 2222
- restore_epoch option (ndb_restore), 2222
- restore_meta option (ndb_restore), 2222
- restore_nodeid option (ndb_restore), 2220
- restore_skip-table-check option (ndb_restore), 2220
- restoring backups
 - in MySQL Cluster, 2217
- restoring from backup
 - in MySQL Cluster replication, 2360
- restrictions
 - character sets, 4081
 - events, 4073
 - InnoDB, 1632
 - performance_schema database, 4082
 - pluggable authentication, 4082
 - server-side cursors, 4077
 - signal, 4077
 - stored routines, 4073
 - subqueries, 4077
 - triggers, 4073
 - views, 4079
 - XA transactions, 4081

result-file option
 mysqlbinlog, 376
 mysqldump, 320
 retrieving
 data from tables, 206
 RETURN, 1426
 return (\r), 938, 1348
 return values
 UDFs, 3849
 REVERSE(), 1107
 REVOKE, 1455
 revoking
 privileges, 1455
 rewrite-database option (ndb_restore), 2228
 rhost option
 mysqlaccess, 369
 RIGHT JOIN, 1366
 RIGHT OUTER JOIN, 1366
 RIGHT(), 1107
 RLIKE, 1114
 ROLLBACK, 29, 1390
 XA transactions, 1404
 rollback, 4171
 rollback option
 mysqlaccess, 369
 rollback segment, 4171
 ROLLBACK TO SAVEPOINT, 1393
 Rolling back
 thread state, 899
 rolling restart (MySQL Cluster), 2255
 ROLLUP, 1207
 root password, 171
 root user, 678
 password resetting, 4048
 ROUND(), 1129
 rounding, 1237
 rounding errors, 1038
 ROUTINES
 INFORMATION_SCHEMA table, 2488
 routines option
 mysqldump, 320
 ROW, 1380
 row, 4171
 row format, 4172
 row lock, 4172
 row subqueries, 1380
 row-based replication, 4173
 advantages, 1853
 disadvantages, 1854
 row-level locking, 862, 4173
 rowid option
 ndb_select_all, 2231
 rows
 counting, 215
 deleting, 4060
 locking, 31
 matching problems, 4061
 selecting, 206
 sorting, 209
 rows option
 mysql_find_rows, 391
 ndb_config, 2200
 ROW_COUNT(), 1196
 ROW_FORMAT
 COMPACT, 1657
 COMPRESSED, 1642, 1657
 DYNAMIC, 1657
 REDUNDANT, 1657
 RPAD(), 1107
 Rpl_semi_sync_master_clients status variable, 605
 rpl_semi_sync_master_enabled system variable, 548
 Rpl_semi_sync_master_net_avg_wait_time status variable, 605
 Rpl_semi_sync_master_net_waits status variable, 606
 Rpl_semi_sync_master_net_wait_time status variable, 605
 Rpl_semi_sync_master_no_times status variable, 606
 Rpl_semi_sync_master_no_tx status variable, 606
 Rpl_semi_sync_master_status status variable, 606
 Rpl_semi_sync_master_timefunc_failures status variable, 606
 rpl_semi_sync_master_timeout system variable, 548
 rpl_semi_sync_master_trace_level system variable, 549
 Rpl_semi_sync_master_tx_avg_wait_time status variable, 606
 Rpl_semi_sync_master_tx_waits status variable, 606
 Rpl_semi_sync_master_tx_wait_time status variable, 606
 rpl_semi_sync_master_wait_no_slave system variable, 549
 Rpl_semi_sync_master_wait_pos_backtraverse status variable, 606
 Rpl_semi_sync_master_wait_sessions status variable, 607
 Rpl_semi_sync_master_yes_tx status variable, 607
 rpl_semi_sync_slave_enabled system variable, 550
 Rpl_semi_sync_slave_status status variable, 607
 rpl_semi_sync_slave_trace_level system variable, 550
 Rpl_status status variable, 607
 RPM file, 127
 rpm option
 mysql_install_db, 266
 RPM Package Manager, 127
 RTRIM(), 1108
 Ruby API, 3790
 running
 ANSI mode, 25
 batch mode, 220
 multiple servers, 650

- queries, 198
- running CMake after prior invocation, 144, 158
- rw-lock, 4173
- rwlock_instances table
 - performance_schema, 2545

S

- safe statement (replication)
 - defined, 1856
- safe-mode option
 - mysqld, 459
- safe-recover option
 - myisamchk, 351
- safe-show-database option
 - mysqld, 459
- safe-updates option, 292
 - mysql, 281
- safe-user-create option
 - mysqld, 459
- Sakila, 9
- same value wins (conflict resolution), 2374
- SAVEPOINT, 1393
- savepoint, 4173
- Saving state
 - thread state, 899
- scalability, 4174
- scale
 - arithmetic, 1237
 - numeric, 1036
- SchedulerExecutionTimer, 2077
- SchedulerSpinTimer, 2077
- schema, 4174
 - altering, 1248
 - creating, 1272
 - deleting, 1321
- SCHEMA(), 1197
- SCHEMATA
 - INFORMATION_SCHEMA table, 2489
- SCHEMA_PRIVILEGES
 - INFORMATION_SCHEMA table, 2490
- SCI (Scalable Coherent Interface) (see MySQL Cluster)
- script files, 220
- scripts, 254, 260
 - mysql_install_db, 165
 - SQL, 272
- search index, 4174
- searching
 - and case sensitivity, 4055
 - full-text, 1153
 - MySQL Web pages, 20
 - two keys, 226
- Searching rows for update
 - thread state, 899

- SECOND(), 1144
- secondary index, 4174
 - InnoDB, 1610
- secure-auth option
 - mysql, 281
 - mysqld, 460
- secure-file-priv option
 - mysqld, 460
- secure_auth system variable, 550
- secure_file_priv system variable, 550
- securing a MySQL Cluster, 2308
- security
 - against attackers, 687
 - and malicious SQL statements, 2307
 - and NDB utilities, 2309
- security system, 692
- SEC_TO_TIME(), 1144
- segment, 4175
- SELECT
 - INTO, 1364
 - LIMIT, 1358
 - optimizing, 833, 1519
 - Query Cache, 855
- SELECT INTO TABLE, 29
- SELECT speed, 799
- selecting
 - databases, 202
- selectivity, 4175
- select_limit variable, 283
- semi-consistent read, 4176
 - InnoDB, 1561
- semisynchronous replication, 1932
 - administrative interface, 1934
 - configuration, 1935
 - installation, 1935
 - monitoring, 1936
- semisynchronous replication plugins, 3799
- SendBufferMemory, 2096
- Sending binlog event to slave
 - thread state, 903
- sending cached result to client
 - thread state, 903
- SendLimit, 2102
- SendSignalId, 2097, 2100, 2103
- SEQUENCE, 226
- sequence emulation, 1196
- sequences, 226
- SERIAL, 1036, 1037
- SERIAL DEFAULT VALUE, 1069
- SERIALIZABLE, 4176
 - transaction isolation level, 1402
- server, 4176
 - connecting, 197, 237
 - debugging, 3854

- disconnecting, 197
- logs, 629
- restart, 163
- shutdown, 163
- signal handling, 627
- starting, 160
- starting and stopping, 167
- starting problems, 169
- server administration, 293
- server plugins, 618
- server variable (see system variable)
- server variables (see system variables)
- server-id option
 - mysqlbinlog, 376
 - mysqld, 1858
- server-id-bits option
 - mysqlbinlog, 376
 - mysqld, 2144
- server-side cursor
 - restrictions, 4077
- ServerPort, 2040
- servers
 - multiple, 650
- server_id system variable, 551
- server_id_bits system variable, 2160
- server_operations
 - ndbinfo table, 2297
- server_transactions
 - ndbinfo table, 2298
- service-startup-timeout option
 - mysql.server, 260
- services
 - for plugins, 3840
- servlet, 2973
- session variable
 - autocommit, 484
 - big_tables, 485
 - error_count, 498
 - external_user, 499
 - foreign_key_checks, 500
 - identity, 505
 - insert_id, 506
 - last_insert_id, 511
 - profiling, 539
 - profiling_history_size, 539
 - proxy_user, 539
 - rand_seed1, 543
 - rand_seed2, 543
 - sql_auto_is_null, 554
 - sql_big_selects, 555
 - sql_buffer_result, 555
 - sql_notes, 556
 - sql_quote_show_create, 556
 - sql_safe_updates, 557
 - sql_warnings, 557
 - timestamp, 567
 - transaction_allow_batching, 2158
 - unique_checks, 571
 - warning_count, 573
- session variables
 - and replication, 1959
- SESSION_STATUS
 - INFORMATION_SCHEMA table, 2479
- SESSION_USER(), 1197
- SESSION_VARIABLES
 - INFORMATION_SCHEMA table, 2479
- SET, 1468
 - CHARACTER SET, 973, 1470
 - NAMES, 973, 976, 1470
 - ONE_SHOT, 1470
 - size, 1072
- SET data type, 1044, 1065
- SET GLOBAL sql_slave_skip_counter, 1413
- Set option
 - thread command, 895
- SET OPTION, 1468
- SET PASSWORD, 1456
- SET PASSWORD statement, 723
- SET sql_log_bin, 1408
- SET statement
 - assignment operator, 1096
- SET TRANSACTION, 1400
- set-auto-increment[option
 - myisamchk, 352
- set-character-set option
 - myisamchk, 351
- set-charset option
 - mysqlbinlog, 376
 - mysqldump, 320
- set-collation option
 - myisamchk, 351
- setting
 - passwords, 723
- setting passwords, 1456
- setting program variables, 248
- setup
 - postinstallation, 160
 - thread state, 899
- setup_consumers table
 - performance_schema, 2541
- setup_instruments table
 - performance_schema, 2541
- setup_timers table
 - performance_schema, 2542
- SHA(), 1189
- SHA1(), 1189
- SHA2(), 1189
- shared lock, 4176

shared memory transporter (see MySQL Cluster)
 shared tablespace, 4177
 shared-memory option
 mysqld, 460
 shared-memory-base-name option, 240
 mysqld, 460
 mysqslap, 340
 SharedBufferSize, 2102
 SharedGlobalMemory, 2083
 shared_memory system variable, 551
 shared_memory_base_name system variable, 551
 sharp checkpoint, 4177
 shell syntax, 4
 ShmKey, 2099
 ShmSize, 2099
 short-form option
 mysqlbinlog, 376
 SHOW
 in MySQL Cluster management client, 2025
 SHOW AUTHORS, 1471, 1472
 SHOW BINARY LOGS, 1471, 1472
 SHOW BINLOG EVENTS, 1471, 1472
 SHOW CHARACTER SET, 1471, 1473
 SHOW COLLATION, 1471, 1473
 SHOW COLUMNS, 1471, 1474
 SHOW command (MySQL Cluster), 2246
 SHOW CONTRIBUTORS, 1471, 1476
 SHOW CREATE DATABASE, 1471, 1476
 SHOW CREATE EVENT, 1471
 SHOW CREATE FUNCTION, 1471, 1477
 SHOW CREATE PROCEDURE, 1471, 1477
 SHOW CREATE SCHEMA, 1471, 1476
 SHOW CREATE TABLE, 1471, 1477
 SHOW CREATE TRIGGER, 1471, 1478
 SHOW CREATE VIEW, 1471, 1478
 SHOW DATABASES, 1471, 1479
 SHOW ENGINE, 1471, 1479
 and MySQL Cluster, 2279
 SHOW ENGINE INNODB MUTEX, 1683
 SHOW ENGINE INNODB STATUS, 1479
 and innodb_adaptive_hash_index, 1669
 and innodb_use_sys_malloc, 1667
 SHOW ENGINE NDB STATUS, 1479, 2279
 SHOW ENGINE NDBCLUSTER STATUS, 1479, 2279
 SHOW ENGINES, 1471, 1483
 and MySQL Cluster, 2279
 SHOW ERRORS, 1471, 1485
 SHOW EVENTS, 1471, 1485
 SHOW extensions, 2516
 SHOW FIELDS, 1471, 1475
 SHOW FUNCTION CODE, 1471, 1487
 SHOW FUNCTION STATUS, 1471, 1487
 SHOW GRANTS, 1471, 1487
 SHOW INDEX, 1471, 1487
 SHOW KEYS, 1471, 1487
 SHOW MASTER LOGS, 1471, 1472
 SHOW MASTER STATUS, 1471, 1489
 SHOW OPEN TABLES, 1471, 1489
 SHOW PLUGINS, 1471, 1490
 SHOW PRIVILEGES, 1471, 1491
 SHOW PROCEDURE CODE, 1471, 1491
 SHOW PROCEDURE STATUS, 1471, 1492
 SHOW PROCESSLIST, 1471, 1492
 SHOW PROFILE, 1471, 1494
 SHOW PROFILES, 1471, 1494, 1497
 SHOW RELAYLOG EVENTS, 1497
 SHOW SCHEDULER STATUS, 2453
 SHOW SCHEMAS, 1479
 SHOW SLAVE HOSTS, 1471, 1497
 SHOW SLAVE STATUS, 1471, 1498
 SHOW STATUS, 1471
 and MySQL Cluster, 2281
 SHOW STORAGE ENGINES, 1483
 SHOW TABLE STATUS, 1471
 SHOW TABLES, 1471, 1506
 SHOW TRIGGERS, 1471, 1506
 SHOW VARIABLES, 1471
 and MySQL Cluster, 2280
 SHOW WARNINGS, 1471, 1509
 SHOW with WHERE, 2469, 2516
 show-slave-auth-info option
 mysqld, 1882
 show-table-type option
 mysqlshow, 333
 show-temp-status option
 ndb_show_tables, 2234
 show-warnings option
 mysql, 281
 showing
 database information, 330
 Shutdown
 thread command, 895
 shutdown, 4177
 SHUTDOWN command (MySQL Cluster), 2248
 shutdown_timeout variable, 300
 shutting down
 the server, 163
 Shutting down
 thread state, 906
 sigint-ignore option
 mysql, 281
 SIGN(), 1130
 SIGNAL, 1438
 signal
 restrictions, 4077
 signals
 server response, 627
 SigNum, 2100

silent column changes, 1311
 silent option
 myisamchk, 347
 myisampack, 362
 mysql, 281
 mysqladmin, 299
 mysqlcheck, 306
 mysqld_multi, 262
 mysqlimport, 329
 mysqlslap, 341
 perror, 397
 resolveip, 399
 SIN(), 1130
 single quote (\'), 938
 single user mode (MySQL Cluster), 2248, 2278
 and ndb_restore, 2217
 single-transaction option
 mysqldump, 321
 single-user option
 ndb_waiter, 2238
 size of tables, 4085
 sizes
 display, 1035
 --skip option prefix, 243
 skip-broken-objects option (ndb_restore), 2228
 skip-column-names option
 mysql, 282
 skip-comments option
 mysqldump, 321
 skip-concurrent-insert option
 mysqld, 460
 skip-event-scheduler option
 mysqld, 461
 skip-external-locking option
 mysqld, 460
 skip-grant-tables option
 mysqld, 461
 skip-host-cache option
 mysqld, 461
 skip-innodb option
 mysqld, 461, 1547
 skip-kill-mysqld option
 mysqld_safe, 257
 skip-line-numbers option
 mysql, 282
 skip-name-resolve option
 mysqld, 461
 mysql_install_db, 266
 skip-ndbcluster option
 mysqld, 2145
 skip-networking option
 mysqld, 461
 skip-opt option
 mysqldump, 321
 mysqld, 462
 skip-safemalloc option
 mysqld, 462
 skip-show-database option
 mysqld, 463
 skip-slave-start option
 mysqld, 1882
 skip-ssl option, 749
 skip-stack-trace option
 mysqld, 463
 skip-symbolic-links option
 mysqld, 462
 skip-syslog option
 mysqld_safe, 258
 skip-thread-priority option
 mysqld, 463
 skip-unknown-objects option (ndb_restore), 2228
 skip-use-db option
 mysql_find_rows, 391
 skip_external_locking system variable, 552
 skip_name_resolve system variable, 552
 skip_networking system variable, 552
 skip_show_database system variable, 552
 Slave has read all relay log; waiting for the slave I/O
 thread to update it
 thread state, 905
 slave server, 4177
 slave-load-tmpdir option
 mysqld, 1882
 slave-max-allowed-packet (mysqld), 1869
 slave-net-timeout option
 mysqld, 1883
 slave-skip-errors option
 mysqld, 1883
 slave_allow_batching, 2357
 slave_compressed_protocol option
 mysqld, 1882
 slave_compressed_protocol system variable, 1887
 slave_exec_mode system variable, 1887
 slave_load_tmpdir system variable, 1887
 slave_max_allowed_packet system variable, 1869
 slave_net_timeout system variable, 1888
 slave_skip_errors system variable, 1888
 slave_transaction_retries system variable, 1889
 slave_type_conversions system variable, 1889
 Sleep
 thread command, 895
 sleep option
 mysqladmin, 299
 SLEEP(), 1201
 slow queries, 295
 slow query log, 647, 4178
 and replication, 1952

- slow shutdown, 4178
- slow-query-log option
 - mysqld, 463
- slow-start-timeout option
 - mysqld, 464
- slow_launch_time system variable, 552
- slow_query_log system variable, 552
- slow_query_log_file system variable, 553
- SMALLINT data type, 1037
- snapshot, 4178
- SNAPSHOTEND (START BACKUP command), 2251
- SNAPSHOTSTART (START BACKUP command), 2251
- socket option, 240
 - mysql, 282
 - mysqldadmin, 299
 - mysqlbinlog, 376
 - mysqlcheck, 306
 - mysqld, 464
 - mysqldump, 321
 - mysqld_safe, 258
 - mysqlhotcopy, 389
 - mysqlimport, 329
 - mysqlshow, 333
 - mysqslap, 341
 - mysql_config, 395
 - mysql_convert_table_format, 390
 - mysql_setpermission, 392
- socket system variable, 553
- Solaris
 - installation, 135
- Solaris installation problems, 135
- Solaris troubleshooting, 159
- Solaris x86_64 issues, 826
- SOME, 1379
- sort-index option
 - myisamchk, 352
- sort-records option
 - myisamchk, 352
- sort-recover option
 - myisamchk, 351
- sorting
 - data, 209
 - grant tables, 707, 709
 - table rows, 209
- Sorting for group
 - thread state, 899
- Sorting for order
 - thread state, 899
- Sorting index
 - thread state, 900
- Sorting result
 - thread state, 900
- sort_buffer_size myisamchk variable, 347
- sort_buffer_size system variable, 554
- sort_key_blocks myisamchk variable, 347
- SOUNDEX(), 1108
- SOUNDS LIKE, 1108
- source (mysql client command), 221, 290
- source command
 - mysql, 286
- source distribution
 - installing, 140
- space ID, 4178
- SPACE(), 1108
- spassword option
 - mysqlaccess, 369
- Spatial Extensions in MySQL, 1211
- speed
 - increasing with replication, 1839
 - inserting, 803
 - of queries, 798, 799
- spin, 4179
- sporadic-binlog-dump-fail option
 - mysqld, 1897
- Spring framework, 2964
- SQL, 4179
 - defined, 5
- SQL Editor, 2694
- SQL mode, 611
 - ALLOW_INVALID_DATES, 612
 - and partitioning, 1955, 2428
 - and replication, 1955
 - ANSI, 612, 617
 - ANSI_QUOTES, 612
 - DB2, 618
 - ERROR_FOR_DIVISION_BY_ZERO, 613
 - HIGH_NOT_PRECEDENCE, 613
 - IGNORE_SPACE, 613
 - MAXDB, 618
 - MSSQL, 618
 - MYSQL323, 618
 - MYSQL40, 618
 - NO_AUTO_CREATE_USER, 613
 - NO_AUTO_VALUE_ON_ZERO, 613
 - NO_BACKSLASH_ESCAPES, 614
 - NO_DIR_IN_CREATE, 614
 - NO_ENGINE_SUBSTITUTION, 614
 - NO_FIELD_OPTIONS, 614
 - NO_KEY_OPTIONS, 614
 - NO_TABLE_OPTIONS, 614
 - NO_UNSIGNED_SUBTRACTION, 614
 - NO_ZERO_DATE, 615
 - NO_ZERO_IN_DATE, 615
 - ONLY_FULL_GROUP_BY, 615, 1209
 - ORACLE, 618
 - PAD_CHAR_TO_FULL_LENGTH, 616
 - PIPES_AS_CONCAT, 616
 - POSTGRESQL, 618

- REAL_AS_FLOAT, 616
- strict, 612
- STRICT_ALL_TABLES, 616
- STRICT_TRANS_TABLES, 612, 617
- TRADITIONAL, 612, 618
- SQL node (MySQL Cluster)
 - defined, 1972
- SQL nodes (MySQL Cluster), 2254
- SQL scripts, 272
- SQL statements
 - replication masters, 1406
 - replication slaves, 1408
- SQL statements relating to MySQL Cluster, 2279
- SQL-92
 - extensions to, 24
- sql-mode option
 - mysqld, 464
- SQLState error codes, 2941
- sql_auto_is_null session variable, 554
- SQL_BIG_RESULT, 1364
- sql_big_selects session variable, 555
- SQL_BUFFER_RESULT, 1364
- sql_buffer_result session variable, 555
- SQL_CACHE, 858, 1364
- SQL_CALC_FOUND_ROWS, 1364
- sql_log_bin system variable, 555
- sql_log_off system variable, 556
- sql_log_update system variable, 556
- sql_mode system variable, 556
- sql_notes session variable, 556
- SQL_NO_CACHE, 858, 1364
- sql_quote_show_create session variable, 556
- sql_safe_updates session variable, 557
- sql_select_limit system variable, 557
- sql_slave_skip_counter, 1413
- sql_slave_skip_counter system variable, 1890
- SQL_SMALL_RESULT, 1364
- sql_warnings session variable, 557
- SQRT(), 1130
- square brackets, 1036
- srcdir option
 - mysql_install_db, 266
- SRID(), 1226
- SSD, 1642, 4179
- SSH, 756
- SSL, 745, 2784, 2937
 - command options, 749
 - configuring, 746
 - establishing connections, 747
 - X509 Basics, 746
- SSL certificates, 2784
- ssl option, 749
- SSL options, 240
 - mysql, 282
 - mysqladmin, 299
 - mysqlcheck, 306
 - mysqld, 462
 - mysqldump, 321
 - mysqlimport, 329
 - mysqlshow, 334
 - mysqslap, 341
- SSL related options, 1452
- ssl-ca option, 750
- ssl-capath option, 750
- ssl-cert option, 750
- ssl-cipher option, 750
- ssl-key option, 751
- ssl-verify-server-cert option, 751
- ssl_ca system variable, 557
- ssl_capath system variable, 557
- ssl_cert system variable, 558
- ssl_cipher system variable, 558
- ssl_key system variable, 558
- standalone option
 - mysqld, 462
- Standard Monitor
 - InnoDB, 1620
- Standard SQL
 - differences from, 29, 1454
 - extensions to, 24, 26
- standards compatibility, 24
- START
 - XA transactions, 1404
- START BACKUP
 - NOWAIT, 2250
 - SNAPSHOTEND, 2251
 - SNAPSHOTSTART, 2251
 - syntax, 2250
 - WAIT COMPLETED, 2251
 - WAIT STARTED, 2251
- START command (MySQL Cluster), 2246
- START SLAVE, 1413
- START TRANSACTION, 1390
- start-datetime option
 - mysqlbinlog, 377
- start-position option
 - mysqlbinlog, 377
- StartFailRetryDelay, 2089
- StartFailureTimeout, 2061
- starting
 - comments, 32
 - mysqld, 690
 - the server, 160
 - the server automatically, 167
- Starting many servers, 650
- StartNoNodeGroupTimeout, 2062
- StartPartialTimeout, 2061
- StartPartitionedTimeout, 2061

- StartPoint(), 1228
- startup, 4179
- STARTUP Events (MySQL Cluster), 2262
- startup options
 - default, 243
- startup parameters, 868
 - mysql, 273
 - mysqladmin, 296
 - tuning, 868
- StartupStatusReportFrequency, 2073
- start_row option
 - mysql_find_rows, 391
- statefile option
 - comp_err, 265
- statement-based replication, 4180
 - advantages, 1852
 - disadvantages, 1852
 - unsafe statements, 1852
- statements
 - compound, 1419
 - GRANT, 718
 - INSERT, 719
 - replication masters, 1406
 - replication slaves, 1408
- Statistics
 - thread command, 895
- statistics, 4180
 - thread state, 900
- STATISTICS
 - INFORMATION_SCHEMA table, 2490
- STATISTICS Events (MySQL Cluster), 2264
- stats option
 - myisam_ftdump, 343
- stats_method myisamchk variable, 347
- status
 - tables, 1504
- status command
 - mysql, 286
 - results, 295
- STATUS command (MySQL Cluster), 2247
- status option
 - mysqlshow, 334
- status variable
 - Rpl_semi_sync_master_clients, 605
 - Rpl_semi_sync_master_net_avg_wait_time, 605
 - Rpl_semi_sync_master_net_waits, 606
 - Rpl_semi_sync_master_net_wait_time, 605
 - Rpl_semi_sync_master_no_times, 606
 - Rpl_semi_sync_master_no_tx, 606
 - Rpl_semi_sync_master_status, 606
 - Rpl_semi_sync_master_timefunc_failures, 606
 - Rpl_semi_sync_master_tx_avg_wait_time, 606
 - Rpl_semi_sync_master_tx_waits, 606
 - Rpl_semi_sync_master_tx_wait_time, 606
 - Rpl_semi_sync_master_wait_pos_backtraverse, 606
 - Rpl_semi_sync_master_wait_sessions, 607
 - Rpl_semi_sync_master_yes_tx, 607
 - Rpl_semi_sync_slave_status, 607
 - Rpl_status, 607
- status variables, 585, 1503
 - MySQL Cluster, 2161
- STD(), 1206
- STDDEV(), 1206
- STDDEV_POP(), 1206
- STDDEV_SAMP(), 1206
- stemming, 4181
- STOP command (MySQL Cluster), 2247
- STOP SLAVE, 1414
- stop-datetime option
 - mysqlbinlog, 377
- stop-position option
 - mysqlbinlog, 377
- StopOnError, 2058
- stopping
 - the server, 167
- stopword, 4181
- stopword list
 - user-defined, 1165
- storage engine, 4181
 - ARCHIVE, 1705
 - InnoDB, 1529
 - PERFORMANCE_SCHEMA, 2519
- storage engine plugins, 3798
- storage engines
 - and application feature requirements, 1985
 - applications supported, 1984
 - availability, 1982
 - choosing, 1524
 - differences between NDB and InnoDB, 1982
 - InnoDB as default, 1531
 - usage scenarios, 1985
- storage nodes - see data nodes, ndbd (see data nodes, ndbd)
- storage nodes - see data nodes, ndbd, ndbmtd (see data nodes, ndbd, ndbmtd)
- storage requirements
 - data type, 1069
- storage space
 - minimizing, 817
- storage_engine system variable, 559
- stored functions, 2443
 - and INSERT DELAYED, 1337
- stored procedures, 2443
 - calling from Connector/Net, 2705
 - editing with Visual Studio, 2673
 - with Connector/Net, 2755
- stored programs, 1419, 2441
- stored routines

- and replication, 1947
- LAST_INSERT_ID(), 2445
- metadata, 2445
- restrictions, 4073
- storing result in query cache
 - thread state, 903
- storing row into queue
 - thread state, 902
- STRAIGHT_JOIN, 833, 844, 916, 916, 1363, 1366
- STRCMP(), 1113
- strict mode, 1681, 4181
- strict SQL mode, 612
- STRICT_ALL_TABLES SQL mode, 616
- STRICT_TRANS_TABLES SQL mode, 612, 617
- string collating, 1017
- string comparison functions, 1111
- string comparisons
 - case sensitivity, 1111
- string concatenation, 937, 1102
- string functions, 1099
- string literal introducer, 938, 970
- string replacement
 - replace utility, 397
- string types, 1058, 1071
- StringMemory, 2044
- strings
 - defined, 937
 - escape sequences, 937
 - nondelimited, 941
- striping
 - defined, 876
- STR_TO_DATE(), 1144
- SUBDATE(), 1145
- sublist, 4181
- SUBPARTITION BY KEY
 - known issues, 2432
- subpartitioning, 2406
- subpartitions, 2406
 - known issues, 2432
- subqueries, 1377
 - correlated, 1382
 - errors, 1384
 - optimization, 932
 - restrictions, 4077
 - rewriting as joins, 1387
 - with ALL, 1380
 - with ANY, IN, SOME, 1379
 - with EXISTS, 1381
 - with NOT EXISTS, 1381
 - with ROW, 1380
- subquery (see subqueries)
- subselects, 1377
- SUBSTR(), 1109
- SUBSTRING(), 1109
- SUBSTRING_INDEX(), 1109
- SUBTIME(), 1145
- subtraction (-), 1122
- suffix option
 - mysqlhotcopy, 389
- SUM(), 1206
- SUM(DISTINCT), 1206
- super-large-pages option
 - mysqld, 462
- superuser, 171
- superuser option
 - mysqlaccess, 369
- support
 - for operating systems, 48
- suppression
 - default values, 34
- supremum record, 4182
- surrogate key, 4182
- symbolic links, 876, 878
- symbolic-links option
 - mysqld, 462
- symbols-file option
 - resolve_stack_dump, 396
- SymDifference(), 1232
- synchronization of master and slave
 - in MySQL Cluster Replication, 2363
- Syncing ndb table schema operation and binlog
 - thread state, 906
- sync_binlog system variable, 1904
- sync_frm system variable, 559
- sync_master_info system variable, 1890
- sync_relay_log system variable, 1890
- sync_relay_log_info system variable, 1891
- syntax
 - regular expression, 1114
- syntax conventions, 3
- synthetic key, 4182
- sys-check option (ndb_index_stat), 2213
- sys-create option (ndb_index_stat), 2213
- sys-create-if-not-exist option (ndb_index_stat), 2213
- sys-create-if-not-valid option (ndb_index_stat), 2213
- sys-drop option (ndb_index_stat), 2213
- sys-skip-events option (ndb_index_stat), 2214
- sys-skip-tables option (ndb_index_stat), 2214
- SYSCONFDIR option
 - CMake, 153
- SYSDATE(), 1146
- sysdate-is-now option
 - mysqld, 465
- syslog option
 - mysqld_safe, 258
- syslog-tag option
 - mysqld_safe, 258
- system

privilege, 692
security, 678
system command
mysql, 286
System lock
thread state, 900
system optimization, 868
system option
ndb_config, 2199
system table
optimizer, 837, 1363
system tablespace, 4182
system variable, 573, 1507
and replication, 1959
audit_log_buffer_size, 764
audit_log_file, 765
audit_log_flush, 765
audit_log_policy, 766
audit_log_rotate_on_size, 766
audit_log_strategy, 767
authentication_windows_log_level, 482
authentication_windows_use_principal_name, 483
automatic_sp_privileges, 484
auto_increment_increment, 1865
auto_increment_offset, 1868
back_log, 485
basedir, 485
binlog_cache_size, 1898
binlog_direct_non_transactional_updates, 1898
binlog_format, 1899
binlog_stmt_cache_size, 1901
bulk_insert_buffer_size, 486
character_sets_dir, 488
character_set_client, 486
character_set_connection, 487
character_set_database, 487
character_set_filesystem, 487
character_set_results, 488
character_set_server, 488
character_set_system, 488
collation_connection, 489
collation_database, 489
collation_server, 489
completion_type, 489
concurrent_insert, 490
connect_timeout, 492
datadir, 492
datetime_format, 492
date_format, 492
debug, 492
debug_sync, 493
default_storage_engine, 494
default_week_format, 494
delayed_insert_limit, 495
delayed_insert_timeout, 496
delayed_queue_size, 496
delay_key_write, 494
div_precision_increment, 497
engine_condition_pushdown, 497
event_scheduler, 498
expire_logs_days, 498
flush, 499
flush_time, 500
ft_boolean_syntax, 500
ft_max_word_len, 501
ft_min_word_len, 501
ft_query_expansion_limit, 502
ft_stopword_file, 502
general_log, 502
general_log_file, 503
group_concat_max_len, 503
have_compress, 504
have_crypt, 504
have_csv, 504
have_dynamic_loading, 504
have_geometry, 504
have_innodb, 504
have_openssl, 504
have_partitioning, 504
have_profiling, 504
have_query_cache, 505
have_rtree_keys, 505
have_ssl, 505
have_symlink, 505
hostname, 505
ignore_builtin_innodb, 1547
init_connect, 505
init_file, 506
init_slave, 1885
innodb_adaptive_flushing, 1547
innodb_adaptive_hash_index, 1547
innodb_additional_mem_pool_size, 1548
innodb_autoextend_increment, 1548
innodb_autoinc_lock_mode, 1548
innodb_buffer_pool_instances, 1549
innodb_buffer_pool_size, 1549
innodb_change_buffering, 1550
innodb_checksums, 1551
innodb_commit_concurrency, 1551
innodb_concurrency_tickets, 1552
innodb_data_file_path, 1552
innodb_data_home_dir, 1552
innodb_doublewrite, 1553
innodb_fast_shutdown, 1553
innodb_file_format, 1554
innodb_file_format_check, 1554
innodb_file_format_max, 1555
innodb_file_per_table, 1555

innodb_flush_log_at_trx_commit, 1556
innodb_flush_method, 1557
innodb_force_recovery, 1558
innodb_io_capacity, 1559
innodb_locks_unsafe_for_binlog, 1561
innodb_lock_wait_timeout, 1560
innodb_log_buffer_size, 1563
innodb_log_files_in_group, 1564
innodb_log_file_size, 1564
innodb_log_group_home_dir, 1564
innodb_max_dirty_pages_pct, 1565
innodb_max_purge_lag, 1565
innodb_mirrored_log_groups, 1566
innodb_old_blocks_pct, 1566
innodb_old_blocks_time, 1566
innodb_open_files, 1567
innodb_purge_batch_size, 1567
innodb_purge_threads, 1567
innodb_read_ahead_threshold, 1568
innodb_read_io_threads, 1568
innodb_replication_delay, 1569
innodb_rollback_on_timeout, 1569
innodb_spin_wait_delay, 1570
innodb_stats_method, 1570
innodb_stats_on_metadata, 1571
innodb_stats_sample_pages, 1571
innodb_strict_mode, 1572
innodb_support_xa, 1572
innodb_sync_spin_loops, 1573
innodb_table_locks, 1573
innodb_thread_concurrency, 1573
innodb_thread_sleep_delay, 1574
innodb_use_native_aio, 1574
innodb_use_sys_malloc, 1575
innodb_version, 1575
innodb_write_io_threads, 1575
interactive_timeout, 506
join_buffer_size, 507
keep_files_on_create, 507
key_buffer_size, 508
key_cache_age_threshold, 509
key_cache_block_size, 509
key_cache_division_limit, 510
language, 510
large_files_support, 511
large_pages, 511
large_page_size, 511
lc_messages, 512
lc_messages_dir, 512
lc_time_names, 512
license, 513
local_infile, 513
locked_in_memory, 514
lock_wait_timeout, 513
log, 514
log_bin, 514
log_bin_trust_function_creators, 514
log_bin_use_v1_row_events, 1901
log_error, 515
log_output, 515
log_queries_not_using_indexes, 515
log_slave_updates, 1897
log_slow_queries, 516
log_warnings, 516
long_query_time, 517
lower_case_file_system, 518
lower_case_table_names, 518
low_priority_updates, 517
max_allowed_packet, 519
max_binlog_cache_size, 1902
max_binlog_size, 1903
max_binlog_stmt_cache_size, 1903
max_connections, 519
max_connect_errors, 519
max_delayed_threads, 520
max_error_count, 520
max_heap_table_size, 521
max_insert_delayed_threads, 521
max_join_size, 522
max_length_for_sort_data, 522
max_long_data_size, 523
max_prepared_stmt_count, 523
max_relay_log_size, 523
max_seeks_for_key, 524
max_sort_length, 524
max_sp_recursion_depth, 525
max_tmp_tables, 525
max_user_connections, 525
max_write_lock_count, 526
metadata_locks_cache_size, 526
min_examined_row_limit, 527
myisam_data_pointer_size, 527
myisam_max_sort_file_size, 528
myisam_mmap_size, 528
myisam_recover_options, 529
myisam_repair_threads, 529
myisam_sort_buffer_size, 530
myisam_stats_method, 530
myisam_use_mmap, 531
named_pipe, 531
ndb_log_empty_epochs, 2160
ndb_log_transaction_id, 2154
net_buffer_length, 531
net_read_timeout, 532
net_retry_count, 532
net_write_timeout, 533
new, 533
old, 534

old_alter_table, 534
old_passwords, 534
one_shot, 535
open_files_limit, 535
optimizer_prune_level, 535
optimizer_search_depth, 536
optimizer_switch, 536
performance_schema, 2554
performance_schema_events_waits_history_long_size, 2554
performance_schema_events_waits_history_size, 2554
performance_schema_max_cond_classes, 2554
performance_schema_max_cond_instances, 2555
performance_schema_max_file_classes, 2555
performance_schema_max_file_handles, 2555
performance_schema_max_file_instances, 2556
performance_schema_max_mutex_classes, 2556
performance_schema_max_mutex_instances, 2556
performance_schema_max_rwlock_classes, 2557
performance_schema_max_rwlock_instances, 2557
performance_schema_max_table_handles, 2557
performance_schema_max_table_instances, 2558
performance_schema_max_thread_classes, 2558
performance_schema_max_thread_instances, 2558
pid_file, 537
plugin_dir, 537
port, 538
preload_buffer_size, 538
protocol_version, 539
pseudo_thread_id, 539
query_alloc_block_size, 539
query_cache_limit, 540
query_cache_min_res_unit, 541
query_cache_size, 541
query_cache_type, 542
query_cache_wlock_invalidate, 542
query_prealloc_size, 543
range_alloc_block_size, 544
read_buffer_size, 544
read_only, 544
read_rnd_buffer_size, 545
relay_log, 1885
relay_log_index, 1885
relay_log_info_file, 1886
relay_log_purge, 546
relay_log_recovery, 1886
relay_log_space_limit, 546
report_host, 547
report_password, 547
report_port, 547
report_user, 548
rpl_semi_sync_master_enabled, 548
rpl_semi_sync_master_timeout, 548
rpl_semi_sync_master_trace_level, 549
rpl_semi_sync_master_wait_no_slave, 549
rpl_semi_sync_slave_enabled, 550
rpl_semi_sync_slave_trace_level, 550
secure_auth, 550
secure_file_priv, 550
server_id, 551
server_id_bits, 2160
shared_memory, 551
shared_memory_base_name, 551
skip_external_locking, 552
skip_name_resolve, 552
skip_networking, 552
skip_show_database, 552
slave_compressed_protocol, 1887
slave_exec_mode, 1887
slave_load_tmpdir, 1887
slave_max_allowed_packet, 1869
slave_net_timeout, 1888
slave_skip_errors, 1888
slave_transaction_retries, 1889
slave_type_conversions, 1889
slow_launch_time, 552
slow_query_log, 552
slow_query_log_file, 553
socket, 553
sort_buffer_size, 554
sql_log_bin, 555
sql_log_off, 556
sql_log_update, 556
sql_mode, 556
sql_select_limit, 557
sql_slave_skip_counter, 1890
ssl_ca, 557
ssl_capath, 557
ssl_cert, 558
ssl_cipher, 558
ssl_key, 558
storage_engine, 559
sync_binlog, 1904
sync_frm, 559
sync_master_info, 1890
sync_relay_log, 1890
sync_relay_log_info, 1891
system_time_zone, 560
sysvar_stored_program_cache, 559
table_definition_cache, 560
table_lock_wait_timeout, 561
table_open_cache, 561
table_type, 561
thread_cache_size, 561
thread_concurrency, 562
thread_handling, 562
thread_pool_algorithm, 563

- thread_pool_high_priority_connection, 563
- thread_pool_max_unused_threads, 564
- thread_pool_prio_kickup_timer, 565
- thread_pool_size, 565
- thread_pool_stall_limit, 566
- thread_stack, 566
- timed_mutexes, 567
- time_format, 567
- time_zone, 567
- tmpdir, 568
- tmp_table_size, 568
- transaction_alloc_block_size, 569
- transaction_prealloc_size, 569
- tx_isolation, 570
- updatable_views_with_limit, 571
- version, 572
- version_comment, 572
- version_compile_machine, 572
- version_compile_os, 572
- wait_timeout, 572
- system variables, 467
- system_time_zone system variable, 560
- SYSTEM_USER(), 1197
- sysvar_stored_program_cache system variable, 559

T

- tab (\t), 938, 1348
- tab option
 - mysqldump, 321
- tab option (ndb_restore), 2223
- table, 4183
 - changing, 1253, 1258, 4063
 - deleting, 1323
 - rebuilding, 190
 - repair, 190
 - row size, 1069
- Table 'mysql.proxies_priv' doesn't exist, 179, 185
- table aliases, 1360
- table cache, 820, 2754
- table description
 - myisamchk, 352
- Table Designer, 2665
- Table Dump
 - thread command, 895
- table is full, 485, 4045
- Table is full errors (MySQL Cluster), 2043
- Table lock
 - thread state, 900
- table lock, 4185
- Table Monitor
 - InnoDB, 1620, 1631
- table names
 - case sensitivity, 26, 946

- table option
 - mysql, 282
 - mysqlaccess, 369
- table scan, 1674
- table type, 4185
 - choosing, 1524
- table-level locking, 862
- TableDirect commands, 2784
- tables
 - BLACKHOLE, 1706
 - checking, 349
 - closing, 820
 - compressed, 361
 - compressed format, 1697
 - const, 837
 - constant, 800
 - copying, 1307, 1308
 - counting rows, 215
 - creating, 203
 - CSV, 1703
 - defragment, 1696
 - defragmenting, 794, 1461
 - deleting rows, 4060
 - displaying, 330
 - displaying status, 1504
 - dumping, 307, 386
 - dynamic, 1696
 - error checking, 790
 - EXAMPLE, 1719
 - FEDERATED, 1714
 - flush, 296
 - fragmentation, 1461
 - HEAP, 1699
 - host, 710
 - improving performance, 817
 - information, 352
 - information about, 219
 - InnoDB, 1529
 - loading data, 204
 - maintenance, 300
 - maintenance schedule, 794
 - maximum size, 4085
 - MEMORY, 1699
 - MERGE, 1709
 - merging, 1709
 - multiple, 217
 - MyISAM, 1691
 - names, 943
 - open, 820
 - opening, 820
 - optimizing, 793
 - partitioning, 1709
 - repair, 300
 - repairing, 791

- retrieving data, 206
- selecting columns, 208
- selecting rows, 206
- sorting rows, 209
- symbolic links, 877
- system, 837
- too many, 822
- unique ID for last row, 3173
- TABLES
 - INFORMATION_SCHEMA table, 2491
- tables option
 - mysqlcheck, 306
 - mysqldump, 322
- TABLESPACE
 - INFORMATION_SCHEMA table, 2492
- tablespace, 4185
- tablespace dictionary, 4186
- Tablespace Monitor
 - InnoDB, 1594, 1615, 1620
- table_definition_cache system variable, 560
- table_lock_wait_timeout system variable, 561
- table_open_cache, 820
- table_open_cache system variable, 561
- TABLE_PRIVILEGES
 - INFORMATION_SCHEMA table, 2493
- table_type system variable, 561
- TAN(), 1131
- tar
 - problems on Solaris, 135, 135
- tc-heuristic-recover option
 - mysqld, 465
- Tcl API, 3791
- tcp-ip option
 - mysqld_multi, 262
- TCP/IP, 107, 112
- tee command
 - mysql, 287
- tee option
 - mysql, 282
- temp-pool option
 - mysqld, 465
- temporary file
 - write access, 166
- temporary files, 4054
- temporary table, 4186
- temporary tables
 - and replication, 1953
 - internal, 822
 - problems, 4064
- terminal monitor
 - defined, 197
- test option
 - myisampack, 362
- testing
 - connection to the server, 705
 - installation, 160
 - of MySQL releases, 50
 - postinstallation, 160
 - testing mysqld
 - mysqltest, 3794
- TEXT
 - size, 1071
- text collection, 4187
- TEXT columns
 - default values, 1062
 - indexing, 811, 1295
- TEXT data type, 1043, 1061
- text files
 - importing, 290, 325, 1342
- thread, 4187
- thread cache, 882
- thread command
 - Binlog Dump, 893
 - Change user, 893
 - Close stmt, 893
 - Connect, 893
 - Connect Out, 893
 - Create DB, 893
 - Daemon, 894
 - Debug, 894
 - Delayed insert, 894
 - Drop DB, 894
 - Error, 894
 - Execute, 894
 - Fetch, 894
 - Field List, 894
 - Init DB, 894
 - Kill, 894
 - Long Data, 894
 - Ping, 894
 - Prepare, 894
 - Processlist, 894
 - Query, 894
 - Quit, 895
 - Refresh, 895
 - Register Slave, 895
 - Reset stmt, 895
 - Set option, 895
 - Shutdown, 895
 - Sleep, 895
 - Statistics, 895
 - Table Dump, 895
 - Time, 895
- thread commands, 893
- thread state
 - After create, 895
 - allocating local table, 902
 - Analyzing, 895

- Changing master, 906
- Checking master version, 904
- checking permissions, 895
- checking privileges on cached query, 903
- checking query cache for query, 903
- Checking table, 896
- cleaning up, 896
- Clearing, 907
- closing tables, 896
- Committing events to binlog, 906
- Connecting to master, 904
- converting HEAP to MyISAM, 896
- copy to tmp table, 896
- Copying to group table, 896
- Copying to tmp table, 896
- Copying to tmp table on disk, 896
- Creating delayed handler, 902
- Creating index, 896
- Creating sort index, 896
- creating table, 896
- Creating tmp table, 896
- deleting from main table, 896
- deleting from reference tables, 897
- discard_or_import_tablespace, 897
- end, 897
- executing, 897
- Execution of init_command, 897
- Finished reading one binlog; switching to next binlog, 904
- Flushing tables, 897
- freeing items, 897
- FULLTEXT initialization, 897
- got handler lock, 902
- got old table, 902
- init, 897
- Initialized, 907
- insert, 903
- invalidating query cache entries, 903
- Killed, 897
- Killing slave, 906
- Locked, 897
- logging slow query, 898
- login, 898
- Making temp file, 905
- manage keys, 898
- Master has sent all binlog to slave; waiting for binlog to be updated, 904
- NULL, 898
- Opening master dump table, 906
- Opening mysql.ndb_apply_status, 906
- Opening table, 898
- Opening tables, 898
- optimizing, 898
- preparing, 898
- Processing events, 906
- Processing events from schema table, 906
- Purging old relay logs, 898
- query end, 898
- Queueing master event to the relay log, 905
- Reading event from the relay log, 905
- Reading from net, 898
- Reading master dump table data, 906
- Rebuilding the index on master dump table, 906
- Reconnecting after a failed binlog dump request, 904
- Reconnecting after a failed master event read, 905
- Registering slave on master, 904
- Removing duplicates, 898
- removing tmp table, 898
- rename, 899
- rename result table, 899
- Reopen tables, 899
- Repair by sorting, 899
- Repair done, 899
- Repair with keycache, 899
- Requesting binlog dump, 904
- reschedule, 903
- Rolling back, 899
- Saving state, 899
- Searching rows for update, 899
- Sending binlog event to slave, 903
- sending cached result to client, 903
- setup, 899
- Shutting down, 906
- Slave has read all relay log; waiting for the slave I/O thread to update it, 905
- Sorting for group, 899
- Sorting for order, 899
- Sorting index, 900
- Sorting result, 900
- statistics, 900
- storing result in query cache, 903
- storing row into queue, 902
- Syncing ndb table schema operation and binlog, 906
- System lock, 900
- Table lock, 900
- update, 902
- Updating, 900
- updating main table, 900
- updating reference tables, 900
- upgrading lock, 903
- User lock, 900
- User sleep, 900
- Waiting for all running commits to finish, 900
- Waiting for commit lock, 900
- waiting for delay_list, 902
- Waiting for event from ndbcluster, 906
- Waiting for first event from ndbcluster, 906
- Waiting for global metadata lock, 901

- Waiting for global read lock, 901, 901
- waiting for handler insert, 902
- waiting for handler lock, 902
- waiting for handler open, 902
- Waiting for INSERT, 903
- Waiting for master to send event, 904
- Waiting for master update, 904
- Waiting for ndbcluster binlog update to reach current position, 906
- Waiting for ndbcluster to start, 906
- Waiting for next activation, 907
- Waiting for query cache lock, 903
- Waiting for release of readlock, 901
- Waiting for scheduler to stop, 907
- Waiting for schema epoch, 906
- Waiting for schema metadata lock, 901
- Waiting for slave mutex on exit, 905, 905
- Waiting for stored function metadata lock, 901
- Waiting for stored procedure metadata lock, 901
- Waiting for table, 901
- Waiting for table level lock, 901
- Waiting for table metadata lock, 901
- Waiting for tables, 901
- Waiting for the next event in relay log, 905
- Waiting for the slave SQL thread to free enough relay log space, 905
- Waiting for trigger metadata lock, 901
- Waiting on cond, 901
- Waiting on empty queue, 907
- Waiting to finalize termination, 904
- Waiting to get readlock, 901
- Waiting to reconnect after a failed binlog dump request, 904
- Waiting to reconnect after a failed master event read, 905
- Writing to net, 902
- thread states, 892
 - delayed inserts, 902
 - event scheduler, 906
 - general, 895
 - MySQL Cluster, 906
 - query cache, 903
 - replication master, 903
 - replication slave, 904, 905, 905
- thread support, 48
- threadblocks
 - ndbinfo table, 2299
- ThreadConfig, 2081
- threaded clients, 3184
- ThreadPool (see DiskIOThreadPool)
- threads, 295, 1492, 3793
 - display, 1492
 - monitoring, 892, 1492, 1492, 2486
- threads table
 - performance_schema, 2552
- threadstat
 - ndbinfo table, 2299
- thread_cache_size system variable, 561
- thread_concurrency system variable, 562
- thread_handling system variable, 562
- thread_pool_algorithm system variable, 563
- thread_pool_high_priority_connection system variable, 563
- thread_pool_max_unused_threads system variable, 564
- thread_pool_prio_kickup_timer system variable, 565
- thread_pool_size system variable, 565
- thread_pool_stall_limit system variable, 566
- thread_stack system variable, 566
- Time
 - thread command, 895
- TIME columns
 - with Connector/Net, 2765
- TIME data type, 1040, 1051
- time literals, 940
- time representation
 - Event Scheduler, 2453
- time types, 1071
- time zone problems, 4055
- time zone tables, 269
- time zones
 - and replication, 1956
 - leap seconds, 1030
 - support, 1026
 - upgrading, 1028
- TIME(), 1146
- TimeBetweenEpochs, 2065
- TimeBetweenEpochsTimeout, 2065
- TimeBetweenGlobalCheckpoints, 2065, 2087
- TimeBetweenInactiveTransactionAbortCheck, 2066
- TimeBetweenLocalCheckpoints, 2064
- TimeBetweenWatchDogCheck, 2060
- TimeBetweenWatchDogCheckInitial, 2060
- TIMEDIFF(), 1146
- timed_mutexes system variable, 567
- timeout, 492, 1199, 1341
 - connect_timeout variable, 283, 300
 - shutdown_timeout variable, 300
- timeout option
 - ndb_waiter, 2238
- timeouts (replication), 1956
- TIMESTAMP
 - and logs, 181
 - and NULL values, 4059
 - and replication, 181, 1938, 1956
 - initialization and updating, 1054
- TIMESTAMP data type, 1039, 1050
- timestamp session variable, 567
- TIMESTAMP(), 1147

TIMESTAMPADD(), 1147
 TIMESTAMPDIFF(), 1147
 timezone option
 mysqld_safe, 258
 time_format system variable, 567
 TIME_FORMAT(), 1148
 TIME_TO_SEC(), 1148
 time_zone system variable, 567
 TINYBLOB data type, 1043
 TINYINT data type, 1036
 TINYTEXT data type, 1043
 tips
 optimization, 809
 TMPDIR environment variable, 166, 193, 236, 4054
 tmpdir option
 myisamchk, 351
 myisampack, 362
 mysqld, 466
 mysqlhotcopy, 389
 mysql_upgrade, 272
 tmpdir system variable, 568
 tmp_table_size system variable, 568
 to-last-log option
 mysqlbinlog, 377
 Tomcat application server, 2962
 tools
 command-line, 76, 272
 list of, 43
 mysqld_multi, 260
 mysqld_safe, 254
 torn page, 4187
 TotalSendBufferMemory
 API and SQL nodes, 2093
 data nodes, 2088
 management nodes, 2038
 Touches(), 1234
 TO_DAYS(), 1148
 TO_SECONDS(), 1149
 TPS, 4187
 trace DBI method, 3857
 trace files
 ndbmtd, 2185
 trace files (MySQL Cluster), 2182
 tracing
 with Connector/Net, 2768
 TRADITIONAL SQL mode, 612, 618
 transaction, 4187
 transaction ID, 4188
 transaction isolation level, 1400
 READ COMMITTED, 1402
 READ UNCOMMITTED, 1402
 REPEATABLE READ, 1401
 SERIALIZABLE, 1402
 transaction-isolation option
 mysqld, 465
 transaction-safe tables, 29, 1529
 transactional option
 ndb_delete_all, 2204
 TransactionBufferMemory, 2049
 TransactionDeadlockDetectionTimeout, 2067
 TransactionInactiveTimeout, 2066
 transactions, 1596
 and replication, 1956, 1956
 isolation levels, 1596, 4483
 metadata locking, 866
 support, 29, 1529
 transaction_alloc_block_size system variable, 569
 transaction_allow_batching session variable (MySQL Cluster), 2158
 transaction_prealloc_size system variable, 569
 Translators
 list of, 41
 transportable tablespace, 4188
 transporters
 ndbinfo table, 2300
 trigger
 restrictions, 4073
 trigger, creating, 1313
 trigger, dropping, 1324
 triggers, 1506, 2441, 2445
 and INSERT DELAYED, 1337
 and replication, 1947, 1958
 editing with Visual Studio, 2675
 LAST_INSERT_ID(), 2445
 metadata, 2448
 TRIGGERS
 INFORMATION_SCHEMA table, 2494
 triggers option
 mysqldump, 322
 TRIM(), 1109
 troubleshooting, 4188
 Connector/J, 2975
 Connector/ODBC connection problems, 2596
 FreeBSD, 159
 JDBC, 2907
 JDBC SQLState codes, 2941
 PHP, 3788
 Solaris, 159
 with MySQL Enterprise Monitor, 3863
 TRUE, 940, 942
 testing for, 1090, 1090
 truncate, 4188
 TRUNCATE TABLE, 1326, 1680
 and MySQL Cluster, 1988
 and replication, 1958
 performance_schema database, 2540, 4082
 TRUNCATE(), 1131
 tuning, 796

tuple, 4189
tupscan option
 ndb_select_all, 2231
tutorial, 197
two-phase commit, 4189
TwoPassInitialNodeRestartCopy, 2078
tx_isolation system variable, 570
type codes
 C prepared statement API, 3137
type conversions, 1083, 1089
type option
 MySQLInstallerConsole, 77
 mysql_convert_table_format, 390
 ndb_config, 2199
 ndb_show_tables, 2234
types
 columns, 1035, 1072
 data, 1035
 date, 1071
 Date and Time, 1048
 numeric, 1070
 of tables, 1524
 portability, 1072
 string, 1071
 strings, 1058
 time, 1071
Types namespace, 2891
typographical conventions, 3
TZ environment variable, 193, 4055
tz-utc option
 mysqldump, 322

U

UCASE(), 1110
UCS-2, 963
ucs2 character set, 994
UDFs, 1465, 1466
 compiling, 3850
 defined, 3841
 return values, 3849
ulimit, 4047
UMASK environment variable, 193, 4048
UMASK_DIR environment variable, 193, 4048
unary minus (-), 1122
unbuffered option
 mysql, 282
UNCOMPRESS(), 1189
UNCOMPRESSED_LENGTH(), 1189
undo, 4189
undo log, 4190
undo tablespace, 4190
UndoDataBuffer, 2070
UndoIndexBuffer, 2069
UNHEX(), 1110
Unicode, 963
 with Connector/J, 2935
Unicode Collation Algorithm, 1003
UNINSTALL PLUGIN, 1467
uninstalling plugins, 619, 1467
UNION, 226, 1375
Union(), 1232
UNIQUE, 1258
unique constraint, 4190
unique ID, 3173
unique index, 4190
unique key, 4191
 constraint, 34
unique keys
 and partitioning keys, 2433
unique_checks session variable, 571
unique_subquery join type
 optimizer, 838
Unix, 2567, 2651
 compiling clients on, 3183
UNIX_TIMESTAMP(), 1150
UNKNOWN
 testing for, 1090, 1090
unloading
 tables, 206
UNLOCK TABLES, 1394
unnamed views, 1382
unpack option
 myisamchk, 352
unqualified option
 ndb_show_tables, 2234
unsafe statement (replication)
 defined, 1856
unsafe statements (replication), 1857
UNSIGNED, 1036, 1044
UNTIL, 1426
updatable views, 2459
updatable_views_with_limit system variable, 571
UPDATE, 29, 1388
update
 thread state, 902
update option (ndb_index_stat), 2212
update-state option
 myisamchk, 350
updates option
 MySQLInstallerConsole, 77
UpdateXML(), 1174
updating
 releases of MySQL, 51
Updating
 thread state, 900
updating main table
 thread state, 900

- updating reference tables
 - thread state, 900
- upgrade-system-tables option
 - mysql_upgrade, 272
- upgrades
 - MySQL Cluster, 2022, 2255
- upgrades and downgrades (MySQL Cluster)
 - compatibility between versions, 2022
- upgrading, 175, 175
 - different architecture, 191
 - to ¤t-series;, 177, 183
- upgrading lock
 - thread state, 903
- upgrading MySQL, 270
- UPPER(), 1110
- uptime, 295
- URLs for downloading MySQL, 52
- usage option
 - ndb_config, 2197
- usage option (MySQL Cluster programs), 2241
- USE, 1522
- use command
 - mysql, 287
- USE INDEX, 1373
- USE KEY, 1373
- use-frm option
 - mysqlcheck, 306
- use-mysqld_safe option
 - mysql.server, 260
- use-threads option
 - mysqlimport, 329
- useConfigs connection properties, 2929
- useHexFormat option
 - ndb_select_all, 2231
- user accounts
 - creating, 1443
 - renaming, 1455
 - resource limits, 525, 721, 1452
- USER environment variable, 193, 240
- User lock
 - thread state, 900
- user names
 - and passwords, 716
 - in account names, 703
 - in default accounts, 171
- user option, 240
 - mysql, 282
 - mysql.server, 260
 - mysqlaccess, 369
 - mysqladmin, 300
 - mysqlbinlog, 377
 - mysqlcheck, 306
 - mysqld, 466
 - mysqldump, 322
 - mysqld_multi, 262
 - mysqld_safe, 258
 - mysqlhotcopy, 389
 - mysqlimport, 329
 - MySQLInstallerConsole, 77
 - mysqlshow, 334
 - mysqslap, 341
 - mysql_convert_table_format, 390
 - mysql_install_db, 267
 - mysql_setpermission, 392
 - mysql_upgrade, 272
- user privileges
 - adding, 718
 - deleting, 721, 1444
 - dropping, 721, 1444
- User sleep
 - thread state, 900
- user table
 - sorting, 707
- user variables, 956
 - and replication, 1959
- USER(), 1197
- User-defined functions, 1465, 1466
- user-defined functions
 - adding, 3841, 3842
 - editing with Visual Studio, 2675
- users
 - adding, 165
 - deleting, 721, 1444
 - root, 171
- USER_PRIVILEGES
 - INFORMATION_SCHEMA table, 2496
- using multiple disks to start data, 878
- using MySQL Cluster programs, 2176
- UTC_DATE(), 1150
- UTC_TIME(), 1150
- UTC_TIMESTAMP(), 1151
- UTF-8, 963
- utf16 character set, 994
- utf16_bin collation, 1005
- utf32 character set, 994
- utf8 character set, 995
- utf8mb3 character set, 995
- utf8mb4 character set, 996
- utilities
 - program-development, 235
- utility programs, 234
- UUID(), 1201
- UUID_SHORT(), 1202

V

- valid numbers
 - examples, 940

VALUES(), 1202
 VARBINARY data type, 1043, 1060
 VARCHAR
 size, 1071
 VARCHAR data type, 1042, 1058
 VARCHARACTER data type, 1042
 variables
 and replication, 1959
 environment, 236
 mysqld, 868
 server, 1507
 status, 585, 1503
 system, 467, 573, 1507
 user, 956
 VARIANCE(), 1207
 VAR_POP(), 1207
 VAR_SAMP(), 1207
 verbose option
 myisamchk, 347
 myisampack, 362
 myisam_ftdump, 343
 mysql, 282
 mysqladmin, 300
 mysqlbinlog, 377
 mysqlcheck, 306
 mysqld, 467
 mysqldump, 322
 mysqldumpslow, 386
 mysqld_multi, 262
 mysqlimport, 330
 mysqlshow, 334
 mysqlslap, 341
 mysql_convert_table_format, 390
 mysql_install_db, 267
 mysql_plugin, 269
 mysql_upgrade, 272
 mysql_waitpid, 393
 my_print_defaults, 396
 perror, 397
 verbose option (ndb_index_stat), 2214
 verbose option (ndb_restore), 2225
 version
 choosing, 49
 latest, 52
 VERSION file
 CMake, 159
 version option
 comp_err, 265
 myisamchk, 347
 myisampack, 362
 mysql, 282
 mysqlaccess, 369
 mysqladmin, 300
 mysqlbinlog, 378
 mysqlcheck, 307
 mysqld, 467
 mysqldump, 322
 mysqld_multi, 262
 mysqlimport, 330
 mysqlshow, 334
 mysqlslap, 341
 mysql_config, 395
 mysql_convert_table_format, 390
 mysql_plugin, 269
 mysql_waitpid, 393
 my_print_defaults, 396
 ndb_config, 2197
 perror, 397
 resolveip, 399
 resolve_stack_dump, 396
 version option (MySQL Cluster programs), 2243
 version system variable, 572
 VERSION(), 1198
 version_comment system variable, 572
 version_compile_machine system variable, 572
 version_compile_os system variable, 572
 vertical option
 mysql, 282
 mysqladmin, 300
 victim, 4191
 Vietnamese, 3957
 view
 restrictions, 4079
 views, 1317, 2441, 2456
 algorithms, 2457
 and replication, 1958
 metadata, 2461
 updatable, 1317, 2459
 VIEWS
 INFORMATION_SCHEMA table, 2496
 Views
 limitations, 4080
 privileges, 4080
 problems, 4080
 Vision, 2645
 Visual Objects, 2642
 Visual Studio, 2662

W

wait, 4191
 WAIT COMPLETED (START BACKUP command), 2251
 wait option
 myisamchk, 347
 myisampack, 362
 mysql, 282
 mysqladmin, 300
 WAIT STARTED (START BACKUP command), 2251

wait-nodes option
 ndb_waiter, 2239
 Waiting for all running commits to finish
 thread state, 900
 Waiting for commit lock
 thread state, 900
 waiting for delay_list
 thread state, 902
 Waiting for event from ndbcluster
 thread state, 906
 Waiting for event metadata lock
 thread state, 901
 Waiting for event read lock
 thread state, 901
 Waiting for first event from ndbcluster
 thread state, 906
 Waiting for global metadata lock
 thread state, 901
 Waiting for global read lock
 thread state, 901, 901
 waiting for handler insert
 thread state, 902
 waiting for handler lock
 thread state, 902
 waiting for handler open
 thread state, 902
 Waiting for INSERT
 thread state, 903
 Waiting for master to send event
 thread state, 904
 Waiting for master update
 thread state, 904
 Waiting for ndbcluster binlog update to reach current
 position
 thread state, 906
 Waiting for ndbcluster to start
 thread state, 906
 Waiting for next activation
 thread state, 907
 Waiting for query cache lock
 thread state, 903
 Waiting for release of readlock
 thread state, 901
 Waiting for scheduler to stop
 thread state, 907
 Waiting for schema epoch
 thread state, 906
 Waiting for schema metadata lock
 thread state, 901
 Waiting for slave mutex on exit
 thread state, 905, 905
 Waiting for stored function metadata lock
 thread state, 901
 Waiting for stored procedure metadata lock
 thread state, 901
 Waiting for table
 thread state, 901
 Waiting for table level lock
 thread state, 901
 Waiting for table metadata lock
 thread state, 901
 Waiting for tables
 thread state, 901
 Waiting for the next event in relay log
 thread state, 905
 Waiting for the slave SQL thread to free enough relay log
 space
 thread state, 905
 Waiting for trigger metadata lock
 thread state, 901
 Waiting on cond
 thread state, 901
 Waiting on empty queue
 thread state, 907
 Waiting to finalize termination
 thread state, 904
 Waiting to get readlock
 thread state, 901
 Waiting to reconnect after a failed binlog dump request
 thread state, 904
 Waiting to reconnect after a failed master event read
 thread state, 905
 wait_timeout system variable, 572
 warm backup, 4192
 warm up, 4192
 warnings command
 mysql, 287
 warning_count session variable, 573
 Website Configuration Tool, 2688
 WEEK(), 1151
 WEEKDAY(), 1152
 WEEKOFYEAR(), 1152
 Well-Known Binary format, 1219
 Well-Known Text format, 1218
 WHERE, 799
 with SHOW, 2469, 2516
 where option
 mysqldump, 322
 WHILE, 1427
 labels, 1420
 widths
 display, 1035
 Wildcard character (%), 938
 Wildcard character (_), 938
 wildcards
 and LIKE, 815
 in account names, 704
 in mysql.columns_priv table, 709

- in mysql.db table, 708
- in mysql.host table, 708
- in mysql.procs_priv table, 709
- in mysql.tables_priv table, 709
- Windows, 2567, 2651, 4192
 - compiling clients on, 3183
 - MySQL limitations, 4088
 - path name separators, 246
 - pluggable authentication, 736
 - upgrading, 114
- Windows Failover Clustering, 1727
- windows option
 - mysql_install_db, 267
- Windows Server Failover Clustering, 1727
- Within(), 1234
- WITH_BUNDLED_LIBEVENT option
 - CMake, 157
- WITH_BUNDLED_MEMCACHED option
 - CMake, 157
- WITH_CLASSPATH option
 - CMake, 157
- WITH_DEBUG option
 - CMake, 155
- WITH_EMBEDDED_SERVER option
 - CMake, 155
- WITH_ERROR_INSERT option
 - CMake, 157
- WITH_EXTRA_CHARSETS option
 - CMake, 155
- WITH_LIBWRAP option
 - CMake, 155
- WITH_NDBCLUSTER option
 - CMake, 157
- WITH_NDBCLUSTER_STORAGE_ENGINE option
 - CMake, 157
- WITH_NDBMTD option
 - CMake, 157
- WITH_NDB_BINLOG option
 - CMake, 157
- WITH_NDB_DEBUG option
 - CMake, 157
- WITH_NDB_JAVA option
 - CMake, 157
- WITH_NDB_PORT option
 - CMake, 158
- WITH_NDB_TEST option
 - CMake, 158
- WITH_READLINE option
 - CMake, 155
- WITH_SSL option
 - CMake, 156
- WITH_UNIXODBC option
 - CMake, 153
- WITH_ZLIB option

- CMake, 156
- WKB format, 1219
- WKT format, 1218
- workload, 4192
- wrappers
 - Eiffel, 3791
- write access
 - tmp, 166
- write combining, 4192
- write-binlog option
 - mysqlcheck, 307
 - mysql_upgrade, 272
- write_buffer_size myisamchk variable, 347
- Writing to net
 - thread state, 902

X

- X(), 1227
- X509/Certificate, 746
- XA, 4193
- XA BEGIN, 1404
- XA COMMIT, 1404
- XA PREPARE, 1404
- XA RECOVER, 1404
- XA ROLLBACK, 1404
- XA START, 1404
- XA transactions, 1402
 - restrictions, 4081
 - transaction identifiers, 1404
- xid
 - XA transaction identifier, 1404
- xml option
 - mysql, 282
 - mysqldump, 322
 - ndb_config, 2201
- XOR
 - bitwise, 1182
 - logical, 1095

Y

- Y(), 1227
- yaSSL, 746
- YEAR data type, 1040, 1051
- YEAR(), 1152
- YEARWEEK(), 1152
- Yen sign (Japanese), 3957
- young, 4193

Z

- ZEROFILL, 1036, 1044, 3177
- ZFS, 1734

C Function Index

my_init()

Section 22.8.8.1, “my_init()”
Section 22.8.8.3, “mysql_thread_init()”
Section 22.8.2, “C API Function Overview”

mysql_affected_rows()

Section 13.2.1, “CALL Syntax”
Section 13.2.5, “INSERT Syntax”
Section 22.8.3.1, “mysql_affected_rows()”
Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.48, “mysql_num_rows()”
Section 22.8.7.1, “mysql_stmt_affected_rows()”
Section 22.8.3.71, “mysql_use_result()”
Section 13.2.8, “REPLACE Syntax”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”
Section 12.14, “Information Functions”
Section 22.8.11.2, “What Results You Can Get from a Query”

mysql_autocommit()

Section 22.8.3.2, “mysql_autocommit()”
Section 22.8.2, “C API Function Overview”

mysql_change_user()

Section 22.8.3.3, “mysql_change_user()”
Section 22.8.2, “C API Function Overview”

mysql_character_set_name()

Section 22.8.3.4, “mysql_character_set_name()”
Section 22.8.2, “C API Function Overview”

mysql_client_find_plugin()

Section 22.8.10.1, “mysql_client_find_plugin()”
Section 22.8.2, “C API Function Overview”

mysql_client_register_plugin()

Section 22.8.10.2,
“mysql_client_register_plugin()”
Section 22.8.2, “C API Function Overview”

mysql_close()

Section 22.8.3.5, “mysql_close()”
Section 22.8.3.6, “mysql_commit()”
Section 22.8.3.7, “mysql_connect()”
Section 22.8.3.36, “mysql_init()”
Section 22.8.3.57, “mysql_rollback()”
Section 22.8.2, “C API Function Overview”
Section C.5.2.11, “Communication Errors and Aborted Connections”

mysql_commit()

Section 22.8.3.6, “mysql_commit()”
Section 22.8.2, “C API Function Overview”

mysql_connect()

Section 22.8.8.1, “my_init()”
Section 22.8.3.5, “mysql_close()”
Section 22.8.3.7, “mysql_connect()”
Section 22.8.3.49, “mysql_options()”
Section 22.8.8.3, “mysql_thread_init()”
Section 22.8.2, “C API Function Overview”
Section 22.8.17.2, “How to Write a Threaded Client”

mysql_create_db()

Section 22.8.3.8, “mysql_create_db()”
Section 22.8.2, “C API Function Overview”

mysql_data_seek()

Section 22.8.3.9, “mysql_data_seek()”
Section 22.8.3.58, “mysql_row_seek()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.2, “C API Function Overview”

mysql_debug()

Section 22.8.3.10, “mysql_debug()”
Section 22.8.2, “C API Function Overview”

mysql_drop_db()

Section 22.8.3.11, “mysql_drop_db()”
Section 22.8.2, “C API Function Overview”

mysql_dump_debug_info()

Section 22.8.3.12, “mysql_dump_debug_info()”
Section 22.8.2, “C API Function Overview”

mysql_eof()

Section 22.8.3.13, “mysql_eof()”
Section 22.8.2, “C API Function Overview”

mysql_errno()

Section 22.8.10.1, “mysql_client_find_plugin()”
Section 22.8.10.2,
“mysql_client_register_plugin()”
Section 22.8.3.7, “mysql_connect()”
Section 22.8.3.13, “mysql_eof()”
Section 22.8.3.14, “mysql_errno()”
Section 22.8.3.22, “mysql_field_count()”
Section 22.8.10.3, “mysql_load_plugin()”
Section 22.8.3.47, “mysql_num_fields()”
Section 22.8.3.66, “mysql_sqlstate()”
Section 22.8.3.69, “mysql_store_result()”

Section 22.8.3.71, “mysql_use_result()”
Section 22.8.3, “C API Function Descriptions”
Section 22.8.2, “C API Function Overview”
Section 13.6.7.4.1, “Signal Condition Information Items”
Section C.2, “Types of Error Values”
Section 22.8.11.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 23.2.4.8, “Writing Audit Plugins”

mysql_error()

Section 22.8.10.1, “mysql_client_find_plugin()”
Section 22.8.10.2, “mysql_client_register_plugin()”
Section 22.8.3.7, “mysql_connect()”
Section 22.8.3.13, “mysql_eof()”
Section 22.8.3.15, “mysql_error()”
Section 22.8.10.3, “mysql_load_plugin()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.3, “C API Function Descriptions”
Section 22.8.2, “C API Function Overview”
Section 13.6.7.4.1, “Signal Condition Information Items”
Section C.2, “Types of Error Values”
Section 22.8.11.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 23.2.4.8, “Writing Audit Plugins”

mysql_escape_string()

Section 22.8.3.16, “mysql_escape_string()”
Section 22.8.2, “C API Function Overview”

mysql_fetch_field()

Section 22.8.3.17, “mysql_fetch_field()”
Section 22.8.3.23, “mysql_field_seek()”
Section 22.8.3.24, “mysql_field_tell()”
Section 22.8.7.23, “mysql_stmt_result_metadata()”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”

mysql_fetch_field_direct()

Section 22.8.3.18, “mysql_fetch_field_direct()”
Section 22.8.7.23, “mysql_stmt_result_metadata()”
Section 22.8.2, “C API Function Overview”

mysql_fetch_fields()

Section 22.8.3.19, “mysql_fetch_fields()”
Section 22.8.7.23, “mysql_stmt_result_metadata()”

Section 22.8.2, “C API Function Overview”

mysql_fetch_lengths()

Section 22.8.3.20, “mysql_fetch_lengths()”
Section 22.8.3.21, “mysql_fetch_row()”
Section 22.8.2, “C API Function Overview”

mysql_fetch_row()

Section 14.11.1, “FEDERATED Storage Engine Overview”
Section 22.8.3.13, “mysql_eof()”
Section 22.8.3.14, “mysql_errno()”
Section 22.8.3.20, “mysql_fetch_lengths()”
Section 22.8.3.21, “mysql_fetch_row()”
Section 22.8.3.59, “mysql_row_tell()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”
Section 22.8.11.2, “What Results You Can Get from a Query”

mysql_field_count()

Section 22.8.3.22, “mysql_field_count()”
Section 22.8.3.47, “mysql_num_fields()”
Section 22.8.3.51, “mysql_query()”
Section 22.8.3.54, “mysql_real_query()”
Section 22.8.7.23, “mysql_stmt_result_metadata()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.2, “C API Function Overview”
Section 22.8.11.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

mysql_field_seek()

Section 22.8.3.17, “mysql_fetch_field()”
Section 22.8.3.23, “mysql_field_seek()”
Section 22.8.3.24, “mysql_field_tell()”
Section 22.8.7.23, “mysql_stmt_result_metadata()”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”

mysql_field_tell()

Section 22.8.3.24, “mysql_field_tell()”
Section 22.8.7.23, “mysql_stmt_result_metadata()”
Section 22.8.2, “C API Function Overview”

mysql_free_result()

Section C.5.2.14, “Commands out of sync”
Section 22.8.3.25, “mysql_free_result()”

Section 22.8.3.41, “[mysql_list_dbs\(\)](#)”
Section 22.8.3.42, “[mysql_list_fields\(\)](#)”
Section 22.8.3.43, “[mysql_list_processes\(\)](#)”
Section 22.8.3.44, “[mysql_list_tables\(\)](#)”
Section 22.8.3.46, “[mysql_next_result\(\)](#)”
Section 22.8.7.23,
“[mysql_stmt_result_metadata\(\)](#)”
Section 22.8.3.69, “[mysql_store_result\(\)](#)”
Section 22.8.3.71, “[mysql_use_result\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”
Section 22.8.6, “[C API Prepared Statement Function Overview](#)”

mysql_get_character_set_info()

Section 22.8.3.26,
“[mysql_get_character_set_info\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”
Section 10.4.2, “[Choosing a Collation ID](#)”

mysql_get_client_info()

Section 22.8.3.7, “[mysql_connect\(\)](#)”
Section 22.8.3.27, “[mysql_get_client_info\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”

mysql_get_client_version()

Section 22.8.3.28, “[mysql_get_client_version\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”

mysql_get_host_info()

Section 22.8.3.29, “[mysql_get_host_info\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”

mysql_get_proto_info()

Section 22.8.3.30, “[mysql_get_proto_info\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”

mysql_get_server_info()

Section 22.8.3.31, “[mysql_get_server_info\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”

mysql_get_server_version()

Section 22.8.3.32, “[mysql_get_server_version\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”

mysql_get_ssl_cipher()

Section 22.8.3.33, “[mysql_get_ssl_cipher\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”
Section 6.3.8.3, “[Using SSL Connections](#)”

mysql_hex_string()

Section 22.8.3.34, “[mysql_hex_string\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”

mysql_info()

Section 13.1.7, “[ALTER TABLE Syntax](#)”
Section 13.2.5.2, “[INSERT DELAYED Syntax](#)”
Section 13.2.5, “[INSERT Syntax](#)”
Section 13.2.6, “[LOAD DATA INFILE Syntax](#)”
Section 22.8.3.35, “[mysql_info\(\)](#)”
Section 22.8.3.49, “[mysql_options\(\)](#)”
Section 1.8.6.1, “[PRIMARY KEY and UNIQUE Index Constraints](#)”
Section 13.2.11, “[UPDATE Syntax](#)”
Section 22.8.2, “[C API Function Overview](#)”
Section 22.8.11.2, “[What Results You Can Get from a Query](#)”

mysql_init()

Section 22.8.8.1, “[my_init\(\)](#)”
Section 22.8.3.5, “[mysql_close\(\)](#)”
Section 22.8.3.33, “[mysql_get_ssl_cipher\(\)](#)”
Section 22.8.3.36, “[mysql_init\(\)](#)”
Section 22.8.3.40, “[mysql_library_init\(\)](#)”
Section 22.8.3.49, “[mysql_options\(\)](#)”
Section 22.8.3.52, “[mysql_real_connect\(\)](#)”
Section 22.8.3.67, “[mysql_ssl_set\(\)](#)”
Section 22.8.8.3, “[mysql_thread_init\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”
Section 22.8.17.2, “[How to Write a Threaded Client](#)”

mysql_insert_id()

Section 13.1.17, “[CREATE TABLE Syntax](#)”
Section 13.2.5, “[INSERT Syntax](#)”
Section 22.8.3.37, “[mysql_insert_id\(\)](#)”
Section 22.8.1, “[C API Data Structures](#)”
Section 22.8.2, “[C API Function Overview](#)”
Section 22.8.11.3, “[How to Get the Unique ID for the Last Inserted Row](#)”
Section 12.14, “[Information Functions](#)”
Section 5.1.4, “[Server System Variables](#)”
Section 1.8.5.3, “[Transaction and Atomic Operation Differences](#)”
Section 3.6.9, “[Using AUTO_INCREMENT](#)”
Section 22.8.11.2, “[What Results You Can Get from a Query](#)”

mysql_kill()

Section 22.8.3.38, “[mysql_kill\(\)](#)”
Section 22.8.3.70, “[mysql_thread_id\(\)](#)”
Section 22.8.2, “[C API Function Overview](#)”
Section 22.8.12, “[Controlling Automatic Reconnection Behavior](#)”

mysql_library_end()

Section 22.8.3.39, “[mysql_library_end\(\)](#)”

Section 22.8.3.40, “mysql_library_init()”
Section 22.8.9.2, “mysql_server_end()”
Section 22.8.9, “C API Embedded Server Function Descriptions”
Section 22.8.2, “C API Function Overview”
Section 22.7, “libmysqld, the Embedded MySQL Server Library”

mysql_library_init()

Section 22.8.8.1, “my_init()”
Section 22.8.3.40, “mysql_library_init()”
Section 22.8.10.3, “mysql_load_plugin()”
Section 22.8.9.1, “mysql_server_init()”
Section 22.8.8.3, “mysql_thread_init()”
Section 22.8.9, “C API Embedded Server Function Descriptions”
Section 22.8.2, “C API Function Overview”
Section 22.8.17.2, “How to Write a Threaded Client”
Section 22.7, “libmysqld, the Embedded MySQL Server Library”
Section 22.7.3, “Options with the Embedded Server”

mysql_list_dbs()

Section 22.8.3.25, “mysql_free_result()”
Section 22.8.3.41, “mysql_list_dbs()”
Section 22.8.2, “C API Function Overview”

mysql_list_fields()

Section 22.8.3.42, “mysql_list_fields()”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”

mysql_list_processes()

Section 22.8.3.43, “mysql_list_processes()”
Section 22.8.2, “C API Function Overview”

mysql_list_tables()

Section 22.8.3.44, “mysql_list_tables()”
Section 22.8.2, “C API Function Overview”

mysql_load_plugin()

Section 22.8.10.3, “mysql_load_plugin()”
Section 22.8.10.4, “mysql_load_plugin_v()”
Section 22.8.2, “C API Function Overview”
Section 23.2.4.2.3, “Client Plugin Descriptors”

mysql_load_plugin_v()

Section 22.8.10.4, “mysql_load_plugin_v()”
Section 22.8.2, “C API Function Overview”

mysql_more_results()

Section 22.8.3.45, “mysql_more_results()”

Section 22.8.3.46, “mysql_next_result()”
Section 22.8.7.17, “mysql_stmt_next_result()”
Section 22.8.2, “C API Function Overview”
Section 22.8.13, “C API Support for Multiple Statement Execution”

mysql_next_result()

Section 13.2.1, “CALL Syntax”
Section 22.8.3.45, “mysql_more_results()”
Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.52, “mysql_real_connect()”
Section 22.8.3.64, “mysql_set_server_option()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.2, “C API Function Overview”
Section 22.8.13, “C API Support for Multiple Statement Execution”

mysql_num_fields()

Section 22.8.3.18, “mysql_fetch_field_direct()”
Section 22.8.3.21, “mysql_fetch_row()”
Section 22.8.3.47, “mysql_num_fields()”
Section 22.8.7.23, “mysql_stmt_result_metadata()”
Section 22.8.2, “C API Function Overview”

mysql_num_rows()

Section 22.8.3.1, “mysql_affected_rows()”
Section 22.8.3.9, “mysql_data_seek()”
Section 22.8.3.48, “mysql_num_rows()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”
Section 22.8.11.2, “What Results You Can Get from a Query”

mysql_options()

Section C.5.2.9, “MySQL server has gone away”
Section 22.8.3.49, “mysql_options()”
Section 22.8.3.50, “mysql_ping()”
Section 22.8.3.52, “mysql_real_connect()”
Section 22.8.7.11, “mysql_stmt_fetch()”
Section 22.8.10, “C API Client Plugin Functions”
Section 22.8.2, “C API Function Overview”
Section 22.8.5, “C API Prepared Statement Data Structures”
Section 23.2.4.2.3, “Client Plugin Descriptors”
Section 10.1.4, “Connection Character Sets and Collations”
Section 22.8.12, “Controlling Automatic Reconnection Behavior”
Section 23.2.2, “Plugin API Components”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

Section 5.3.4, “Using Client Programs in a Multiple-Server Environment”
Section 23.2.4.9.3, “Using the Authentication Plugins”

mysql_ping()

Section C.5.2.9, “MySQL server has gone away”
Section 22.8.3.50, “mysql_ping()”
Section 22.8.3.70, “mysql_thread_id()”
Section 22.8.2, “C API Function Overview”
Section 22.8.12, “Controlling Automatic Reconnection Behavior”

mysql_plugin_options()

Section 22.8.10.5, “mysql_plugin_options()”
Section 22.8.2, “C API Function Overview”

mysql_query()

Section 13.2.1, “CALL Syntax”
Section 22.8.3.1, “mysql_affected_rows()”
Section 22.8.3.8, “mysql_create_db()”
Section 22.8.3.11, “mysql_drop_db()”
Section 22.8.3.17, “mysql_fetch_field()”
Section 22.8.3.38, “mysql_kill()”
Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.51, “mysql_query()”
Section 22.8.3.52, “mysql_real_connect()”
Section 22.8.3.54, “mysql_real_query()”
Section 22.8.3.56, “mysql_reload()”
Section 22.8.3.63, “mysql_set_local_infile_handler()”
Section 22.8.3.64, “mysql_set_server_option()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.2, “C API Function Overview”
Section 22.8.13, “C API Support for Multiple Statement Execution”
Section 22.8.11.3, “How to Get the Unique ID for the Last Inserted Row”
Section 22.8.17.2, “How to Write a Threaded Client”
Section 22.8.11.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

mysql_real_connect()

Section 13.2.1, “CALL Syntax”
Section 22.8.3.1, “mysql_affected_rows()”
Section 22.8.3.3, “mysql_change_user()”
Section 22.8.3.7, “mysql_connect()”
Section 22.8.3.36, “mysql_init()”
Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.49, “mysql_options()”
Section 22.8.3.52, “mysql_real_connect()”
Section 22.8.3.64, “mysql_set_server_option()”

Section 22.8.3.66, “mysql_sqlstate()”
Section 22.8.3.67, “mysql_ssl_set()”
Section 22.8.2, “C API Function Overview”
Section 22.8.13, “C API Support for Multiple Statement Execution”
Chapter 12, *Functions and Operators*
Section 12.14, “Information Functions”
Section 5.1.4, “Server System Variables”
Section 13.5, “SQL Syntax for Prepared Statements”
Section 19.2.1, “Stored Routine Syntax”
Section 5.3.4, “Using Client Programs in a Multiple-Server Environment”
Section 6.3.8.3, “Using SSL Connections”

mysql_real_escape_string()

Section 22.8.3.16, “mysql_escape_string()”
Section 22.8.3.53, “mysql_real_escape_string()”
Section 22.8.3.61, “mysql_set_character_set()”
Section 22.8.2, “C API Function Overview”
Section 6.1.7, “Client Programming Security Guidelines”
Section 12.17.4.4, “Populating Spatial Columns”
Section 9.1.1, “String Literals”

mysql_real_query()

Section 13.2.1, “CALL Syntax”
Section 14.11.1, “FEDERATED Storage Engine Overview”
Section 22.8.3.1, “mysql_affected_rows()”
Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.51, “mysql_query()”
Section 22.8.3.52, “mysql_real_connect()”
Section 22.8.3.54, “mysql_real_query()”
Section 22.8.3.64, “mysql_set_server_option()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.2, “C API Function Overview”
Section 22.8.13, “C API Support for Multiple Statement Execution”

mysql_refresh()

Section 22.8.3.55, “mysql_refresh()”
Section 22.8.2, “C API Function Overview”

mysql_reload()

Section 22.8.3.56, “mysql_reload()”
Section 22.8.2, “C API Function Overview”

mysql_rollback()

Section 22.8.3.57, “mysql_rollback()”
Section 22.8.2, “C API Function Overview”

mysql_row_seek()

Section 22.8.3.58, “mysql_row_seek()”
Section 22.8.3.59, “mysql_row_tell()”

Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.2, “C API Function Overview”

mysql_row_tell()

Section 22.8.3.58, “mysql_row_seek()”
Section 22.8.3.59, “mysql_row_tell()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.2, “C API Function Overview”

mysql_select_db()

Section 22.8.3.60, “mysql_select_db()”
Section 22.8.2, “C API Function Overview”

mysql_server_end()

Section 22.8.9.2, “mysql_server_end()”
Section 22.8.2, “C API Function Overview”

mysql_server_init()

Section 22.8.8.1, “my_init()”
Section 22.8.9.1, “mysql_server_init()”
Section 22.8.8.3, “mysql_thread_init()”
Section 22.8.2, “C API Function Overview”

mysql_set_character_set()

Section 22.8.3.26,
“mysql_get_character_set_info()”
Section 22.8.3.53, “mysql_real_escape_string()”
Section 22.8.3.61, “mysql_set_character_set()”
Section 22.8.2, “C API Function Overview”

mysql_set_local_infile_default()

Section 22.8.3.62,
“mysql_set_local_infile_default()”
Section 22.8.2, “C API Function Overview”

mysql_set_local_infile_handler()

Section 22.8.3.62,
“mysql_set_local_infile_default()”
Section 22.8.3.63,
“mysql_set_local_infile_handler()”
Section 22.8.2, “C API Function Overview”

mysql_set_server_option()

Section 22.8.3.64, “mysql_set_server_option()”
Section 22.8.2, “C API Function Overview”
Section 22.8.13, “C API Support for Multiple Statement Execution”

mysql_shutdown()

Section 22.8.3.65, “mysql_shutdown()”

Section 22.8.2, “C API Function Overview”

mysql_sqlstate()

Section 22.8.3.14, “mysql_errno()”
Section 22.8.3.66, “mysql_sqlstate()”
Section 22.8.2, “C API Function Overview”
Section 13.6.7.4.1, “Signal Condition Information Items”
Section C.2, “Types of Error Values”

mysql_ssl_set()

Section 22.8.3.52, “mysql_real_connect()”
Section 22.8.3.67, “mysql_ssl_set()”
Section 22.8.2, “C API Function Overview”
Section 6.3.8.3, “Using SSL Connections”

mysql_stat()

Section 22.8.3.68, “mysql_stat()”
Section 22.8.2, “C API Function Overview”

mysql_stmt_affected_rows()

Section 22.8.7.1, “mysql_stmt_affected_rows()”
Section 22.8.7.10, “mysql_stmt_execute()”
Section 22.8.7.17, “mysql_stmt_next_result()”
Section 22.8.7.18, “mysql_stmt_num_rows()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_attr_get()

Section 22.8.7.2, “mysql_stmt_attr_get()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_attr_set()

Section 22.8.7.3, “mysql_stmt_attr_set()”
Section 22.8.7.10, “mysql_stmt_execute()”
Section 22.8.7.11, “mysql_stmt_fetch()”
Section 22.8.7.28, “mysql_stmt_store_result()”
Section 22.8.1, “C API Data Structures”
Section 22.8.6, “C API Prepared Statement Function Overview”
Section 22.8.5.2, “C API Prepared Statement Type Conversions”
Section E.3, “Restrictions on Server-Side Cursors”

mysql_stmt_bind_param()

Section 22.8.7.4, “mysql_stmt_bind_param()”
Section 22.8.7.10, “mysql_stmt_execute()”
Section 22.8.7.21, “mysql_stmt_prepare()”
Section 22.8.7.26, “mysql_stmt_send_long_data()”
Section 22.8.5, “C API Prepared Statement Data Structures”
Section 22.8.6, “C API Prepared Statement Function Overview”

Section 22.8.15, “C API Prepared Statement Handling of Date and Time Values”

mysql_stmt_bind_result()

Section 22.8.7.5, “mysql_stmt_bind_result()”

Section 22.8.7.11, “mysql_stmt_fetch()”

Section 22.8.7.12, “mysql_stmt_fetch_column()”

Section 22.8.7.17, “mysql_stmt_next_result()”

Section 22.8.7.28, “mysql_stmt_store_result()”

Section 22.8.5, “C API Prepared Statement Data Structures”

Section 22.8.6, “C API Prepared Statement Function Overview”

Section 22.8.15, “C API Prepared Statement Handling of Date and Time Values”

mysql_stmt_close()

Section 22.8.7.6, “mysql_stmt_close()”

Section 22.8.7.15, “mysql_stmt_init()”

Section 22.8.5, “C API Prepared Statement Data Structures”

Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_data_seek()

Section 22.8.7.7, “mysql_stmt_data_seek()”

Section 22.8.7.24, “mysql_stmt_row_seek()”

Section 22.8.7.28, “mysql_stmt_store_result()”

Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_errno()

Section 22.8.7.8, “mysql_stmt_errno()”

Section 22.8.7.11, “mysql_stmt_fetch()”

Section 22.8.6, “C API Prepared Statement Function Overview”

Section C.2, “Types of Error Values”

mysql_stmt_error()

Section 22.8.7.9, “mysql_stmt_error()”

Section 22.8.7.11, “mysql_stmt_fetch()”

Section 22.8.7.21, “mysql_stmt_prepare()”

Section 22.8.6, “C API Prepared Statement Function Overview”

Section C.2, “Types of Error Values”

mysql_stmt_execute()

Section 22.8.7.1, “mysql_stmt_affected_rows()”

Section 22.8.7.3, “mysql_stmt_attr_set()”

Section 22.8.7.10, “mysql_stmt_execute()”

Section 22.8.7.11, “mysql_stmt_fetch()”

Section 22.8.7.17, “mysql_stmt_next_result()”

Section 22.8.7.26, “mysql_stmt_send_long_data()”

Section 22.8.7.28, “mysql_stmt_store_result()”

Section 22.8.5, “C API Prepared Statement Data Structures”

Section 22.8.6, “C API Prepared Statement Function Overview”

Section 22.8.15, “C API Prepared Statement Handling of Date and Time Values”

Section 22.8.5.2, “C API Prepared Statement Type Conversions”

Section 8.9.3.1, “How the Query Cache Operates”

mysql_stmt_fetch()

Section 22.8.7.5, “mysql_stmt_bind_result()”

Section 22.8.7.10, “mysql_stmt_execute()”

Section 22.8.7.11, “mysql_stmt_fetch()”

Section 22.8.7.23,

“mysql_stmt_result_metadata()”

Section 22.8.7.25, “mysql_stmt_row_tell()”

Section 22.8.7.28, “mysql_stmt_store_result()”

Section 22.8.5, “C API Prepared Statement Data Structures”

Section 22.8.6, “C API Prepared Statement Function Overview”

Section 22.8.5.2, “C API Prepared Statement Type Conversions”

mysql_stmt_fetch_column()

Section 22.8.7.11, “mysql_stmt_fetch()”

Section 22.8.7.12, “mysql_stmt_fetch_column()”

Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_field_count()

Section 22.8.7.13, “mysql_stmt_field_count()”

Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_free_result()

Section 22.8.7.3, “mysql_stmt_attr_set()”

Section 22.8.7.14, “mysql_stmt_free_result()”

Section 22.8.7.17, “mysql_stmt_next_result()”

Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_init()

Section 22.8.7.10, “mysql_stmt_execute()”

Section 22.8.7.15, “mysql_stmt_init()”

Section 22.8.7.21, “mysql_stmt_prepare()”

Section 22.8.5, “C API Prepared Statement Data Structures”

Section 22.8.7, “C API Prepared Statement Function Descriptions”

Section 22.8.6, “C API Prepared Statement Function Overview”
Section 22.8.4, “C API Prepared Statements”

mysql_stmt_insert_id()

Section 22.8.7.16, “mysql_stmt_insert_id()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_next_result()

Section 13.2.1, “CALL Syntax”
Section 22.8.7.17, “mysql_stmt_next_result()”
Section 22.8.6, “C API Prepared Statement Function Overview”
Section 22.8.16, “C API Support for Prepared CALL Statements”

mysql_stmt_num_rows()

Section 22.8.7.7, “mysql_stmt_data_seek()”
Section 22.8.7.18, “mysql_stmt_num_rows()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_param_count()

Section 22.8.7.10, “mysql_stmt_execute()”
Section 22.8.7.19, “mysql_stmt_param_count()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_param_metadata()

Section 22.8.7.20, “mysql_stmt_param_metadata()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_prepare()

Section 22.8.7.4, “mysql_stmt_bind_param()”
Section 22.8.7.10, “mysql_stmt_execute()”
Section 22.8.7.13, “mysql_stmt_field_count()”
Section 22.8.7.21, “mysql_stmt_prepare()”
Section 22.8.7.22, “mysql_stmt_reset()”
Section 22.8.7.23,
“mysql_stmt_result_metadata()”
Section 13.5.4, “Automatic Prepared Statement Reparation”
Section 22.8.5, “C API Prepared Statement Data Structures”
Section 22.8.6, “C API Prepared Statement Function Overview”
Section 22.8.15, “C API Prepared Statement Handling of Date and Time Values”
Section 8.9.3.1, “How the Query Cache Operates”
Section 13.5, “SQL Syntax for Prepared Statements”

mysql_stmt_reset()

Section 22.8.7.3, “mysql_stmt_attr_set()”
Section 22.8.7.22, “mysql_stmt_reset()”
Section 22.8.7.26, “mysql_stmt_send_long_data()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_result_metadata()

Section 22.8.7.11, “mysql_stmt_fetch()”
Section 22.8.7.23,
“mysql_stmt_result_metadata()”
Section 22.8.7.28, “mysql_stmt_store_result()”
Section 22.8.6, “C API Prepared Statement Function Overview”
Section 22.8.5.2, “C API Prepared Statement Type Conversions”

mysql_stmt_row_seek()

Section 22.8.7.24, “mysql_stmt_row_seek()”
Section 22.8.7.25, “mysql_stmt_row_tell()”
Section 22.8.7.28, “mysql_stmt_store_result()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_row_tell()

Section 22.8.7.24, “mysql_stmt_row_seek()”
Section 22.8.7.25, “mysql_stmt_row_tell()”
Section 22.8.7.28, “mysql_stmt_store_result()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_send_long_data()

Section 22.8.7.22, “mysql_stmt_reset()”
Section 22.8.7.26, “mysql_stmt_send_long_data()”
Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_stmt_sqlstate()

Section 22.8.7.27, “mysql_stmt_sqlstate()”
Section 22.8.6, “C API Prepared Statement Function Overview”
Section C.2, “Types of Error Values”

mysql_stmt_store_result()

Section 22.8.7.3, “mysql_stmt_attr_set()”
Section 22.8.7.7, “mysql_stmt_data_seek()”
Section 22.8.7.11, “mysql_stmt_fetch()”
Section 22.8.7.18, “mysql_stmt_num_rows()”
Section 22.8.7.24, “mysql_stmt_row_seek()”
Section 22.8.7.25, “mysql_stmt_row_tell()”
Section 22.8.7.28, “mysql_stmt_store_result()”
Section 22.8.1, “C API Data Structures”

Section 22.8.6, “C API Prepared Statement Function Overview”

mysql_store_result()

Section C.5.2.14, “Commands out of sync”
Section 14.11.1, “FEDERATED Storage Engine Overview”
Section 22.8.3.1, “mysql_affected_rows()”
Section 22.8.3.9, “mysql_data_seek()”
Section 22.8.3.13, “mysql_eof()”
Section 22.8.3.17, “mysql_fetch_field()”
Section 22.8.3.21, “mysql_fetch_row()”
Section 22.8.3.22, “mysql_field_count()”
Section 22.8.3.25, “mysql_free_result()”
Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.47, “mysql_num_fields()”
Section 22.8.3.48, “mysql_num_rows()”
Section 22.8.3.58, “mysql_row_seek()”
Section 22.8.3.59, “mysql_row_tell()”
Section 22.8.7.10, “mysql_stmt_execute()”
Section 22.8.7.23,
“mysql_stmt_result_metadata()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”
Section 4.5.1, “mysql — The MySQL Command-Line Tool”
Section 22.8.17.2, “How to Write a Threaded Client”
Section 22.8.11.2, “What Results You Can Get from a Query”
Section 22.8.11.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

mysql_thread_end()

Section 22.8.8.2, “mysql_thread_end()”
Section 22.8.2, “C API Function Overview”
Section 22.8.17.2, “How to Write a Threaded Client”
Section 22.7, “libmysqld, the Embedded MySQL Server Library”

mysql_thread_id()

Section 22.8.3.50, “mysql_ping()”
Section 22.8.3.70, “mysql_thread_id()”
Section 22.8.2, “C API Function Overview”
Section 22.8.12, “Controlling Automatic Reconnection Behavior”

mysql_thread_init()

Section 22.8.8.1, “my_init()”
Section 22.8.8.2, “mysql_thread_end()”
Section 22.8.8.3, “mysql_thread_init()”
Section 22.8.2, “C API Function Overview”

Section 22.8.17.2, “How to Write a Threaded Client”
Section 22.7, “libmysqld, the Embedded MySQL Server Library”

mysql_thread_safe()

Section 22.8.8.4, “mysql_thread_safe()”
Section 22.8.2, “C API Function Overview”

mysql_use_result()

Section C.5.2.14, “Commands out of sync”
Section 22.8.3.9, “mysql_data_seek()”
Section 22.8.3.13, “mysql_eof()”
Section 22.8.3.21, “mysql_fetch_row()”
Section 22.8.3.25, “mysql_free_result()”
Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.47, “mysql_num_fields()”
Section 22.8.3.48, “mysql_num_rows()”
Section 22.8.3.58, “mysql_row_seek()”
Section 22.8.3.59, “mysql_row_tell()”
Section 22.8.7.10, “mysql_stmt_execute()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section C.5.2.8, “Out of memory”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”
Section 4.5.1, “mysql — The MySQL Command-Line Tool”
Section 22.8.17.2, “How to Write a Threaded Client”
Section 22.8.11.2, “What Results You Can Get from a Query”

mysql_warning_count()

Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.72, “mysql_warning_count()”
Section 13.7.5.41, “SHOW WARNINGS Syntax”
Section 22.8.2, “C API Function Overview”
Section C.2, “Types of Error Values”

Command Index

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [G](#) | [H](#) | [I](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [Y](#) | [Z](#)

A

[\[index top \[4287\]\]](#)

Access

[Section 13.2.2, “DELETE Syntax”](#)

addgroup

[Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”](#)

[Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#)

addr2line

[Section 23.4.1.5, “Using a Stack Trace”](#)

adduser

[Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”](#)

[Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”](#)

ALL STATUS

[Section 17.5.8, “MySQL Cluster Single User Mode”](#)

APF

[Section 17.5.11.1, “MySQL Cluster Security and Networking Issues”](#)

apt-get

[Section 2.5.2, “Installing MySQL on Linux using Native Package Manager”](#)

B

[\[index top \[4287\]\]](#)

bash

[Section 6.1.2.1, “End-User Guidelines for Password Security”](#)

[Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”](#)

[Section 4.2.1, “Invoking MySQL Programs”](#)

[Section 16.1.3.3, “Replication Slave Options and Variables”](#)

[Section 4.2.4, “Setting Environment Variables”](#)

[Section 1.2, “Typographical and Syntax Conventions”](#)

bison

[Section 2.10.5, “Dealing with Problems Compiling MySQL”](#)

[Section 2.10.3, “Installing MySQL from a Development Source Tree”](#)

[Section 2.10, “Installing MySQL from Source”](#)

bzip

[Section 2.10.3, “Installing MySQL from a Development Source Tree”](#)

[Section 22.4, “MySQL Connector/C++”](#)

C

[\[index top \[4287\]\]](#)

c++filt

[Section 23.4.1.5, “Using a Stack Trace”](#)

cat

[Section 4.5.1.1, “mysql Options”](#)

chkconfig

[Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”](#)

[Section 2.5.2, “Installing MySQL on Linux using Native Package Manager”](#)

[Section 2.11.1.2, “Starting and Stopping MySQL Automatically”](#)

chroot

[Section 4.6.9, “mysqlhotcopy — A Database Backup Program”](#)

CMake

[Section C.5.2.17, “Can't initialize character set”](#)

[Section 10.3, “Adding a Character Set”](#)

[Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”](#)

[Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”](#)

[Section 23.2.4.3, “Compiling and Installing Plugin Libraries”](#)

[Section 23.3.2.5, “Compiling and Installing User-Defined Functions”](#)

[Section 6.3.8.2, “Configuring MySQL for SSL”](#)

[Section 10.1.5, “Configuring the Character Set and Collation for Applications”](#)

Section 2.10.5, “Dealing with Problems Compiling MySQL”
Section 2.13, “Environment Variables”
Section C.5.4.5, “How to Protect or Change the MySQL Unix Socket File”
Section 2.10.3, “Installing MySQL from a Development Source Tree”
Section 2.10.2, “Installing MySQL from a Standard Source Distribution”
Section 2.10, “Installing MySQL from Source”
Section 22.7, “libmysqld, the Embedded MySQL Server Library”
Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Chapter 17, *MySQL Cluster NDB 7.2*
Section 23.2.5, “MySQL Services for Plugins”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 2.1.1, “Operating Systems Supported by MySQL Community Server”
Section 21.2.1, “Performance Schema Build Configuration”
Section 5.3.3, “Running Multiple MySQL Instances on Unix”
Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”
Section 10.1.3.1, “Server Character Set and Collation”
Section 5.1.4, “Server System Variables”
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 14.9, “The `BLACKHOLE` Storage Engine”
Section 14.12, “The `EXAMPLE` Storage Engine”
Section 14.11, “The `FEDERATED` Storage Engine”
Section 5.4, “Tracing `mysqld` Using DTrace”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”
Section 4.2.3.3, “Using Option Files”
Section 1.4, “What Is New in MySQL 5.5”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

cmake

Section 23.3.2.5, “Compiling and Installing User-Defined Functions”
Section 2.10.2, “Installing MySQL from a Standard Source Distribution”

cmd

Section C.5.4.1.1, “Resetting the Root Password: Windows Systems”

cmd.exe

Section 4.2.1, “Invoking MySQL Programs”
Section 1.2, “Typographical and Syntax Conventions”

command.com

Section 4.2.1, “Invoking MySQL Programs”

Section 1.2, “Typographical and Syntax Conventions”

comp_err

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.1, “Overview of MySQL Programs”

configure

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 2.13, “Environment Variables”
Section 1.7, “How to Report Bugs or Problems”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 1.2, “Typographical and Syntax Conventions”
Section 1.4, “What Is New in MySQL 5.5”

copy

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

coreadm

Section 2.7, “Installing MySQL on Solaris and OpenSolaris”
Section 5.1.3, “Server Command Options”

cp

Section 16.3.1.2, “Backing Up Raw Data from a Slave”
Section 7.1, “Backup and Recovery Types”
Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”

crash-me

Section 8.12.2, “The MySQL Benchmark Suite”

cron

Section C.5.2.2, “Can't connect to [local] MySQL server”
Section 13.7.2.2, “`CHECK TABLE` Syntax”
Section 14.5.1, “`MyISAM` Startup Options”
Section 5.2.6, “Server Log Maintenance”
Section 7.6.5, “Setting Up a `MyISAM` Table Maintenance Schedule”
Section 3.5, “Using `mysql` in Batch Mode”

csh

Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.4, “Setting Environment Variables”

Section 1.2, “Typographical and Syntax Conventions”

D

[index top [4287]]

date

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

df

Section C.5.1, “How to Determine What Is Causing a Problem”

Directory Utility

Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”

drwtsn32.exe

Section 23.4.1.3, “Using pdb to create a Windows crashdump”

dump

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

E

[index top [4287]]

emerge

Section 2.5.2, “Installing MySQL on Linux using Native Package Manager”

EXIT SINGLE USER MODE

Section 17.5.8, “MySQL Cluster Single User Mode”

G

[index top [4287]]

gcc

Section 23.3.2.5, “Compiling and Installing User-Defined Functions”

Section 22.7.1, “Compiling Programs with libmysqld”

Section 1.4.4, “Enhanced Solaris Support”

Section 2.14.3, “Problems Using the Perl DBI/DBD Interface”

gdb

Section 23.4.1.1, “Compiling MySQL for Debugging”

Section 23.4.1.4, “Debugging mysqld under gdb”

Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

gmake

Section 2.10.2, “Installing MySQL from a Standard Source Distribution”

Section 2.10, “Installing MySQL from Source”

Section 2.9, “Installing MySQL on FreeBSD”

GnuPG

Section 2.1.4.2, “Signature Checking Using GnuPG”

gnutar

Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”

Section 2.10, “Installing MySQL from Source”

gogoc

Section 5.1.9.5, “Obtaining an IPv6 Address from a Broker”

gpg

Section 2.1.4.2, “Signature Checking Using GnuPG”

grep

Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”

Section 3.3.4.7, “Pattern Matching”

groupadd

Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”

Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”

Section 2.5.1, “Installing MySQL from RPM Packages on Linux”

Section 2.7, “Installing MySQL on Solaris and OpenSolaris”

gtar

Section 2.8.1, “General Notes on Installing MySQL on HP-UX”

Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”

Section 2.10, “Installing MySQL from Source”

Section 2.7, “Installing MySQL on Solaris and OpenSolaris”

gunzip

Section 2.10.2, “Installing MySQL from a Standard Source Distribution”

Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”

gzip

Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 1.7, “How to Report Bugs or Problems”
Section 2.4, “Installing MySQL on Mac OS X”

H

[index top [4287]]

hdparm

Section 14.3.4, “InnoDB Startup Options and System Variables”

help contents

Section 4.5.1.4, “mysql Server-Side Help”

hostname

Section C.5.2.2, “Can't connect to [local] MySQL server”

I

[index top [4287]]

icc

Section 2.1.6, “Compiler-Specific Build Characteristics”
Section 23.4, “Debugging and Porting MySQL”

ifconfig

Section 5.1.9.1, “Verifying System Support for IPv6”

innochecksum

Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”
Section 13.7.2.2, “CHECK TABLE Syntax”
Section 4.1, “Overview of MySQL Programs”

iptables

Section 17.5.11.1, “MySQL Cluster Security and Networking Issues”

isamlog

Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.1, “Overview of MySQL Programs”

K

[index top [4287]]

kill

Section C.5.2.2, “Can't connect to [local] MySQL server”
Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section E.6, “Restrictions on XA Transactions”

ksh

Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.4, “Setting Environment Variables”

kswapd

Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”

L

[index top [4287]]

ldconfig

Section 23.3.2.5, “Compiling and Installing User-Defined Functions”

less

Section 4.5.1.2, “mysql Commands”
Section 4.5.1.1, “mysql Options”

ln

Section 8.11.3.1.2, “Using Symbolic Links for MyISAM Tables on Unix”

logger

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

M

[index top [4287]]

m4

Section 2.10, “Installing MySQL from Source”

make

Section 23.2.4.3, “Compiling and Installing Plugin Libraries”
Section 2.10.5, “Dealing with Problems Compiling MySQL”
Section 2.10.2, “Installing MySQL from a Standard Source Distribution”
Section 2.10, “Installing MySQL from Source”
Section 2.9, “Installing MySQL on FreeBSD”

Section 2.14.3, “Problems Using the Perl `DBI/DBD` Interface”

make && make install

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”

make install

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”

Section 23.2.4.3, “Compiling and Installing Plugin Libraries”

Section 2.10.4, “MySQL Source-Configuration Options”

make package

Section 2.10.2, “Installing MySQL from a Standard Source Distribution”

Section 2.10.4, “MySQL Source-Configuration Options”

make realclean

Section 2.14.3, “Problems Using the Perl `DBI/DBD` Interface”

make test

Section 2.10.3, “Installing MySQL from a Development Source Tree”

Section 2.14.1, “Installing Perl on Unix”

Section 23.1.2, “The MySQL Test Suite”

md5

Section 2.1.4.1, “Verifying the MD5 Checksum”

md5sum

Section 2.1.4.1, “Verifying the MD5 Checksum”

memcached

Section 17.1.1, “MySQL Cluster Core Concepts”

Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 15.6, “Using MySQL with `memcached`”

mgmd

Section 17.2, “MySQL Cluster Installation”

mkdir

Section 13.1.10, “`CREATE DATABASE` Syntax”

mklink

Section 8.11.3.1.3, “Using Symbolic Links for Databases on Windows”

more

Section 4.5.1.2, “`mysql` Commands”

Section 4.5.1.1, “`mysql` Options”

msiexec.exe

Section 2.3.4.2, “Automating MySQL Installation on Microsoft Windows using the MSI Package”

msql2mysql

Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”

Section 4.8.2, “`replace` — A String-Replacement Utility”

Section 4.1, “Overview of MySQL Programs”

msql2mysqlC-source-file ...

Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”

mv

Section 5.2.6, “Server Log Maintenance”

Section 5.2.2, “The Error Log”

Section 5.2.3, “The General Query Log”

my_print_defaults

Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 4.7, “MySQL Program Development Utilities”

Section 4.1, “Overview of MySQL Programs”

myisam_ftdump

Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”

Section 12.9, “Full-Text Search Functions”

Section 4.1, “Overview of MySQL Programs”

myisamchk

Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”

Section 4.6.3.2, “`myisamchk` Check Options”

Section 4.6.3.1, “`myisamchk` General Options”

Section 4.6.3.6, “`myisamchk` Memory Usage”

Section 4.6.3.3, “`myisamchk` Repair Options”

Section 13.7.2.1, “`ANALYZE TABLE` Syntax”

Section 13.7.2.2, “`CHECK TABLE` Syntax”

Section 13.2.2, “`DELETE` Syntax”

Section 8.8.2, “`EXPLAIN` Output Format”

Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”

Section 13.7.6.5, “`LOAD INDEX INTO CACHE` Syntax”

Section 14.5.1, “MyISAM Startup Options”

Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 7.6.4, “MyISAM Table Optimization”
Section 14.5.3, “MyISAM Table Storage Formats”
Section 13.7.2.5, “REPAIR TABLE Syntax”
Section 13.7.5.23, “SHOW INDEX Syntax”
Section 13.7.5.37, “SHOW TABLE STATUS Syntax”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 10.5, “Character Set Configuration”
Section 2.3.4.1.2, “Choosing an Install Type”
Section 14.5.3.3, “Compressed Table Characteristics”
Section 14.5.4.1, “Corrupted MyISAM Tables”
Section 7.2, “Database Backup Methods”
Section 23.4.1, “Debugging a MySQL Server”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.9, “mysqlhotcopy — A Database Backup Program”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 14.5.3.2, “Dynamic Table Characteristics”
Section 8.10.5, “External Locking”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section E.10.3, “Limits on Table Size”
Section 18.3.3, “Maintenance of Partitions”
Section 23.4.1.7, “Making a Test Case If You Experience Table Corruption”
Section 4.6.3.5, “Obtaining Table Information with myisamchk”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 4.6.3.4, “Other myisamchk Options”
Section 4.1, “Overview of MySQL Programs”
Section 14.5.4.2, “Problems from Tables Not Being Closed Properly”
Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.1.3, “Server Command Options”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 8.6.3, “Speed of REPAIR TABLE Statements”
Section 14.5.3.1, “Static (Fixed-Length) Table Characteristics”
Section 8.11.1, “System Factors and Startup Parameter Tuning”
Section 14.5, “The MyISAM Storage Engine”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 23.4.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 8.11.3.1.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

myisamchk *.MYI

Section 7.6.3, “How to Repair MyISAM Tables”

myisamchk tbl_name

Section 7.6.2, “How to Check MyISAM Tables for Errors”

myisamlog

Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”

Section 4.1, “Overview of MySQL Programs”

myisampack

Section 4.6.3.3, “myisamchk Repair Options”

Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

Section 13.1.17, “CREATE TABLE Syntax”

Section 14.10.1, “MERGE Table Advantages and Disadvantages”

Section 14.5.3, “MyISAM Table Storage Formats”

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”

Section 14.5.3.3, “Compressed Table Characteristics”

Section 8.10.5, “External Locking”

Section E.10.3, “Limits on Table Size”

Section 4.6.3.5, “Obtaining Table Information with myisamchk”

Section 8.4.1, “Optimizing Data Size”

Section 4.1, “Overview of MySQL Programs”

Section 13.1.17.2, “Silent Column Specification Changes”

Section 14.10, “The MERGE Storage Engine”

Section 14.5, “The MyISAM Storage Engine”

mysql

Section 1.8.5.5, “--' as the Start of a Comment”

Section 4.5.1, “mysql — The MySQL Command-Line Tool”

Section 4.5.1.2, “mysql Commands”

Section 4.5.1.3, “mysql History File”

Section 4.5.1.1, “mysql Options”

Section 4.5.1.4, “mysql Server-Side Help”

Section 4.5.1.6, “mysql Tips”

Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”

Section 13.7.1.3, “GRANT Syntax”

Section 13.8.3, “HELP Syntax”

Section C.5.2.15, “Ignoring user”

Section 13.2.6, “LOAD DATA INFILE Syntax”

Section 13.2.7, “LOAD XML Syntax”

Section 22.8.3.14, “mysql_errno()”

Section 22.8.3.66, “mysql_sqlstate()”

Section C.5.2.8, “Out of memory”

Section C.5.2.10, “`Packet too large`”

Section 18.2.3.1, “`RANGE COLUMNS` partitioning”

Section 13.7.1.5, “`REVOKE` Syntax”

Section 13.2.9.1, “`SELECT ... INTO` Syntax”

Section 14.3.14.2, “`SHOW ENGINE INNODB STATUS` and the `InnoDB` Monitors”

Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”

Section 13.6.7.4, “`SIGNAL` Syntax”

Section 17.5.13.2, “Adding MySQL Cluster Data Nodes Online: Basic procedure”

Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”

Section 17.5.13.1, “Adding MySQL Cluster Data Nodes Online: General Issues”

Section 6.3.2, “Adding User Accounts”

Section 14.3.7, “Backing Up and Recovering an `InnoDB` Database”

Section 7.1, “Backup and Recovery Types”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 2.3.4.1.5, “Changes Made by MySQL Installation Wizard”

Section 2.3.4.1.2, “Choosing an Install Type”

Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”

Section 17.5.2, “Commands in the MySQL Cluster Management Client”

Section 9.6, “Comment Syntax”

Section 10.1.5, “Configuring the Character Set and Collation for Applications”

Section 3.1, “Connecting to and Disconnecting from the Server”

Section 4.2.2, “Connecting to the MySQL Server”

Section 5.1.9.4, “Connecting Using IPv6 Nonlocal Host Addresses”

Section 5.1.9.3, “Connecting Using the IPv6 Local Host Address”

Section 10.1.4, “Connection Character Sets and Collations”

Section 22.8.12, “Controlling Automatic Reconnection Behavior”

Section 2.12.5, “Copying MySQL Databases to Another Machine”

Section 3.3.1, “Creating and Selecting a Database”

Section 14.3.3.2, “Creating the `InnoDB` Tablespace”

Section 2.3.7.6, “Customizing the `PATH` for MySQL Tools”

Section 23.4.2, “Debugging a MySQL Client”

Section 19.1, “Defining Stored Programs”

Section 17.4.5, “`ndb_mgm` — The MySQL Cluster Management Client”

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.4.6, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”

Section 4.5.1.6.3, “Disabling `mysql` Auto-Reconnect”

Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”

Section 2.12.2, “Downgrading MySQL”

Section 6.1.2.1, “End-User Guidelines for Password Security”

Section 3.2, “Entering Queries”

Section 2.13, “Environment Variables”

Section 19.4.2, “Event Scheduler Configuration”

Section 7.3, “Example Backup and Recovery Strategy”

Section 3.6, “Examples of Common Queries”

Section 4.5.1.5, “Executing SQL Statements from a Text File”

Chapter 12, *Functions and Operators*

Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”

Section 14.3.9.9, “How to Cope with Deadlocks”

Section C.5.1, “How to Determine What Is Causing a Problem”

Section 1.7, “How to Report Bugs or Problems”

Section 6.1.5, “How to Run MySQL as a Normal User”

Section 12.14, “Information Functions”

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”

Section 17.2.1.2, “Installing MySQL Cluster from RPM”

Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”

Section 4.2.1, “Invoking MySQL Programs”

Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”

Section 7.4.5.1, “Making a Copy of a Database”

Section 6.1.3, “Making MySQL Secure Against Attackers”

Section 17.5, “Management of MySQL Cluster”

Section 8.12.1, “Measuring the Speed of Expressions and Functions”

Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”

Section 17.5.12.1, “MySQL Cluster Disk Data Objects”

Section 17.2.5, “MySQL Cluster Example with Tables and Data”

Chapter 17, *MySQL Cluster NDB 7.2*

Section 17.6, “MySQL Cluster Replication”

Section 17.6.4, “MySQL Cluster Replication Schema and Tables”

Section 15.7, “MySQL Proxy”

Section 10.6, “MySQL Server Time Zone Support”

Section 17.5.4, “MySQL Server Usage for MySQL Cluster”

Section 17.5.15, “NDB API Statistics Counters and Variables”

Section 8.2.1.3, “Optimizing `LIMIT` Queries”

Section 4.2.3.6, “Option Defaults, Options Expecting Values, and the = Sign”

Section 4.1, “Overview of MySQL Programs”

Section 6.1.2.4, “Password Hashing in MySQL”

Section 6.3.6, “Pluggable Authentication”

Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”

Section 17.6.5, “Preparing the MySQL Cluster for Replication”

Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”

Section 2.11.1.1, “Problems Running `mysql_install_db`”

Section 4.2.3.2, “Program Option Modifiers”

Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”

Section 7.4.4, “Reloading Delimited-Text Format Backups”

Section 7.4.2, “Reloading SQL-Format Backups”

Section C.5.4.1.3, “Resetting the Root Password: Generic Instructions”

Section E.9, “Restrictions on Pluggable Authentication”

Section 2.11.2, “Securing the Initial MySQL Accounts”

Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”

Section 5.1.3, “Server Command Options”

Section C.3, “Server Error Codes and Messages”

Section 5.1.4, “Server System Variables”

Section 5.1.10, “Server-Side Help”

Section 12.17.5, “Spatial Analysis Functions”

Section 4.2.3, “Specifying Program Options”

Section 13.5, “SQL Syntax for Prepared Statements”

Section 2.11.1.2, “Starting and Stopping MySQL Automatically”

Section 2.3.7.7, “Starting MySQL as a Windows Service”

Section 17.6.6, “Starting MySQL Cluster Replication (Single Replication Channel)”

Section 2.3.6.1, “Starting the MySQL Server Instance Configuration Wizard”

Section 9.1.1, “String Literals”

Section 11.4.3, “The `BLOB` and `TEXT` Types”

Section 17.5.10.19, “The `ndbinfo` `transporters` Table”

Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”

Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”

Section 6.3.6.5, “The Socket Peer-Credential Authentication Plugin”

Section 19.3.1, “Trigger Syntax”

Section 14.3.14.4, “Troubleshooting `InnoDB` Data Dictionary Operations”

Chapter 3, *Tutorial*

Section 22.2.4.8, “Tutorial: Using `MySQLScript`”

Section 1.2, “Typographical and Syntax Conventions”

Section 6.3.6.2.2.3, “Unix Password Authentication with Proxy Users and Group Mapping”

Section 6.3.6.2.2.1, “Unix Password Authentication without Proxy Users”

Section 2.11.1, “Unix Postinstallation Procedures”

Section 3.5, “Using `mysql` in Batch Mode”

Section 7.4, “Using `mysqldump` for Backups”

Section 7.3.2, “Using Backups for Recovery”

Section 4.2.3.3, “Using Option Files”

Section 4.2.3.1, “Using Options on the Command Line”

Section 4.2.3.5, “Using Options to Set Program Variables”

Section 23.4.1.6, “Using Server Logs to Find Causes of Errors in `mysqld`”

Section 6.3.8.3, “Using SSL Connections”

Section 17.5.12.2, “Using Symbolic Links with Disk Data Objects”

Section 4.5.1.6.2, “Using the `--safe-updates` Option”

Section 1.4, “What Is New in MySQL 5.5”

Section 2.3.10, “Windows Postinstallation Procedures”

Section 12.11, “XML Functions”

`mysql ...`

Section 23.4.1.1, “Compiling MySQL for Debugging”

`mysql {start|stop}`

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

`mysql-server`

Section 2.9, “Installing MySQL on FreeBSD”

`mysql-test-run.pl`

Section 23.1.2, “The MySQL Test Suite”

`mysql-test-run.pl test_name`

Section 23.1.2, “The MySQL Test Suite”

`mysql.exe`

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”

Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”

`mysql.server`

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 2.5, “Installing MySQL on Linux”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 4.1, “Overview of MySQL Programs”
Section 5.1.3, “Server Command Options”
Section 2.11.1.2, “Starting and Stopping MySQL Automatically”
Section C.5.4.6, “Time Zone Problems”

mysql.server stop

Section 2.11.1.2, “Starting and Stopping MySQL Automatically”

mysql_config

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”
Section 22.8.17, “Building Client Programs”
Section 22.7.1, “Compiling Programs with `libmysqld`”
Section 2.10.5, “Dealing with Problems Compiling MySQL”
Section 4.1, “Overview of MySQL Programs”
Section 23.2.2, “Plugin API Components”
Section 22.8.17.1, “Problems Linking to the MySQL Client Library”

mysql_config options

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

mysql_convert_table_format

Section 4.6.10, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”
Section 4.1, “Overview of MySQL Programs”

mysql_find_rows

Section 4.6.11, “`mysql_find_rows` — Extract SQL Statements from Files”
Section 23.4.1.7, “Making a Test Case If You Experience Table Corruption”
Section 4.1, “Overview of MySQL Programs”

mysql_fix_extensions

Section 4.6.12, “`mysql_fix_extensions` — Normalize Table File Name Extensions”
Section 4.1, “Overview of MySQL Programs”

mysql_fix_extensions data_dir

Section 4.6.12, “`mysql_fix_extensions` — Normalize Table File Name Extensions”

mysql_install_db

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 6.3.2, “Adding User Accounts”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”
Section 2.5.2, “Installing MySQL on Linux using Native Package Manager”
Section 2.4.2, “Installing MySQL on Mac OS X Using Native Packages”
Section 2.7.2, “Installing MySQL on OpenSolaris using IPS”
Section 14.3.15, “Limits on `InnoDB` Tables”
Section 17.6.4, “MySQL Cluster Replication Schema and Tables”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 4.1, “Overview of MySQL Programs”
Section 2.11.1.1, “Problems Running `mysql_install_db`”
Section 2.11.2, “Securing the Initial MySQL Accounts”
Section 5.1.3, “Server Command Options”
Section 5.1.10, “Server-Side Help”
Section 5.3.1, “Setting Up Multiple Data Directories”
Section 2.11.1, “Unix Postinstallation Procedures”

mysql_plugin

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.1, “Overview of MySQL Programs”

mysql_secure_installation

Section 4.4.5, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 2.5.2, “Installing MySQL on Linux using Native Package Manager”
Section 2.7.2, “Installing MySQL on OpenSolaris using IPS”
Section 2.7.1, “Installing MySQL on Solaris using a Solaris `PKG`”
Section 4.1, “Overview of MySQL Programs”
Section 2.11.2, “Securing the Initial MySQL Accounts”

mysql_setpermission

Section 4.6.13, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”
Section 4.1, “Overview of MySQL Programs”
Section 2.11.1, “Unix Postinstallation Procedures”

mysql_setpermissions

Section 4.6.13, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”

mysql_stmt_execute()

Section 5.1.6, “Server Status Variables”

mysql_stmt_prepare()

Section 5.1.6, “Server Status Variables”

mysql_tzinfo_to_sql

Section 4.4.6, “mysql_tzinfo_to_sql — Load the Time Zone Tables”

Section 10.6, “MySQL Server Time Zone Support”

Section 4.1, “Overview of MySQL Programs”

mysql_tzinfo_to_sql arguments

Section 4.4.6, “mysql_tzinfo_to_sql — Load the Time Zone Tables”

mysql_upgrade

Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”

Section 13.1.1, “ALTER DATABASE Syntax”

Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 2.12.2, “Downgrading MySQL”

Section 4.1, “Overview of MySQL Programs”

Section 6.1.2.4, “Password Hashing in MySQL”

Section 21.2.1, “Performance Schema Build Configuration”

Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

Section 2.12.1, “Upgrading MySQL”

Section 2.3.9, “Upgrading MySQL on Windows”

mysql_waitpid

Section 4.6.14, “mysql_waitpid — Kill Process and Wait for Its Termination”

Section 4.1, “Overview of MySQL Programs”

mysql_waitpid()

Section 4.6.14, “mysql_waitpid — Kill Process and Wait for Its Termination”

mysql_zap

Section 4.6.15, “mysql_zap — Kill Processes That Match a Pattern”

Section C.5.2.2, “Can't connect to [local] MySQL server”

Section 4.1, “Overview of MySQL Programs”

mysqlaccess

Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 1.7, “How to Report Bugs or Problems”

Section 4.1, “Overview of MySQL Programs”

Section 2.11.1, “Unix Postinstallation Procedures”

mysqladmin

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

Section C.5.2.2, “Can't connect to [local] MySQL server”

Section 13.1.10, “CREATE DATABASE Syntax”

Section 13.1.21, “DROP DATABASE Syntax”

Section 13.7.6.3, “FLUSH Syntax”

Section 6.3.5, “Assigning Account Passwords”

Section 16.3.1.1, “Backing Up a Slave Using mysqldump”

Section 4.2.2, “Connecting to the MySQL Server”

Section 2.3.7.6, “Customizing the PATH for MySQL Tools”

Section 23.4.1, “Debugging a MySQL Server”

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”

Section C.5.1, “How to Determine What Is Causing a Problem”

Section 7.6.3, “How to Repair MyISAM Tables”

Section 1.7, “How to Report Bugs or Problems”

Section 5.2, “MySQL Server Logs”

Section 4.1, “Overview of MySQL Programs”

Section 6.2.1, “Privileges Provided by MySQL”

Section E.9, “Restrictions on Pluggable Authentication”

Section 5.3.3, “Running Multiple MySQL Instances on Unix”

Section 2.11.2, “Securing the Initial MySQL Accounts”

Section 2.3.7.7, “Starting MySQL as a Windows Service”

Section 2.3.7.5, “Starting MySQL from the Windows Command Line”

Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”

Section 5.1.12, “The Shutdown Process”

Section 8.11.2, “Tuning Server Parameters”

Section 2.11.1, “Unix Postinstallation Procedures”

Section 2.3.9, “Upgrading MySQL on Windows”

Section 4.2.3.3, “Using Option Files”

Section 4.2.3.1, “Using Options on the Command Line”

Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

mysqladmin debug

Section 13.7.1.3, “GRANT Syntax”
Section 23.4.1, “Debugging a MySQL Server”
Section 19.4.5, “Event Scheduler Status”
Section 6.2.1, “Privileges Provided by MySQL”

mysqladmin extended-status

Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.7.5.36, “SHOW STATUS Syntax”

mysqladmin flush-hosts

Section C.5.2.6, “Host ‘host_name’ is blocked”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”
Section 5.1.4, “Server System Variables”

mysqladmin flush-logs

Section 7.3.3, “Backup Strategy Summary”
Section 7.3.1, “Establishing a Backup Policy”
Section 5.2.6, “Server Log Maintenance”
Section 5.2.4, “The Binary Log”
Section 5.2.2, “The Error Log”
Section 16.2.2.1, “The Slave Relay Log”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

mysqladmin flush-privileges

Section 6.2.7, “Causes of Access-Denied Errors”
Section 2.12.5, “Copying MySQL Databases to Another Machine”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 6.2.2, “Privilege System Grant Tables”
Section 2.11.1.1, “Problems Running mysql_install_db”
Section 5.1.3, “Server Command Options”
Section 6.2.6, “When Privilege Changes Take Effect”

mysqladmin flush-tables

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 8.10.5, “External Locking”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 7.6.1, “Using myisamchk for Crash Recovery”

mysqladmin flush-xxx

Section 6.3.2, “Adding User Accounts”

mysqladmin kill

Section 13.7.6.4, “KILL Syntax”
Section C.5.2.9, “MySQL server has gone away”
Section C.5.4.3, “How MySQL Handles a Full Disk”
Section 12.15, “Miscellaneous Functions”
Section 6.2.1, “Privileges Provided by MySQL”

mysqladmin password

Section 6.3.5, “Assigning Account Passwords”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

mysqladmin processlist

Section 13.7.6.4, “KILL Syntax”
Section 22.8.3.43, “mysql_list_processes()”
Section 13.7.5.30, “SHOW PROCESSLIST Syntax”
Section 6.3.2, “Adding User Accounts”
Section 8.12.5, “Examining Thread Information”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 23.1.1, “MySQL Threads”
Section 6.2.1, “Privileges Provided by MySQL”

mysqladmin processlist status

Section 23.4.1, “Debugging a MySQL Server”

mysqladmin refresh

Section 6.3.2, “Adding User Accounts”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 5.2.6, “Server Log Maintenance”

mysqladmin reload

Section 6.3.2, “Adding User Accounts”
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 1.7, “How to Report Bugs or Problems”
Section 6.2.2, “Privilege System Grant Tables”
Section 2.11.1.1, “Problems Running mysql_install_db”
Section 5.1.3, “Server Command Options”
Section 6.3.4, “Setting Account Resource Limits”
Section 6.2.6, “When Privilege Changes Take Effect”

mysqladmin reload version

Section 1.7, “How to Report Bugs or Problems”

mysqladmin shutdown

Section 13.7.1.3, “GRANT Syntax”
Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”

Section 14.3.3.2, “Creating the `InnoDB` Tablespace”
Section 23.4.1.2, “Creating Trace Files”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 7.6.3, “How to Repair `MyISAM` Tables”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 2.4.2, “Installing MySQL on Mac OS X Using Native Packages”
Section 23.4.1.7, “Making a Test Case If You Experience Table Corruption”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.21, “Replication and Temporary Tables”
Section 17.2.6, “Safe Shutdown and Restart of MySQL Cluster”
Section 2.11.1.2, “Starting and Stopping MySQL Automatically”
Section 2.3.7.7, “Starting MySQL as a Windows Service”
Section 5.1.12, “The Shutdown Process”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

`mysqladmin status`

Section 22.8.3.68, “`mysql_stat()`”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”

`mysqladmin variables`

Section C.5.2.9, “MySQL server has gone away”
Section 13.7.5.40, “`SHOW VARIABLES` Syntax”

`mysqladmin variables`

`extended-status processlist`

Section 1.7, “How to Report Bugs or Problems”

`mysqladmin ver`

Section 23.4.1.1, “Compiling MySQL for Debugging”

`mysqladmin version`

Section C.5.2.2, “Can't connect to [local] MySQL server”
Section C.5.2.9, “MySQL server has gone away”
Section 1.7, “How to Report Bugs or Problems”
Section 2.11.1, “Unix Postinstallation Procedures”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

`mysqlanalyze`

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

`mysqlbackup`

Section 14.3.7, “Backing Up and Recovering an `InnoDB` Database”

Section 7.1, “Backup and Recovery Types”
Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

`mysqlbinlog`

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.7.1, “`mysqlbinlog` Hex Dump Format”
Section 4.6.7.2, “`mysqlbinlog` Row Event Display”
Section 13.7.6.1, “`BINLOG` Syntax”
Section 13.7.4, “`SET` Syntax”
Section 13.7.5.3, “`SHOW BINLOG EVENTS` Syntax”
Section 13.7.5.33, “`SHOW RELAYLOG EVENTS` Syntax”
Section 13.4.2.5, “`START SLAVE` Syntax”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 14.3.7, “Backing Up and Recovering an `InnoDB` Database”
Section 5.4.1.2, “Command Probes”
Section 16.4.5, “How to Report Replication Bugs or Problems”
Section C.5.8, “Known Issues in MySQL”
Section 12.15, “Miscellaneous Functions”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 4.1, “Overview of MySQL Programs”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”
Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”
Section 16.4.1.35, “Replication and Variables”
Section 5.2.4, “The Binary Log”
Section 16.2.2.1, “The Slave Relay Log”
Section 16.1.2.2, “Usage of Row-Based Logging and Replication”
Section 7.3.2, “Using Backups for Recovery”
Section 1.4, “What Is New in MySQL 5.5”

`mysqlbinlog binary-log-file | mysql`

Section 23.4.1.7, “Making a Test Case If You Experience Table Corruption”

`mysqlbinlog | mysql`

Section C.5.8, “Known Issues in MySQL”

`mysqlbug`

Section 4.4.2, “`mysqlbug` — Generate Bug Report”

mysqlcheck

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 13.1.1, “ALTER DATABASE Syntax”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 18.3.3, “Maintenance of Partitions”
Section 9.2.3, “Mapping of Identifiers to File Names”
Section 4.1, “Overview of MySQL Programs”
Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”
Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.1.4, “Server System Variables”
Section 14.5, “The MyISAM Storage Engine”

mysqld

Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.1, “myisamchk General Options”
Section 4.3.1, “mysqld — The MySQL Server”
Section 5.4.1, “mysqld DTrace Probe Reference”
Section C.5.2.18, “'File' Not Found and Similar Errors”
Section C.5.2.2, “Can't connect to [local] MySQL server”
Section C.5.2.13, “Can't create/write to file”
Section C.5.2.17, “Can't initialize character set”
Section C.5.2.4, “Client does not support authentication protocol”
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 13.1.17, “CREATE TABLE Syntax”
Section 13.7.6.3, “FLUSH Syntax”
Section C.5.2.6, “Host 'host_name' is blocked”
Section C.5.2.15, “Ignoring user”
Section 14.3.14.3, “InnoDB General Troubleshooting”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.7.6.4, “KILL Syntax”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 14.5.1, “MyISAM Startup Options”
Section C.5.2.9, “MySQL server has gone away”
Section 22.8.3.1, “mysql_affected_rows()”
Section 22.8.3.49, “mysql_options()”
Section 13.7.2.4, “OPTIMIZE TABLE Syntax”

Section C.5.2.10, “Packet too large”
Section 13.7.2.5, “REPAIR TABLE Syntax”
Section 13.4.2.3, “RESET SLAVE Syntax”
Section 13.2.9.1, “SELECT ... INTO Syntax”
Section 13.3.6, “SET TRANSACTION Syntax”
Section 14.3.14.2, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”
Section 13.7.5.16, “SHOW ENGINE Syntax”
Section 13.7.5.40, “SHOW VARIABLES Syntax”
Section C.5.2.7, “Too many connections”
Section 23.3.2, “Adding a New User-Defined Function”
Section 23.3, “Adding New Functions to MySQL”
Section 14.3.6, “Adding, Removing, or Resizing InnoDB Data and Log Files”
Section 14.3.7, “Backing Up and Recovering an InnoDB Database”
Section 16.1.3.4, “Binary Log Options and Variables”
Section 5.2.4.1, “Binary Logging Formats”
Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 2.1.2.2, “Choosing a Distribution Format”
Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 9.6, “Comment Syntax”
Section C.5.2.11, “Communication Errors and Aborted Connections”
Section 23.3.2.5, “Compiling and Installing User-Defined Functions”
Section 23.4.1.1, “Compiling MySQL for Debugging”
Section 14.3.2, “Configuring InnoDB”
Section 6.3.8.2, “Configuring MySQL for SSL”
Section 14.5.4.1, “Corrupted MyISAM Tables”
Section 14.3.3.2, “Creating the InnoDB Tablespace”
Section 23.4.1.2, “Creating Trace Files”
Section 23.4.1.4, “Debugging mysqld under gdb”
Section 23.4.1, “Debugging a MySQL Server”
Section 23.4, “Debugging and Porting MySQL”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 17.3.2.7, “Defining SQL and Other API Nodes in a MySQL Cluster”
Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”
Section 17.4.21, “ndb_show_tables — Display List of NDB Tables”
Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

Section 2.13, “Environment Variables”

Section 8.10.5, “External Locking”

Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”

Section 14.3.7.2, “Forcing `InnoDB` Recovery”

Section 17.6.2, “General Requirements for MySQL Cluster Replication”

Section 8.12.5.2, “General Thread States”

Section 8.4.3.1, “How MySQL Opens and Closes Tables”

Section 8.11.4.1, “How MySQL Uses Memory”

Section 8.2.1.4, “How to Avoid Full Table Scans”

Section C.5.1, “How to Determine What Is Causing a Problem”

Section 7.6.3, “How to Repair `MyISAM` Tables”

Section 1.7, “How to Report Bugs or Problems”

Section 6.1.5, “How to Run MySQL as a Normal User”

Section 9.2.2, “Identifier Case Sensitivity”

Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”

Section 12.14, “Information Functions”

Section 17.2.3, “Initial Configuration of MySQL Cluster”

Section 17.2.4, “Initial Startup of MySQL Cluster”

Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”

Section 17.2.1.2, “Installing MySQL Cluster from RPM”

Section 17.2.1, “Installing MySQL Cluster on Linux”

Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”

Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”

Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”

Section 2.5.1, “Installing MySQL from RPM Packages on Linux”

Section 2.4.2, “Installing MySQL on Mac OS X Using Native Packages”

Section 2.7, “Installing MySQL on Solaris and OpenSolaris”

Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 23.4.1.7, “Making a Test Case If You Experience Table Corruption”

Section 6.1.3, “Making MySQL Secure Against Attackers”

Section 17.5, “Management of MySQL Cluster”

Section 12.15, “Miscellaneous Functions”

Section 5.2.4.3, “Mixed Binary Logging Format”

Section 17.3.4.1, “MySQL Cluster `mysqld` Option and Variable Reference”

Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”

Section 17.3.2.1, “MySQL Cluster Configuration: Basic Example”

Section 17.1.1, “MySQL Cluster Core Concepts”

Section 17.1.4, “MySQL Cluster Development History”

Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”

Section 17.2, “MySQL Cluster Installation”

Chapter 17, *MySQL Cluster NDB 7.2*

Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”

Section 17.1, “MySQL Cluster Overview”

Section 17.4, “MySQL Cluster Programs”

Section 17.6, “MySQL Cluster Replication”

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

Section 17.6.4, “MySQL Cluster Replication Schema and Tables”

Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”

Section 17.3.4.4, “MySQL Cluster Status Variables”

Section 17.3.4.3, “MySQL Cluster System Variables”

Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”

Chapter 5, *MySQL Server Administration*

Section 4.3, “MySQL Server and Server-Startup Programs”

Section 5.2, “MySQL Server Logs”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

Section 10.6, “MySQL Server Time Zone Support”

Section 17.5.4, “MySQL Server Usage for MySQL Cluster”

Section 17.1.5, “MySQL Server using `InnoDB` Compared with MySQL Cluster”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 23.1.1, “MySQL Threads”
Section 17.5.15, “NDB API Statistics Counters and Variables”
Section C.5.6, “Optimizer-Related Issues”
Section 22.7.3, “Options with the Embedded Server”
Section 17.3.3, “Overview of MySQL Cluster Configuration Parameters”
Section 4.1, “Overview of MySQL Programs”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.6.5, “Preparing the MySQL Cluster for Replication”
Section 6.2.2, “Privilege System Grant Tables”
Section 14.5.4.2, “Problems from Tables Not Being Closed Properly”
Section 2.11.1.1, “Problems Running `mysql_install_db`”
Section C.5.3.1, “Problems with File Permissions”
Section 4.2.3.2, “Program Option Modifiers”
Section 8.9.3.3, “Query Cache Configuration”
Section 17.3.2.2, “Recommended Starting Configuration for MySQL Cluster”
Section 16.1.3.1, “Replication and Binary Logging Option and Variable Reference”
Section 16.1.3, “Replication and Binary Logging Options and Variables”
Section 16.1.3.2, “Replication Master Options and Variables”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section C.5.4.1.3, “Resetting the Root Password: Generic Instructions”
Section C.5.4.1.2, “Resetting the Root Password: Unix Systems”
Section C.5.5.5, “Rollback Failure for Nontransactional Tables”
Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 1.8.3, “Running MySQL in ANSI Mode”
Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”
Section 6.1.4, “Security-Related `mysqld` Options and Variables”
Section 2.3.7.3, “Selecting a MySQL Server Type”
Section 1.8.2, “Selecting SQL Modes”
Section 10.1.3.1, “Server Character Set and Collation”
Section 5.1.3, “Server Command Options”
Section 23.2.4.2.2, “Server Plugin Status and System Variables”
Section 5.1.11, “Server Response to Signals”
Section 5.1.7, “Server SQL Modes”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 10.2, “Setting the Error Message Language”
Section 7.6.5, “Setting Up a `MyISAM` Table Maintenance Schedule”
Section 2.11.1.2, “Starting and Stopping MySQL Automatically”
Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”
Section 5.3.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 5.3.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.3.7.7, “Starting MySQL as a Windows Service”
Section 17.6.6, “Starting MySQL Cluster Replication (Single Replication Channel)”
Section 2.3.7.5, “Starting MySQL from the Windows Command Line”
Section 10.6.1, “Staying Current with Time Zone Changes”
Section 8.10.2, “Table Locking Issues”
Section C.5.2.19, “Table-Corruption Issues”
Section 2.3.7.8, “Testing The MySQL Installation”
Section 14.9, “The `BLACKHOLE` Storage Engine”
Section 14.5, “The `MyISAM` Storage Engine”
Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”
Section 5.2.4, “The Binary Log”
Section 5.2.2, “The Error Log”
Section 5.2.3, “The General Query Log”
Section 8.9.3, “The MySQL Query Cache”
Section 5.1, “The MySQL Server”
Section 23.1.2, “The MySQL Test Suite”
Section 5.2.5, “The Slow Query Log”
Section C.5.4.6, “Time Zone Problems”
Section 5.4, “Tracing `mysqld` Using DTrace”
Section 14.3.3.3, “Troubleshooting `InnoDB` I/O Problems”
Section 2.3.8, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 8.11.2, “Tuning Server Parameters”
Section 1.2, “Typographical and Syntax Conventions”
Section 2.11.1, “Unix Postinstallation Procedures”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”
Section 2.12.1, “Upgrading MySQL”
Section 2.3.9, “Upgrading MySQL on Windows”
Section 23.3.2.6, “User-Defined Function Security Precautions”
Section 7.6.1, “Using `myisamchk` for Crash Recovery”
Section 23.4.1.3, “Using `pdb` to create a Windows crashdump”
Section 23.4.1.5, “Using a Stack Trace”
Section 4.2.3.3, “Using Option Files”
Section 14.3.3, “Using Per-Table Tablespace”
Section 23.4.1.6, “Using Server Logs to Find Causes of Errors in `mysqld`”

Section 8.11.3.1.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 8.11.3.1.3, “Using Symbolic Links for Databases on Windows”
Section 2.4.5, “Using the Bundled MySQL on Mac OS X Server”
Section 17.6.7, “Using Two Replication Channels for MySQL Cluster Replication”
Section 1.4, “What Is New in MySQL 5.5”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”
Section 6.2.6, “When Privilege Changes Take Effect”
Section C.5.4.4, “Where MySQL Stores Temporary Files”
Section 23.2.4, “Writing Plugins”

mysqld mysqld.trace

Section 23.4.1.2, “Creating Trace Files”

mysqld-

Section 2.12.1, “Upgrading MySQL”

mysqld-debug

Section 2.1.2.2, “Choosing a Distribution Format”
Section 23.4.1.2, “Creating Trace Files”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”
Section 2.3.7.3, “Selecting a MySQL Server Type”
Section 5.3.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”

mysqld.exe

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”
Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”

mysqld_multi

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.1, “Overview of MySQL Programs”
Section 5.3.3, “Running Multiple MySQL Instances on Unix”
Section 1.4, “What Is New in MySQL 5.5”

mysqld_safe

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section C.5.2.18, “`'File' Not Found` and Similar Errors”

Section 14.3.14.3, “InnoDB General Troubleshooting”
Section C.5.2.10, “`Packet too large`”
Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”
Section 23.4.1.1, “Compiling MySQL for Debugging”
Section 14.3.3.2, “Creating the InnoDB Tablespace”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 8.11.4.2, “Enabling Large Page Support”
Section C.5.4.5, “How to Protect or Change the MySQL Unix Socket File”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”
Section 10.6, “MySQL Server Time Zone Support”
Section 4.2.3.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 4.1, “Overview of MySQL Programs”
Section C.5.3.1, “Problems with File Permissions”
Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 5.3.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 2.11.1.2, “Starting and Stopping MySQL Automatically”
Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”
Section 5.2.2, “The Error Log”
Section C.5.4.6, “Time Zone Problems”
Section 8.11.2, “Tuning Server Parameters”
Section 2.11.1, “Unix Postinstallation Procedures”
Section 4.2.3.3, “Using Option Files”

mysqld_safe options

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

mysqldump

Section 4.5.1.1, “`mysql` Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.5, “`mysqldump` Tips”
Section 14.3.5.4, “`FOREIGN KEY` Constraints”
Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 13.2.7, “`LOAD XML` Syntax”
Section 13.7.3.4, “`UNINSTALL PLUGIN` Syntax”
Section 11.3.4, “`YEAR(2)` Limitations and Migrating to `YEAR(4)`”

Section 14.3.6, “Adding, Removing, or Resizing `InnoDB` Data and Log Files”

Section 16.3.1.3, “Backing Up a Master or Slave by Making It Read Only”

Section 16.3.1.1, “Backing Up a Slave Using `mysqldump`”

Section 14.3.7, “Backing Up and Recovering an `InnoDB` Database”

Chapter 7, *Backup and Recovery*

Section 7.1, “Backup and Recovery Types”

Section 7.3.3, “Backup Strategy Summary”

Section 8.5.4, “Bulk Data Loading for `InnoDB` Tables”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”

Section 2.3.4.1.2, “Choosing an Install Type”

Section 4.2.2, “Connecting to the MySQL Server”

Section 7.4.5.2, “Copy a Database from one Server to Another”

Section 2.12.5, “Copying MySQL Databases to Another Machine”

Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

Section 2.3.7.6, “Customizing the `PATH` for MySQL Tools”

Section 7.2, “Database Backup Methods”

Section 14.3.12.3, “Defragmenting a Table”

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

Section 4.6.1, “`innochecksum` — Offline `InnoDB` File Checksum Utility”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”

Section 2.12.2, “Downgrading MySQL”

Section 7.4.3, “Dumping Data in Delimited-Text Format with `mysqldump`”

Section 7.4.1, “Dumping Data in SQL Format with `mysqldump`”

Section 7.4.5.3, “Dumping Stored Programs”

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”

Section 7.3.1, “Establishing a Backup Policy”

Section 7.3, “Example Backup and Recovery Strategy”

Section 1.8.5.4, “Foreign Key Differences”

Section 1.7, “How to Report Bugs or Problems”

Section 16.1.1, “How to Set Up Replication”

Section 9.2.2, “Identifier Case Sensitivity”

Section 2.6, “Installing MySQL Using Unbreakable Linux Network (ULN)”

Section 7.4.5.1, “Making a Copy of a Database”

Section 9.2.3, “Mapping of Identifiers to File Names”

Section 14.3.8, “Moving or Copying `InnoDB` Tables to Another Machine”

Section 17.2.5, “MySQL Cluster Example with Tables and Data”

Section 17.1, “MySQL Cluster Overview”

Section 5.2, “MySQL Server Logs”

Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”

Section 17.5.3, “Online Backup of MySQL Cluster”

Section 4.1, “Overview of MySQL Programs”

Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”

Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”

Section 17.6.5, “Preparing the MySQL Cluster for Replication”

Section C.5.5.8, “Problems with Floating-Point Values”

Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”

Section 7.4.4, “Reloading Delimited-Text Format Backups”

Section 7.4.2, “Reloading SQL-Format Backups”

Section 16.3.4, “Replicating Different Databases to Different Slaves”

Section E.8, “Restrictions on Performance Schema”

Section E.5, “Restrictions on Views”

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”

Section 5.2.6, “Server Log Maintenance”

Section 5.1.7, “Server SQL Modes”

Section 5.1.4, “Server System Variables”

Section 16.1.1.8, “Setting Up Replication with Existing Data”

Section C.5.5.7, “Solving Problems with No Matching Rows”

Section 4.2.3, “Specifying Program Options”

Section 2.3.7.7, “Starting MySQL as a Windows Service”

Section 2.3.6.1, “Starting the MySQL Server Instance Configuration Wizard”

Section 11.4.3, “The `BLOB` and `TEXT` Types”

Section 8.9.1, “The `InnoDB` Buffer Pool”

Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

Section 10.1.11, “Upgrading from Previous to Current Unicode Support”

Section 7.4, “Using `mysqldump` for Backups”

Section 7.4.5.5, “Using `mysqldump` to Test for Upgrade Incompatibilities”

Section 14.3.3, “Using Per-Table Tablespaces”

Section 16.3.1, “Using Replication for Backups”

Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”

Section 1.4, “What Is New in MySQL 5.5”

Section 12.11, “XML Functions”

mysqldump mysql

Section 6.2.7, “Causes of Access-Denied Errors”

mysqldumpslow

Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”

Section 4.1, “Overview of MySQL Programs”

Section 5.2.5, “The Slow Query Log”

mysqlhotcopy

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

Chapter 7, *Backup and Recovery*

Section 7.1, “Backup and Recovery Types”

Section 7.2, “Database Backup Methods”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.1, “Overview of MySQL Programs”

Section 1.4, “What Is New in MySQL 5.5”

mysqlhotcopy arguments

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

mysqlimport

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 13.2.6, “`LOAD DATA INFILE` Syntax”

Section 7.1, “Backup and Recovery Types”

Section 2.12.5, “Copying MySQL Databases to Another Machine”

Section 7.2, “Database Backup Methods”

Section 2.12.2, “Downgrading MySQL”

Section 4.1, “Overview of MySQL Programs”

Section 7.4.4, “Reloading Delimited-Text Format Backups”

Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”

MySQLInstanceConfig.exe

Section 2.3.6.13, “MySQL Server Instance Config Wizard: Creating an Instance from the Command Line”

mysqloptimize

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

mysqlrepair

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

mysqlshow

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 13.7.5.15, “`SHOW DATABASES` Syntax”

Section 13.7.5.23, “`SHOW INDEX` Syntax”

Section 13.7.5.37, “`SHOW TABLE STATUS` Syntax”

Section 4.2.2, “Connecting to the MySQL Server”

Section 4.1, “Overview of MySQL Programs”

Section 2.3.7.8, “Testing The MySQL Installation”

Section 2.3.10, “Windows Postinstallation Procedures”

mysqlshow db_name

Section 13.7.5.38, “`SHOW TABLES` Syntax”

mysqlshow db_name tbl_name

Section 13.7.5.6, “`SHOW COLUMNS` Syntax”

mysqlshow mysql user

Section C.5.2.15, “Ignoring user”

mysqlslap

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.1, “Overview of MySQL Programs”

Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”

Section 8.12.3, “Using Your Own Benchmarks”

mysqltest

Section 23.1.2, “The MySQL Test Suite”

N

[[index top \[4287\]](#)]

ndb_config

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

Section 17.4, “MySQL Cluster Programs”

ndb_config options

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

ndb_cpcd

Section 17.4.7, “`ndb_cpcd` — Automate Testing for NDB Development”

ndb_cpcd options

Section 17.4.7, “`ndb_cpcd` — Automate Testing for NDB Development”

ndb_delete_all

Section 17.4.8, “[ndb_delete_all](#) — Delete All Rows from an NDB Table”

ndb_delete_all options

Section 17.4.8, “[ndb_delete_all](#) — Delete All Rows from an NDB Table”

ndb_desc

Section 17.4.9, “[ndb_desc](#) — Describe NDB Tables”

Section 18.2.5, “[KEY](#) Partitioning”

Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”

Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”

Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”

Section 17.5.12.1, “MySQL Cluster Disk Data Objects”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

Section 20.29.1, “The `INFORMATION_SCHEMA.FILES` Table”

Section 20.12, “The `INFORMATION_SCHEMA.PARTITIONS` Table”

Section 17.5.10.4, “The `ndbinfo.cluster_operations` Table”

Section 17.5.10.15, “The `ndbinfo.server_operations` Table”

ndb_desc options

Section 17.4.9, “[ndb_desc](#) — Describe NDB Tables”

ndb_drop_index

Section 17.4.10, “[ndb_drop_index](#) — Drop Index from an NDB Table”

ndb_drop_index options

Section 17.4.10, “[ndb_drop_index](#) — Drop Index from an NDB Table”

ndb_drop_table

Section 17.4.11, “[ndb_drop_table](#) — Drop an NDB Table”

Section 17.4.10, “[ndb_drop_index](#) — Drop Index from an NDB Table”

ndb_drop_table options

Section 17.4.11, “[ndb_drop_table](#) — Drop an NDB Table”

ndb_error_reporter

Section 17.4.12, “[ndb_error_reporter](#) — NDB Error-Reporting Utility”

ndb_error_reporter options

Section 17.4.12, “[ndb_error_reporter](#) — NDB Error-Reporting Utility”

ndb_index_stat

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

ndb_index_stat options

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

ndb_mgm

Section 17.4.5, “[ndb_mgm](#) — The MySQL Cluster Management Client”

Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”

Section 17.5.13.1, “Adding MySQL Cluster Data Nodes Online: General Issues”

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”

Section 17.5.2, “Commands in the MySQL Cluster Management Client”

Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”

Section 17.4.4, “[ndb_mgmd](#) — The MySQL Cluster Management Server Daemon”

Section 17.4.18, “[ndb_restore](#) — Restore a MySQL Cluster Backup”

Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”

Section 17.5.6, “Event Reports Generated in MySQL Cluster”

Section 17.2.4, “Initial Startup of MySQL Cluster”

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”

Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”

Section 17.2.1.2, “Installing MySQL Cluster from RPM”

Section 17.2.1, “Installing MySQL Cluster on Linux”

Section 17.1.1, “MySQL Cluster Core Concepts”

Section 17.5.6.1, “MySQL Cluster Logging Management Commands”

Chapter 17, *MySQL Cluster NDB 7.2*

Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”

Section 17.4, “MySQL Cluster Programs”

Section 17.5.11.1, “MySQL Cluster Security and Networking Issues”

Section 17.5.8, “MySQL Cluster Single User Mode”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

Section 17.5.3, “Online Backup of MySQL Cluster”

Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”
Section 17.2.6, “Safe Shutdown and Restart of MySQL Cluster”
Section 17.5.10.1, “The `ndbinfo arbitrator_validity_detail` Table”
Section 17.5.10.11, “The `ndbinfo membership` Table”
Section 17.5.10.12, “The `ndbinfo memoryusage` Table”
Section 17.5.10.13, “The `ndbinfo nodes` Table”
Section 17.5.10.19, “The `ndbinfo transporters` Table”
Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”

ndb_mgm options

Section 17.4.5, “`ndb_mgm` — The MySQL Cluster Management Client”

ndb_mgm.exe

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”

ndb_mgmd

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”
Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 17.3.2.5, “Defining a MySQL Cluster Management Server”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”
Section 17.2.4, “Initial Startup of MySQL Cluster”
Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 17.2.1, “Installing MySQL Cluster on Linux”
Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”
Section 17.3.2.1, “MySQL Cluster Configuration: Basic Example”
Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.5.6.1, “MySQL Cluster Logging Management Commands”

Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 17.4, “MySQL Cluster Programs”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 17.3.3, “Overview of MySQL Cluster Configuration Parameters”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.3.1, “Quick Test Setup of MySQL Cluster”
Section 17.2.6, “Safe Shutdown and Restart of MySQL Cluster”
Section 17.5.1, “Summary of MySQL Cluster Start Phases”
Section 17.3.2.3, “The MySQL Cluster Connectstring”

ndb_mgmd options

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”

ndb_mgmd.exe

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”
Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”

ndb_print_backup_file

Section 17.4.14, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 17.4.15, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 17.4.16, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 17.4.17, “`ndbd_redo_log_reader` — Check and Print Content of Cluster Redo Log”

ndb_print_backup_file file_name

Section 17.4.14, “`ndb_print_backup_file` — Print NDB Backup File Contents”

ndb_print_schema_file

Section 17.4.15, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

Section 17.4.14, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 17.4.16, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 17.4.17, “`ndbd_redo_log_reader` — Check and Print Content of Cluster Redo Log”

`ndb_print_schema_file` `file_name`

Section 17.4.15, “`ndb_print_schema_file` — Print NDB Schema File Contents”

`ndb_print_sys_file`

Section 17.4.16, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 17.4.14, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 17.4.15, “`ndb_print_schema_file` — Print NDB Schema File Contents”

`ndb_print_sys_file file_name`

Section 17.4.16, “`ndb_print_sys_file` — Print NDB System File Contents”

`ndb_restore`

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 7.1, “Backup and Recovery Types”
Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”
Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”
Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.1, “MySQL Cluster Overview”
Section 17.4, “MySQL Cluster Programs”
Section 17.6.4, “MySQL Cluster Replication Schema and Tables”
Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”
Section 17.5.8, “MySQL Cluster Single User Mode”
Section 17.5.3, “Online Backup of MySQL Cluster”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”

`ndb_restore options`

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

`ndb_schema_backup_file`

Section 17.4.15, “`ndb_print_schema_file` — Print NDB Schema File Contents”

`ndb_select_all`

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”
Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”
Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”

`ndb_select_count`

Section 17.4.20, “`ndb_select_count` — Print Row Counts for NDB Tables”

`ndb_show_tables`

Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”
Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”
Section 17.4, “MySQL Cluster Programs”
Section 17.6.4, “MySQL Cluster Replication Schema and Tables”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 17.5.10.4, “The `ndbinfo cluster_operations` Table”
Section 17.5.10.15, “The `ndbinfo server_operations` Table”

`ndb_size.pl`

Section 17.4.22, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 2.14, “Perl Installation Notes”

`ndb_size.pl arguments`

Section 17.4.22, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

`ndb_waiter`

Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”

`ndb_waiter options`

Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”

ndbd

Section 17.4.1, “[ndbd — The MySQL Cluster Data Node Daemon](#)”

Section 17.5.13.2, “[Adding MySQL Cluster Data Nodes Online: Basic procedure](#)”

Section 17.5.13.3, “[Adding MySQL Cluster Data Nodes Online: Detailed Example](#)”

Section 17.2.1.3, “[Building MySQL Cluster from Source on Linux](#)”

Section 17.5.2, “[Commands in the MySQL Cluster Management Client](#)”

Section 17.3.2.6, “[Defining MySQL Cluster Data Nodes](#)”

Section 17.4.4, “[ndb_mgmd — The MySQL Cluster Management Server Daemon](#)”

Section 17.4.18, “[ndb_restore — Restore a MySQL Cluster Backup](#)”

Section 17.4.23, “[ndb_waiter — Wait for MySQL Cluster to Reach a Given Status](#)”

Section 17.4.3, “[ndbmt d — The MySQL Cluster Data Node Daemon \(Multi-Threaded\)](#)”

Section 17.2.4, “[Initial Startup of MySQL Cluster](#)”

Section 17.2.1.1, “[Installing a MySQL Cluster Binary Release on Linux](#)”

Section 17.2.1.2, “[Installing MySQL Cluster from RPM](#)”

Section 17.2.1, “[Installing MySQL Cluster on Linux](#)”

Section 17.5, “[Management of MySQL Cluster](#)”

Section 17.6.9, “[MySQL Cluster Backups With MySQL Cluster Replication](#)”

Section 17.3.2.1, “[MySQL Cluster Configuration: Basic Example](#)”

Section 17.1.1, “[MySQL Cluster Core Concepts](#)”

Section 17.3.3.1, “[MySQL Cluster Data Node Configuration Parameters](#)”

Section 17.2, “[MySQL Cluster Installation](#)”

Section 17.3.5.2, “[MySQL Cluster Interconnects and Performance](#)”

Section 17.1.2, “[MySQL Cluster Nodes, Node Groups, Replicas, and Partitions](#)”

Section 17.4, “[MySQL Cluster Programs](#)”

Section 17.3.4.2, “[MySQL Server Options for MySQL Cluster](#)”

Section 17.3.3, “[Overview of MySQL Cluster Configuration Parameters](#)”

Section 17.5.5, “[Performing a Rolling Restart of a MySQL Cluster](#)”

Section 17.6.9.2, “[Point-In-Time Recovery Using MySQL Cluster Replication](#)”

Section 17.3.1, “[Quick Test Setup of MySQL Cluster](#)”

Section 17.2.6, “[Safe Shutdown and Restart of MySQL Cluster](#)”

Section 17.3.2.11, “[SCI Transport Connections in MySQL Cluster](#)”

Section 17.5.1, “[Summary of MySQL Cluster Start Phases](#)”

Section 17.5.10.13, “[The ndbinfo nodes Table](#)”

Section 17.5.6.3, “[Using CLUSTERLOG STATISTICS in the MySQL Cluster Management Client](#)”

Section 17.3.5, “[Using High-Speed Interconnects with MySQL Cluster](#)”

ndbd options

Section 17.4.1, “[ndbd — The MySQL Cluster Data Node Daemon](#)”

ndbd.exe

Section 17.2.2.3, “[Initial Startup of MySQL Cluster on Windows](#)”

Section 17.2.2.1, “[Installing MySQL Cluster on Windows from a Binary Release](#)”

Section 17.2.2.4, “[Installing MySQL Cluster Processes as Windows Services](#)”

ndbd_redo_log_reader

Section 17.4.17, “[ndbd_redo_log_reader — Check and Print Content of Cluster Redo Log](#)”

ndbd_redo_log_reader

file_name

Section 17.4.17, “[ndbd_redo_log_reader — Check and Print Content of Cluster Redo Log](#)”

ndbinfo_select_all

Section 17.4.2, “[ndbinfo_select_all — Select From ndbinfo Tables](#)”

ndbinfo_select_all options

Section 17.4.2, “[ndbinfo_select_all — Select From ndbinfo Tables](#)”

ndbmt d

Section 17.4.3, “[ndbmt d — The MySQL Cluster Data Node Daemon \(Multi-Threaded\)](#)”

Section 17.5.13.2, “[Adding MySQL Cluster Data Nodes Online: Basic procedure](#)”

Section 17.5.13.3, “[Adding MySQL Cluster Data Nodes Online: Detailed Example](#)”

Section 17.2.1.3, “[Building MySQL Cluster from Source on Linux](#)”

Section 17.3.2.6, “[Defining MySQL Cluster Data Nodes](#)”

Section 17.4.4, “[ndb_mgmd — The MySQL Cluster Management Server Daemon](#)”

Section 17.4.1, “[ndbd — The MySQL Cluster Data Node Daemon](#)”

Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 17.2.1, “Installing MySQL Cluster on Linux”
Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.3.3.1, “MySQL Cluster Data Node Configuration Parameters”
Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 17.4, “MySQL Cluster Programs”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”
Section 17.2.6, “Safe Shutdown and Restart of MySQL Cluster”
Section 17.5.10.13, “The `ndbinfo nodes` Table”

ndbmt options

Section 17.4.3, “`ndbmt` — The MySQL Cluster Data Node Daemon (Multi-Threaded)”

ndbmt.exe

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”
Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”

NET

Section 2.3.7.7, “Starting MySQL as a Windows Service”

NET START

Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”
Section 5.3.2.2, “Starting Multiple MySQL Instances as Windows Services”

net start

Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”

NET START MySQL

Section 2.3.7.7, “Starting MySQL as a Windows Service”
Section 2.3.8, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 2.3.9, “Upgrading MySQL on Windows”

NET STOP

Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”

Section 5.3.2.2, “Starting Multiple MySQL Instances as Windows Services”

net stop

Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”

NET STOP MYSQL

Section 17.2.6, “Safe Shutdown and Restart of MySQL Cluster”

NET STOP MySQL

Section 2.3.7.7, “Starting MySQL as a Windows Service”

Netinfo Manager

Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”

nm

Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”
Section 23.4.1.5, “Using a Stack Trace”

numactl

Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”

O

[index top [4287]]

openssl

Section 6.3.8.5, “Setting Up SSL Certificates and Keys for MySQL”

openssl md5 package_name

Section 2.1.4.1, “Verifying the MD5 Checksum”

P

[index top [4287]]

perror

Section 4.8.1, “`perror` — Explain Error Codes”
Section C.5.2.18, “`'File' Not Found` and Similar Errors”
Section C.5.2.13, “`Can't create/write to file`”
Section 7.6.3, “How to Repair `MyISAM` Tables”
Section 14.3.13.2, “Operating System Error Codes”
Section 4.1, “Overview of MySQL Programs”
Section C.1, “Sources of Error Information”

pfexec

Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”

Section 2.7.2, “Installing MySQL on OpenSolaris using IPS”

PGP

Section 2.1.4.2, “Signature Checking Using `GnuPG`”

ping6

Section 5.1.9.5, “Obtaining an IPv6 Address from a Broker”

pkg

Section 2.7.2, “Installing MySQL on OpenSolaris using IPS”

pkgadd

Section 2.7.1, “Installing MySQL on Solaris using a Solaris `PKG`”

pkgrm

Section 2.7.1, “Installing MySQL on Solaris using a Solaris `PKG`”

ppm

Section 2.14, “Perl Installation Notes”

ps

Section 4.6.15, “`mysql_zap` — Kill Processes That Match a Pattern”

Section 6.1.2.1, “End-User Guidelines for Password Security”

Section 8.11.4.1, “How MySQL Uses Memory”

Section C.5.1, “How to Determine What Is Causing a Problem”

Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”

ps auxw

Section 4.2.2, “Connecting to the MySQL Server”

ps xa | grep mysqld

Section C.5.2.2, “Can't connect to [local] MySQL server”

R

[index top [4287]]

rename

Section 5.2.6, “Server Log Maintenance”

Section 5.2.2, “The Error Log”

Section 5.2.3, “The General Query Log”

replace

Section 1.8.5.5, “`--` as the Start of a Comment”

Section 4.8.2, “`replace` — A String-Replacement Utility”

Section 4.7.1, “`mysql2mysql` — Convert mSQL Programs for Use with MySQL”

Section 4.1, “Overview of MySQL Programs”

Section 16.3.3, “Using Replication for Scale-Out”

replace arguments

Section 4.8.2, “`replace` — A String-Replacement Utility”

resolve_stack_dump

Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”

Section 4.1, “Overview of MySQL Programs”

Section 23.4.1.5, “Using a Stack Trace”

resolveip

Section 4.8.3, “`resolveip` — Resolve Host name to IP Address or Vice Versa”

Section 4.1, “Overview of MySQL Programs”

rm

Section 13.4.1.1, “`PURGE BINARY LOGS` Syntax”

rpm

Section 2.10.2, “Installing MySQL from a Standard Source Distribution”

Section 2.1.4.2, “Signature Checking Using `GnuPG`”

Section 2.1.4.3, “Signature Checking Using `RPM`”

rpmbuild

Section 2.10.2, “Installing MySQL from a Standard Source Distribution”

Section 2.10, “Installing MySQL from Source”

rsync

Section 7.1, “Backup and Recovery Types”

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”

S

[index top [4287]]

scp

Section 7.1, “Backup and Recovery Types”

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

sed

Section 3.3.4.7, “Pattern Matching”

SELECT

Section 17.2.5, “MySQL Cluster Example with Tables and Data”

service

Section 2.5.2, “Installing MySQL on Linux using Native Package Manager”

Service Control Manager

Section 2.3, “Installing MySQL on Microsoft Windows”
Section 2.3.7.7, “Starting MySQL as a Windows Service”

Services

Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”
Section 2.3.7.7, “Starting MySQL as a Windows Service”

setenv

Section 4.2.4, “Setting Environment Variables”

sh

Section C.5.2.18, “`'File' Not Found` and Similar Errors”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.4, “Setting Environment Variables”
Section 1.2, “Typographical and Syntax Conventions”

SHOW

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 17.3.1, “Quick Test Setup of MySQL Cluster”

sleep

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

ssh

Section 17.5.11.1, “MySQL Cluster Security and Networking Issues”

Start>Run>cmd.exe

Section 6.3.8.5, “Setting Up SSL Certificates and Keys for MySQL”

strings

Section 6.1.1, “Security Guidelines”

su root

Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”

sudo

Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”
Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”

swinstall

Section 2.8.2, “Installing MySQL on HP-UX using DEPOT”

System Preferences...

Section 2.4.4, “Installing and Using the MySQL Preference Pane”

T

[index top [4287]]

tar

Section 16.3.1.2, “Backing Up Raw Data from a Slave”
Section 7.1, “Backup and Recovery Types”
Section 2.1.2.2, “Choosing a Distribution Format”
Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 3.3, “Creating and Using a Database”
Section 2.8.1, “General Notes on Installing MySQL on HP-UX”
Section 1.7, “How to Report Bugs or Problems”
Section 2.10.2, “Installing MySQL from a Standard Source Distribution”
Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”
Section 2.10, “Installing MySQL from Source”
Section 2.4, “Installing MySQL on Mac OS X”
Section 2.7, “Installing MySQL on Solaris and OpenSolaris”
Section 2.14.1, “Installing Perl on Unix”
Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”

tcpdump

Section 6.1.1, “Security Guidelines”

tcsh

Section C.5.2.18, “`'File' Not Found` and Similar Errors”
Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”
Section 4.2.1, “Invoking MySQL Programs”

Section 4.2.4, “Setting Environment Variables”
Section 1.2, “Typographical and Syntax Conventions”

tee

Section 4.5.1.2, “mysql Commands”

telnet

Section 6.1.1, “Security Guidelines”

Terminal

Section 2.4, “Installing MySQL on Mac OS X”

Text in this style

Section 1.2, “Typographical and Syntax Conventions”

top

Section C.5.1, “How to Determine What Is Causing a Problem”

U

[index top [4287]]

ulimit

Section C.5.2.18, “`'File' Not Found` and Similar Errors”
Section C.5.2.10, “`Packet too large`”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 8.11.4.2, “Enabling Large Page Support”
Section 5.1.3, “Server Command Options”

update-rc.d

Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”

useradd

Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”
Section 2.2, “Installing MySQL from Generic Binaries on Unix/Linux”
Section 2.5.1, “Installing MySQL from RPM Packages on Linux”
Section 2.7, “Installing MySQL on Solaris and OpenSolaris”

usermod

Section 2.5.1, “Installing MySQL from RPM Packages on Linux”

V

[index top [4287]]

vi

Section 4.5.1.2, “mysql Commands”
Section 17.2.3, “Initial Configuration of MySQL Cluster”
Section 3.3.4.7, “Pattern Matching”

W

[index top [4287]]

WinDbg

Section 23.4.1.3, “Using `pdb` to create a Windows crashdump”

winMd5Sum

Section 2.1.4.1, “Verifying the MD5 Checksum”

WinZip

Section 16.3.1.2, “Backing Up Raw Data from a Slave”
Section 2.10.2, “Installing MySQL from a Standard Source Distribution”
Section 2.10, “Installing MySQL from Source”

WordPad

Section 13.2.6, “`LOAD DATA INFILE` Syntax”

Y

[index top [4287]]

yacc

Section 2.10.5, “Dealing with Problems Compiling MySQL”
Section 9.3, “Reserved Words”

yum

Section 2.5.2, “Installing MySQL on Linux using Native Package Manager”

Z

[index top [4287]]

zip

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 1.7, “How to Report Bugs or Problems”

zsh

Section 4.2.4, “Setting Environment Variables”

Function Index

[Symbols](#) | [A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#)

Symbols

[\[index top \[4313\]\]](#)

[%](#)

[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)

A

[\[index top \[4313\]\]](#)

ABS ()

[Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”](#)

[Section 23.3, “Adding New Functions to MySQL”](#)

[Section 12.6.2, “Mathematical Functions”](#)

[Section 18.5.3, “Partitioning Limitations Relating to Functions”](#)

ACOS ()

[Section 12.6.2, “Mathematical Functions”](#)

ADDDATE ()

[Section 12.7, “Date and Time Functions”](#)

addslashes ()

[Section 6.1.7, “Client Programming Security Guidelines”](#)

ADDTIME ()

[Section 12.7, “Date and Time Functions”](#)

AES_DECRYPT ()

[Section 12.13, “Encryption and Compression Functions”](#)

AES_ENCRYPT ()

[Section 12.13, “Encryption and Compression Functions”](#)

[Section 12.2, “Type Conversion in Expression Evaluation”](#)

Area ()

[Section 12.17.5.2, “Geometry Property Functions”](#)

[Section 12.17.5.2.6, “MultiPolygon Functions”](#)

[Section 12.17.5.2.5, “Polygon Functions”](#)

AsBinary ()

[Section 12.17.4.5, “Fetching Spatial Data”](#)

[Section 12.17.5.1, “Geometry Format Conversion Functions”](#)

ASCII ()

[Section 13.8.3, “HELP Syntax”](#)

[Section 12.5, “String Functions”](#)

ASIN ()

[Section 12.6.2, “Mathematical Functions”](#)

AsText ()

[Section 12.17.4.5, “Fetching Spatial Data”](#)

[Section 12.17.5.1, “Geometry Format Conversion Functions”](#)

AsWKB ()

[Section 12.17.5.1, “Geometry Format Conversion Functions”](#)

AsWKT ()

[Section 12.17.5.1, “Geometry Format Conversion Functions”](#)

ATAN ()

[Section 12.6.2, “Mathematical Functions”](#)

ATAN2 ()

[Section 12.6.2, “Mathematical Functions”](#)

AVG ()

[Section 12.16.1, “GROUP BY \(Aggregate\) Functions”](#)

[Section 11.1.2, “Date and Time Type Overview”](#)

[Section 8.13.10.1, “Loose Index Scan”](#)

[Section 11.4.4, “The ENUM Type”](#)

[Section 11.4.5, “The SET Type”](#)

B

[\[index top \[4313\]\]](#)

BdMPolyFromText ()

[Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”](#)

BdMPolyFromWKB ()

[Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”](#)

BdPolyFromText ()

[Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”](#)

BdPolyFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

BENCHMARK()

Section 8.9.3.1, “How the Query Cache Operates”
Section 12.14, “Information Functions”
Section 8.12.1, “Measuring the Speed of Expressions and Functions”
Section 13.2.10.10, “Optimizing Subqueries”
Section 13.2.10.8, “Subqueries in the `FROM` Clause”

BIN()

Section 9.1.6, “Bit-Field Literals”
Section 12.5, “String Functions”

BIT_AND()

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”

BIT_COUNT()

Section 12.12, “Bit Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”

BIT_LENGTH()

Section 12.5, “String Functions”

BIT_OR()

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”

BIT_XOR()

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”

Boundary()

Section 12.17.5.2.1, “General Geometry Functions”

Buffer()

Section 12.17.5.3.2, “Spatial Operators”

C

[index top [4313]]

CAST()

Section 10.1.9.2, “`CONVERT()` and `CAST()`”
Section 9.1.6, “Bit-Field Literals”
Section 12.10, “Cast Functions and Operators”

Section 12.3.2, “Comparison Functions and Operators”
Section 11.3.7, “Conversion Between Date and Time Types”

Section 12.7, “Date and Time Functions”

Section 9.1.4, “Hexadecimal Literals”

Section 1.8.5, “MySQL Differences from Standard SQL”

Section 10.1.9.1, “Result Strings”

Section 10.1.7.7, “The `BINARY` Operator”

Section 12.2, “Type Conversion in Expression Evaluation”

Section 9.4, “User-Defined Variables”

CEIL()

Section 12.6.2, “Mathematical Functions”

CEILING()

Section 18.2.4.1, “`LINEAR HASH` Partitioning”
Section 12.6.2, “Mathematical Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

Centroid()

Section 12.17.5.2.6, “MultiPolygon Functions”

CHAR()

Section 12.10, “Cast Functions and Operators”
Section 12.13, “Encryption and Compression Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 12.5, “String Functions”
Section 12.2, “Type Conversion in Expression Evaluation”

CHAR_LENGTH()

Section 12.5, “String Functions”

CHARACTER_LENGTH()

Section 12.5, “String Functions”

CHARSET()

Section 12.14, “Information Functions”
Section 10.1.9.1, “Result Strings”
Section 12.2, “Type Conversion in Expression Evaluation”

COALESCE()

Section 13.2.9.2, “`JOIN` Syntax”
Section 12.3.2, “Comparison Functions and Operators”

COERCIBILITY()

Section 10.1.7.5, “Collation of Expressions”
Section 12.14, “Information Functions”

COLLATION ()

Section C.5.5.1, “Case Sensitivity in String Searches”
Section 12.14, “Information Functions”
Section 10.1.9.1, “Result Strings”
Section 12.2, “Type Conversion in Expression Evaluation”

COMPRESS ()

Section 12.13, “Encryption and Compression Functions”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 5.1.4, “Server System Variables”
Section 12.2, “Type Conversion in Expression Evaluation”

CONCAT ()

Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 12.16.1, “GROUP BY (Aggregate) Functions”
Section 13.7.5.14, “SHOW CREATE VIEW Syntax”
Section 12.10, “Cast Functions and Operators”
Section 10.1.7.5, “Collation of Expressions”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 10.1.9.1, “Result Strings”
Section 5.1.7, “Server SQL Modes”
Section 12.5, “String Functions”
Section 10.1.8, “String Repertoire”
Section 20.27, “The INFORMATION_SCHEMA VIEWS Table”
Section 12.2, “Type Conversion in Expression Evaluation”
Section 12.11, “XML Functions”

CONCAT_WS ()

Section 12.16.1, “GROUP BY (Aggregate) Functions”
Section 12.5, “String Functions”

CONNECTION_ID ()

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 12.14, “Information Functions”
Section 21.7.5.2, “The threads Table”
Section 6.3.10.2, “The Audit Log File”

Contains ()

Section 12.17.5.4.2, “Functions That Test Spatial Relationships Between Geometries”

CONV ()

Section 12.6.2, “Mathematical Functions”
Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

CONVERT ()

Section 10.1.9.2, “CONVERT () and CAST ()”
Section 12.10, “Cast Functions and Operators”
Section 10.1.3.5, “Character String Literal Character Set and Collation”
Section 12.3.2, “Comparison Functions and Operators”

CONVERT_TZ ()

Section 12.7, “Date and Time Functions”
Section 8.9.3.1, “How the Query Cache Operates”
Section 16.4.1.30, “Replication and Time Zones”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”

ConvexHull ()

Section 12.17.5.3.2, “Spatial Operators”

COS ()

Section 12.6.2, “Mathematical Functions”

COT ()

Section 12.6.2, “Mathematical Functions”

COUNT ()

Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 8.8.2, “EXPLAIN Output Format”
Section 12.16.1, “GROUP BY (Aggregate) Functions”
Section 3.3.4.8, “Counting Rows”
Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”
Section 8.13.10.1, “Loose Index Scan”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 18.1, “Overview of Partitioning in MySQL”
Section C.5.5.3, “Problems with NULL Values”
Section 5.1.7, “Server SQL Modes”
Section 19.5.3, “Updatable and Insertable Views”
Section 19.5.2, “View Processing Algorithms”

CRC32 ()

Section 12.6.2, “Mathematical Functions”

Crosses ()

Section 12.17.5.4.2, “Functions That Test Spatial Relationships Between Geometries”

crypt ()

Section 12.13, “Encryption and Compression Functions”
Section 5.1.4, “Server System Variables”

CURDATE ()

Section 12.7, “Date and Time Functions”

Section 3.3.4.5, “Date Calculations”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”

CURRENT_DATE

Section 13.1.17, “CREATE TABLE Syntax”
Section 11.5, “Data Type Default Values”
Section 12.7, “Date and Time Functions”

CURRENT_DATE ()

Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”

CURRENT_TIME

Section 12.7, “Date and Time Functions”

CURRENT_TIME ()

Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”

CURRENT_TIMESTAMP

Section 13.1.11, “CREATE EVENT Syntax”
Section 13.1.17, “CREATE TABLE Syntax”
Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP”
Section 11.5, “Data Type Default Values”
Section 12.7, “Date and Time Functions”

CURRENT_TIMESTAMP ()

Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP”
Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”

CURRENT_USER

Section 13.1.11, “CREATE EVENT Syntax”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.19, “CREATE TRIGGER Syntax”
Section 13.1.20, “CREATE VIEW Syntax”
Section 19.6, “Access Control for Stored Programs and Views”
Section 12.14, “Information Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.13, “Replication and System Functions”

Section 16.4.1.6, “Replication of CURRENT_USER ()”
Section 6.2.3, “Specifying Account Names”

CURRENT_USER ()

Section 13.1.11, “CREATE EVENT Syntax”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.19, “CREATE TRIGGER Syntax”
Section 13.1.20, “CREATE VIEW Syntax”
Section 13.7.1.6, “SET PASSWORD Syntax”
Section 6.2.4, “Access Control, Stage 1: Connection Verification”
Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”
Section 12.14, “Information Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 6.3.7, “Proxy Users”
Section 16.4.1.13, “Replication and System Functions”
Section 16.4.1.6, “Replication of CURRENT_USER ()”
Section 6.2.3, “Specifying Account Names”
Section 6.3.11, “SQL-Based MySQL Account Activity Auditing”
Section 10.1.12, “UTF-8 for Metadata”
Section 23.2.4.9.1, “Writing the Server-Side Authentication Plugin”

CURTIME ()

Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 10.6, “MySQL Server Time Zone Support”

D

[index top [4313]]

DATABASE ()

Section 13.1.21, “DROP DATABASE Syntax”
Section 3.3.1, “Creating and Selecting a Database”
Section 3.4, “Getting Information About Databases and Tables”
Section 8.9.3.1, “How the Query Cache Operates”
Section 12.14, “Information Functions”
Section C.5.8, “Known Issues in MySQL”
Section 10.1.12, “UTF-8 for Metadata”

DATE ()

Section 12.7, “Date and Time Functions”

DATE_ADD ()

Section 13.1.11, “CREATE EVENT Syntax”

Section 12.6.1, “Arithmetic Operators”
Section 12.7, “Date and Time Functions”
Section 11.3, “Date and Time Types”
Section 3.3.4.5, “Date Calculations”
Section 9.5, “Expression Syntax”

DATE_FORMAT ()

Section 22.8.14, “C API Prepared Statement Problems”
Section 12.7, “Date and Time Functions”
Section 10.7, “MySQL Server Locale Support”
Section 5.1.4, “Server System Variables”

DATE_SUB ()

Section 12.7, “Date and Time Functions”
Section 11.3, “Date and Time Types”

DATEDIFF ()

Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

DAY ()

Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

DAYNAME ()

Section 12.7, “Date and Time Functions”
Section 10.7, “MySQL Server Locale Support”
Section 5.1.4, “Server System Variables”

DAYOFMONTH ()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

DAYOFWEEK ()

Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

DAYOFYEAR ()

Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”
Section 18.2, “Partitioning Types”

DECODE ()

Section 12.13, “Encryption and Compression Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”

DEFAULT ()

Section 13.2.5, “`INSERT` Syntax”
Section 13.2.8, “`REPLACE` Syntax”
Section 11.5, “Data Type Default Values”
Section 12.15, “Miscellaneous Functions”

DEGREES ()

Section 12.6.2, “Mathematical Functions”

DES_DECRYPT ()

Section 12.13, “Encryption and Compression Functions”
Section 5.1.3, “Server Command Options”

DES_ENCRYPT ()

Section 12.13, “Encryption and Compression Functions”
Section 5.1.3, “Server Command Options”

Difference ()

Section 12.17.5.3.2, “Spatial Operators”

Dimension ()

Section 12.17.5.2.1, “General Geometry Functions”

Disjoint ()

Section 12.17.5.4.2, “Functions That Test Spatial Relationships Between Geometries”

E

[[index top \[4313\]](#)]

ELT ()

Section C.5.8, “Known Issues in MySQL”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

ENCODE ()

Section 12.13, “Encryption and Compression Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”

ENCRYPT ()

Section 12.13, “Encryption and Compression Functions”
Section 8.9.3.1, “How the Query Cache Operates”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section E.7, “Restrictions on Character Sets”
Section 5.1.4, “Server System Variables”
Section 6.3.1, “User Names and Passwords”

EndPoint ()

Section 12.17.5.2.3, “`LineString` Functions”

Section 12.17.5.2.4, “MultiLineString Functions”
Section 12.17.5.3.1, “Geometry Functions That Produce New Geometries”

Envelope ()

Section 12.17.5.2.1, “General Geometry Functions”
Section 12.17.5.3.1, “Geometry Functions That Produce New Geometries”

Equals ()

Section 12.17.5.4.2, “Functions That Test Spatial Relationships Between Geometries”

EXP ()

Section 12.6.2, “Mathematical Functions”

EXPORT_SET ()

Section 12.5, “String Functions”

expr IN ()

Section 12.3.2, “Comparison Functions and Operators”

expr NOT IN ()

Section 12.3.2, “Comparison Functions and Operators”

ExteriorRing ()

Section 12.17.5.2.5, “Polygon Functions”
Section 12.17.5.3.1, “Geometry Functions That Produce New Geometries”

EXTRACT ()

Section 12.10, “Cast Functions and Operators”
Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

ExtractValue ()

Section 12.11, “XML Functions”

F

[[index top \[4313\]](#)]

FIELD ()

Section 12.5, “String Functions”

FIND_IN_SET ()

Section 12.5, “String Functions”
Section 11.4.5, “The SET Type”

FLOOR ()

Section 12.6.1, “Arithmetic Operators”
Section 12.6.2, “Mathematical Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

FORMAT ()

Section 12.6.2, “Mathematical Functions”
Section 12.15, “Miscellaneous Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 10.7, “MySQL Server Locale Support”
Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

FOUND_ROWS ()

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 12.14, “Information Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.13, “Replication and System Functions”

FROM_DAYS ()

Section 12.7, “Date and Time Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”

FROM_UNIXTIME ()

Section 6.3.10.3, “Audit Log Plugin Logging Control”
Section 12.7, “Date and Time Functions”
Section 16.4.1.30, “Replication and Time Zones”

G

[[index top \[4313\]](#)]

GeomCollFromText ()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

GeomCollFromWKB ()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

GeometryCollection ()

Section 12.17.4.2.3, “Creating Geometry Values Using MySQL-Specific Functions”

GeometryCollectionFromText ()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

GeometryCollectionFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

GeometryFromText()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

GeometryFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

GeometryN()

Section 12.17.5.2.7, “GeometryCollection Functions”
Section 12.17.5.3.1, “Geometry Functions That Produce New Geometries”

GeometryType()

Section 12.17.5.2.1, “General Geometry Functions”

GeomFromText()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

Section 12.17.5.1, “Geometry Format Conversion Functions”

Section 12.17.4.4, “Populating Spatial Columns”

GeomFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

Section 12.17.5.1, “Geometry Format Conversion Functions”

GET_FORMAT()

Section 12.7, “Date and Time Functions”

Section 10.7, “MySQL Server Locale Support”

GET_LOCK()

Section 13.1.11, “CREATE EVENT Syntax”

Section 13.7.6.4, “KILL Syntax”

Section 22.8.3.3, “mysql_change_user()”

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”

Section 22.8.12, “Controlling Automatic Reconnection Behavior”

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

Section 19.4.1, “Event Scheduler Overview”

Section 8.12.5.2, “General Thread States”

Section 8.9.3.1, “How the Query Cache Operates”

Section 8.10.1, “Internal Locking Methods”

Section 12.15, “Miscellaneous Functions”

Section 16.4.1.13, “Replication and System Functions”

Section 13.3.5.3, “Table-Locking Restrictions and Conditions”

gethostbyaddr()

Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”

gethostbyaddr_r()

Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”

gethostbyname()

Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”

gethostbyname_r()

Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”

getrusage()

Section 17.5.10.18, “The ndbinfo threadstat Table”

gettimeofday()

Section 17.5.10.18, “The ndbinfo threadstat Table”

GLength()

Section 12.17.5.2.3, “LineString Functions”

Section 12.17.5.2.4, “MultiLineString Functions”

Section 12.17.7, “MySQL Conformance and Compatibility”

GREATEST()

Section 12.3.2, “Comparison Functions and Operators”

Section 10.1.9.1, “Result Strings”

GROUP_CONCAT()

Section 12.16.1, “GROUP BY (Aggregate) Functions”

Section C.5.8, “Known Issues in MySQL”

Section 1.8.4, “MySQL Extensions to Standard SQL”

Section 5.1.4, “Server System Variables”

H

[index top [4313]]

HEX()

Section 10.1.3.5, “Character String Literal Character Set and Collation”

Section 9.1.4, “Hexadecimal Literals”

Section 12.6.2, “Mathematical Functions”

Section 10.1.9.1, “Result Strings”

Section 12.5, “String Functions”

HOUR ()

Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

I

[index top [4313]]

IF ()

Section 13.6.5.2, “IF Syntax”
Section 12.4, “Control Flow Functions”
Section C.5.8, “Known Issues in MySQL”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 10.1.9.1, “Result Strings”

IFNULL ()

Section 12.4, “Control Flow Functions”
Section C.5.5.3, “Problems with NULL Values”

IN

Section 12.3.1, “Operator Precedence”

IN ()

Section 8.8.2, “EXPLAIN Output Format”
Section E.4, “Restrictions on Subqueries”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”
Section 12.2, “Type Conversion in Expression Evaluation”

INET_ATON ()

Section 12.15, “Miscellaneous Functions”

INET_NTOA ()

Section 12.15, “Miscellaneous Functions”

INSERT ()

Section 12.5, “String Functions”

INSTR ()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

InteriorRingN ()

Section 12.17.5.2.5, “Polygon Functions”
Section 12.17.5.3.1, “Geometry Functions That Produce New Geometries”

Intersection ()

Section 12.17.5.3.2, “Spatial Operators”

Intersects ()

Section 12.17.5.4.2, “Functions That Test Spatial Relationships Between Geometries”

INTERVAL ()

Section 12.3.2, “Comparison Functions and Operators”

IS_FREE_LOCK ()

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.15, “Miscellaneous Functions”
Section 16.4.1.13, “Replication and System Functions”

IS_USED_LOCK ()

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.15, “Miscellaneous Functions”
Section 16.4.1.13, “Replication and System Functions”

IsClosed ()

Section 12.17.5.2.4, “MultiLineString Functions”

IsEmpty ()

Section 12.17.5.2.1, “General Geometry Functions”

ISNULL ()

Section 12.3.2, “Comparison Functions and Operators”

IsRing ()

Section 12.17.5.2.3, “LineString Functions”

IsSimple ()

Section 12.17.5.2.1, “General Geometry Functions”

L

[index top [4313]]

LAST_DAY ()

Section 12.7, “Date and Time Functions”

LAST_INSERT_ID ()

Section 4.6.7.1, “mysqlbinlog Hex Dump Format”
Section 13.1.17, “CREATE TABLE Syntax”
Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”

Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 13.2.5, “`INSERT` Syntax”
Section 22.8.3.37, “`mysql_insert_id()`”
Section 22.8.7.16, “`mysql_stmt_insert_id()`”
Section 12.3.2, “Comparison Functions and Operators”
Section 22.8.12, “Controlling Automatic Reconnection Behavior”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 22.8.11.3, “How to Get the Unique ID for the Last Inserted Row”
Section 12.14, “Information Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 16.4.1.1, “Replication and `AUTO_INCREMENT`”
Section 5.1.4, “Server System Variables”
Section 19.2.4, “Stored Procedures, Functions, Triggers, and `LAST_INSERT_ID()`”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 1.8.5.3, “Transaction and Atomic Operation Differences”
Section 16.4.4, “Troubleshooting Replication”
Section 19.5.3, “Updatable and Insertable Views”
Section 3.6.9, “Using `AUTO_INCREMENT`”

LCASE ()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

LEAST ()

Section 12.3.2, “Comparison Functions and Operators”
Section 10.1.9.1, “Result Strings”

LEFT ()

Section 12.10, “Cast Functions and Operators”
Section 12.5, “String Functions”

LENGTH ()

Section 12.5, “String Functions”

Length ()

Section 12.17.5.2.3, “`LineString` Functions”
Section 12.17.5.2.4, “`MultiLineString` Functions”
Section 12.17.7, “MySQL Conformance and Compatibility”

LineFromText ()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”
Section 12.17.5.1, “Geometry Format Conversion Functions”

LineFromWKB ()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”
Section 12.17.5.1, “Geometry Format Conversion Functions”

LineString ()

Section 12.17.4.2.3, “Creating Geometry Values Using MySQL-Specific Functions”

LineStringFromText ()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

LineStringFromWKB ()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

LN ()

Section 12.6.2, “Mathematical Functions”

LOAD_FILE ()

Section 13.2.7, “`LOAD XML` Syntax”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.13, “Replication and System Functions”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 12.5, “String Functions”

LOCALTIME

Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP`”
Section 12.7, “Date and Time Functions”

LOCALTIME ()

Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP`”
Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

LOCALTIMESTAMP

Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP`”
Section 12.7, “Date and Time Functions”

LOCALTIMESTAMP ()

Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP`”

Section 12.7, “Date and Time Functions”

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

LOCATE ()

Section 12.5, “String Functions”

LOG ()

Section 18.2.4.1, “`LINEAR HASH` Partitioning”

Section 12.6.2, “Mathematical Functions”

LOG10 ()

Section 12.6.2, “Mathematical Functions”

LOG2 ()

Section 12.6.2, “Mathematical Functions”

LOWER ()

Section 12.10, “Cast Functions and Operators”

Section 10.1.7.9, “Collation and `INFORMATION_SCHEMA` Searches”

Section 10.1.9.1, “Result Strings”

Section 12.5, “String Functions”

LPAD ()

Section 12.5, “String Functions”

LTRIM ()

Section 10.1.9.1, “Result Strings”

Section 12.5, “String Functions”

M

[index top [4313]]

MAKE_SET ()

Section 12.5, “String Functions”

MAKEDATE ()

Section 12.7, “Date and Time Functions”

MAKETIME ()

Section 12.7, “Date and Time Functions”

MASTER_POS_WAIT ()

Section 13.4.2.2, “`MASTER_POS_WAIT ()` Syntax”

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

Section 8.9.3.1, “How the Query Cache Operates”

Section 12.15, “Miscellaneous Functions”

MATCH

Section 9.5, “Expression Syntax”

MATCH ()

Section 12.9, “Full-Text Search Functions”

MATCH ()

Section 12.9.2, “Boolean Full-Text Searches”

Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”

Section 12.9.5, “Full-Text Restrictions”

Section 12.9, “Full-Text Search Functions”

Section 12.9.1, “Natural Language Full-Text Searches”

MAX ()

Section 8.8.2, “`EXPLAIN` Output Format”

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”

Section 8.3.1, “How MySQL Uses Indexes”

Section C.5.8, “Known Issues in MySQL”

Section 8.13.10.1, “Loose Index Scan”

Section 12.16.3, “MySQL Extensions to `GROUP BY`”

Section 11.1.1, “Numeric Type Overview”

Section 13.2.10.10, “Optimizing Subqueries”

Section 5.1.7, “Server SQL Modes”

Section 20.30.2, “The `INFORMATION_SCHEMA` `TP_THREAD_GROUP_STATE` Table”

Section 11.3.8, “Two-Digit Years in Dates”

Section 19.5.3, “Updatable and Insertable Views”

Section 3.6.9, “Using `AUTO_INCREMENT`”

Section 19.5.2, “View Processing Algorithms”

MBRContains ()

Section 12.17.5.4.1, “Relations on Geometry Minimal Bounding Rectangles (MBRs)”

Section 12.17.6.2, “Using a Spatial Index”

MBRDisjoint ()

Section 12.17.5.4.1, “Relations on Geometry Minimal Bounding Rectangles (MBRs)”

MBREqual ()

Section 12.17.5.4.1, “Relations on Geometry Minimal Bounding Rectangles (MBRs)”

MBRIntersects ()

Section 12.17.5.4.1, “Relations on Geometry Minimal Bounding Rectangles (MBRs)”

MBROverlaps()

Section 12.17.5.4.1, “Relations on Geometry Minimal Bounding Rectangles (MBRs)”

MBRTouches()

Section 12.17.5.4.1, “Relations on Geometry Minimal Bounding Rectangles (MBRs)”

MBRWithin()

Section 12.17.5.4.1, “Relations on Geometry Minimal Bounding Rectangles (MBRs)”

Section 12.17.6.2, “Using a Spatial Index”

MD5()

Section 18.2.5, “**KEY** Partitioning”

Section 12.13, “Encryption and Compression Functions”

Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”

Section 1.8.4, “MySQL Extensions to Standard SQL”

Section 9.2, “Schema Object Names”

Section 6.1.1, “Security Guidelines”

Section 12.2, “Type Conversion in Expression Evaluation”

MICROSECOND()

Section 12.7, “Date and Time Functions”

Section 18.5.3, “Partitioning Limitations Relating to Functions”

MID()

Section 10.1.9.1, “Result Strings”

Section 12.5, “String Functions”

MIN()

Section 8.8.2, “**EXPLAIN** Output Format”

Section 12.16.1, “**GROUP BY** (Aggregate) Functions”

Section 22.8.14, “C API Prepared Statement Problems”

Section 8.2.1.2, “How MySQL Optimizes **WHERE** Clauses”

Section 8.3.1, “How MySQL Uses Indexes”

Section C.5.8, “Known Issues in MySQL”

Section 8.13.10.1, “Loose Index Scan”

Section 12.16.3, “MySQL Extensions to **GROUP BY**”

Section 11.1.1, “Numeric Type Overview”

Section 13.2.10.10, “Optimizing Subqueries”

Section C.5.5.3, “Problems with **NULL** Values”

Section 11.3.8, “Two-Digit Years in Dates”

Section 19.5.3, “Updatable and Insertable Views”

Section 19.5.2, “View Processing Algorithms”

MINUTE()

Section 12.7, “Date and Time Functions”

Section 18.5.3, “Partitioning Limitations Relating to Functions”

MLineFromText()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

MLineFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

MOD()

Section 12.6.1, “Arithmetic Operators”

Section 3.3.4.5, “Date Calculations”

Section 12.6.2, “Mathematical Functions”

Section 1.8.4, “MySQL Extensions to Standard SQL”

Section 18.5.3, “Partitioning Limitations Relating to Functions”

Section 5.1.7, “Server SQL Modes”

MONTH()

Section 12.7, “Date and Time Functions”

Section 3.3.4.5, “Date Calculations”

Section 18.5.3, “Partitioning Limitations Relating to Functions”

Section 18.2, “Partitioning Types”

MONTHNAME()

Section 12.7, “Date and Time Functions”

Section 10.7, “MySQL Server Locale Support”

Section 5.1.4, “Server System Variables”

MPointFromText()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

MPointFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

MPolyFromText()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

MPolyFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

MultiLineString()

Section 12.17.4.2.3, “Creating Geometry Values Using MySQL-Specific Functions”

MultiLineStringFromText()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

MultiLineStringFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

MultiPoint()

Section 12.17.4.2.3, “Creating Geometry Values Using MySQL-Specific Functions”

MultiPointFromText()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

MultiPointFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

MultiPolygon()

Section 12.17.4.2.3, “Creating Geometry Values Using MySQL-Specific Functions”

MultiPolygonFromText()

Section 12.17.4.2.1, “Creating Geometry Values Using WKT Functions”

MultiPolygonFromWKB()

Section 12.17.4.2.2, “Creating Geometry Values Using WKB Functions”

my_open()

Section 5.1.6, “Server Status Variables”

N

[index top [4313]]

NAME_CONST()

Section 19.7, “Binary Logging of Stored Programs”
Section 12.15, “Miscellaneous Functions”

NOW()

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.17, “CREATE TABLE Syntax”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP”

Section 11.5, “Data Type Default Values”
Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 10.6, “MySQL Server Time Zone Support”
Section 16.4.1.13, “Replication and System Functions”
Section 16.4.1.30, “Replication and Time Zones”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 11.3.3, “The YEAR Type”
Section 10.6.2, “Time Zone Leap Second Support”

NULLIF()

Section 12.4, “Control Flow Functions”

NumGeometries()

Section 12.17.5.2.7, “GeometryCollection Functions”

NumInteriorRings()

Section 12.17.5.2.5, “Polygon Functions”

NumPoints()

Section 12.17.5.2.3, “LineString Functions”

O

[index top [4313]]

OCT()

Section 12.5, “String Functions”

OCTET_LENGTH()

Section 12.5, “String Functions”

OLD_PASSWORD()

Section C.5.2.4, “Client does not support authentication protocol”
Section 13.7.1.6, “SET PASSWORD Syntax”
Section 12.13, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 5.1.4, “Server System Variables”

ORD()

Section 12.5, “String Functions”

Overlaps()

Section 12.17.5.4.2, “Functions That Test Spatial Relationships Between Geometries”

P

[\[index top \[4313\]\]](#)

PASSWORD ()

Section 13.7.1.1, “[CREATE USER Syntax](#)”
Section C.5.2.15, “[Ignoring user](#)”
Section 18.2.5, “[KEY Partitioning](#)”
Section 13.7.1.6, “[SET PASSWORD Syntax](#)”
Section 6.2.4, “[Access Control, Stage 1: Connection Verification](#)”
Section 6.3.2, “[Adding User Accounts](#)”
Section 6.3.5, “[Assigning Account Passwords](#)”
Section 6.2.7, “[Causes of Access-Denied Errors](#)”
Section 12.13, “[Encryption and Compression Functions](#)”
Section 6.1.2.5, “[Implications of Password Hashing Changes in MySQL 4.1 for Application Programs](#)”
Section 1.8.4, “[MySQL Extensions to Standard SQL](#)”
Section 6.1.2.4, “[Password Hashing in MySQL](#)”
Section 6.1.2.3, “[Passwords and Logging](#)”
Section 2.11.2, “[Securing the Initial MySQL Accounts](#)”
Section 5.1.4, “[Server System Variables](#)”
Section 12.2, “[Type Conversion in Expression Evaluation](#)”
Section 6.3.1, “[User Names and Passwords](#)”

PERIOD_ADD ()

Section 12.7, “[Date and Time Functions](#)”
Section 1.8.4, “[MySQL Extensions to Standard SQL](#)”

PERIOD_DIFF ()

Section 12.7, “[Date and Time Functions](#)”
Section 1.8.4, “[MySQL Extensions to Standard SQL](#)”

PI ()

Section 9.2.4, “[Function Name Parsing and Resolution](#)”
Section 12.6.2, “[Mathematical Functions](#)”

POINT ()

Section 12.17.3.1, “[Well-Known Text \(WKT\) Format](#)”

Point ()

Section 12.17.4.2.3, “[Creating Geometry Values Using MySQL-Specific Functions](#)”

PointFromText ()

Section 12.17.4.2.1, “[Creating Geometry Values Using WKT Functions](#)”
Section 12.17.5.1, “[Geometry Format Conversion Functions](#)”

PointFromWKB ()

Section 12.17.4.2.2, “[Creating Geometry Values Using WKB Functions](#)”
Section 12.17.5.1, “[Geometry Format Conversion Functions](#)”

PointN ()

Section 12.17.5.2.3, “[LineString Functions](#)”
Section 12.17.5.3.1, “[Geometry Functions That Produce New Geometries](#)”

PolyFromText ()

Section 12.17.4.2.1, “[Creating Geometry Values Using WKT Functions](#)”

PolyFromWKB ()

Section 12.17.4.2.2, “[Creating Geometry Values Using WKB Functions](#)”

Polygon ()

Section 12.17.4.2.3, “[Creating Geometry Values Using MySQL-Specific Functions](#)”

PolygonFromText ()

Section 12.17.4.2.1, “[Creating Geometry Values Using WKT Functions](#)”

PolygonFromWKB ()

Section 12.17.4.2.2, “[Creating Geometry Values Using WKB Functions](#)”

POSITION ()

Section 12.5, “[String Functions](#)”

POW ()

Section 18.2.4, “[HASH Partitioning](#)”
Section 12.6.2, “[Mathematical Functions](#)”

POWER ()

Section 18.2.4.1, “[LINEAR HASH Partitioning](#)”
Section 12.6.2, “[Mathematical Functions](#)”

Q

[\[index top \[4313\]\]](#)

QUARTER ()

Section 12.7, “[Date and Time Functions](#)”
Section 18.5.3, “[Partitioning Limitations Relating to Functions](#)”

QUOTE ()

Section 22.8.3.53, “mysql_real_escape_string()”
Section 12.5, “String Functions”
Section 9.1.1, “String Literals”

R

[[index top \[4313\]](#)]

RADIANS ()

Section 12.6.2, “Mathematical Functions”

RAND ()

Section 4.6.7.1, “mysqlbinlog Hex Dump Format”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 12.6.2, “Mathematical Functions”
Section 16.4.1.13, “Replication and System Functions”
Section 5.1.4, “Server System Variables”

RELEASE_LOCK ()

Section 13.2.3, “DO Syntax”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 8.10.1, “Internal Locking Methods”
Section 12.15, “Miscellaneous Functions”
Section 16.4.1.13, “Replication and System Functions”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”

REPEAT ()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

REPLACE ()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

REVERSE ()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

RIGHT ()

Section 3.3.4.5, “Date Calculations”

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

ROUND ()

Section 12.6.2, “Mathematical Functions”
Section 12.18, “Precision Math”
Section 12.18.5, “Precision Math Examples”
Section 12.18.4, “Rounding Behavior”

ROW_COUNT ()

Section 13.2.1, “CALL Syntax”
Section 13.2.2, “DELETE Syntax”
Section 13.2.5, “INSERT Syntax”
Section 22.8.3.1, “mysql_affected_rows()”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.14, “Information Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.13, “Replication and System Functions”

RPAD ()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

RTRIM ()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

S

[[index top \[4313\]](#)]

SCHEMA ()

Section 12.14, “Information Functions”

SEC_TO_TIME ()

Section 12.7, “Date and Time Functions”

SECOND ()

Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

SESSION_USER ()

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.14, “Information Functions”
Section 10.1.12, “UTF-8 for Metadata”

setrlimit ()

Section 5.1.3, “Server Command Options”

SHA ()

Section 12.13, “Encryption and Compression Functions”

SHA1 ()

Section 12.13, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 6.1.1, “Security Guidelines”

SHA2 ()

Section 12.13, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 6.1.1, “Security Guidelines”

SIGN ()

Section 12.6.2, “Mathematical Functions”

SIN ()

Section 12.6.2, “Mathematical Functions”
Section 23.3.2.3, “UDF Argument Processing”

SLEEP ()

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.12.5.2, “General Thread States”
Section 8.9.3.1, “How the Query Cache Operates”
Section 12.15, “Miscellaneous Functions”
Section 16.4.1, “Replication Features and Issues”
Section 20.30.3, “The `INFORMATION_SCHEMA` `TP_THREAD_GROUP_STATS` Table”

SOUNDEX ()

Section 23.3, “Adding New Functions to MySQL”
Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

SPACE ()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

SQRT ()

Section 12.6.2, “Mathematical Functions”

SRID ()

Section 12.17.5.2.1, “General Geometry Functions”

StartPoint ()

Section 12.17.5.2.3, “`LineString` Functions”

Section 12.17.5.2.4, “`MultiLineString` Functions”
Section 12.17.5.3.1, “Geometry Functions That Produce New Geometries”

STD ()

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”

STDDEV ()

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”

STDDEV_POP ()

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”

STDDEV_SAMP ()

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”

STR_TO_DATE ()

Section 12.7, “Date and Time Functions”
Section 10.7, “MySQL Server Locale Support”

STRCMP ()

Section C.5.5.2, “Problems Using `DATE` Columns”
Section 12.5.1, “String Comparison Functions”

SUBDATE ()

Section 12.7, “Date and Time Functions”

SUBSTR ()

Section 12.5, “String Functions”

SUBSTRING ()

Section 10.1.9.1, “Result Strings”
Section 6.3.11, “SQL-Based MySQL Account Activity Auditing”
Section 12.5, “String Functions”

SUBSTRING_INDEX ()

Section 12.5, “String Functions”

SUBTIME ()

Section 12.7, “Date and Time Functions”

SUM ()

Section 12.16.1, “`GROUP BY` (Aggregate) Functions”
Section 23.3.2, “Adding a New User-Defined Function”
Section 11.1.2, “Date and Time Type Overview”
Section 8.13.10.1, “Loose Index Scan”
Section 18.1, “Overview of Partitioning in MySQL”
Section C.5.5.3, “Problems with `NULL` Values”

Section 11.4.4, “The `ENUM` Type”
Section 11.4.5, “The `SET` Type”
Section 19.5.3, “Updatable and Insertable Views”
Section 19.5.2, “View Processing Algorithms”

SymDifference()

Section 12.17.5.3.2, “Spatial Operators”

SYSDATE()

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 16.4.1.13, “Replication and System Functions”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

SYSTEM_USER()

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.14, “Information Functions”
Section 10.1.12, “UTF-8 for Metadata”

T

[[index top \[4313\]](#)]

TAN()

Section 12.6.2, “Mathematical Functions”

thr_setconcurrency()

Section 5.1.4, “Server System Variables”

TIME()

Section 12.7, “Date and Time Functions”

TIME_FORMAT()

Section 12.7, “Date and Time Functions”

TIME_TO_SEC()

Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”

TIMEDIFF()

Section 12.7, “Date and Time Functions”

TIMESTAMP()

Section 12.7, “Date and Time Functions”

TIMESTAMPADD()

Section 12.7, “Date and Time Functions”
Section 1.4, “What Is New in MySQL 5.5”

TIMESTAMPDIFF()

Section 12.7, “Date and Time Functions”

TO_DAYS()

Section 18.2.4, “`HASH` Partitioning”
Section 12.7, “Date and Time Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 18.4, “Partition Pruning”
Section 18.5.3, “Partitioning Limitations Relating to Functions”
Section 18.2, “Partitioning Types”

TO_SECONDS()

Section 12.7, “Date and Time Functions”
Section 18.4, “Partition Pruning”
Section 18.5.3, “Partitioning Limitations Relating to Functions”
Section 18.2, “Partitioning Types”
Section 1.4, “What Is New in MySQL 5.5”

Touches()

Section 12.17.5.4.2, “Functions That Test Spatial Relationships Between Geometries”

TRIM()

Section 10.1.13, “Column Character Set Conversion”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

TRUNCATE()

Section 12.6.2, “Mathematical Functions”

U

[[index top \[4313\]](#)]

UCASE()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

UNCOMPRESS()

Section 12.13, “Encryption and Compression Functions”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 5.1.4, “Server System Variables”

UNCOMPRESSED_LENGTH()

Section 12.13, “Encryption and Compression Functions”

UNHEX ()

Section 12.13, “Encryption and Compression Functions”
Section 12.5, “String Functions”

Union ()

Section 12.17.5.3.2, “Spatial Operators”

UNIX_TIMESTAMP ()

Section 18.2.1, “**RANGE** Partitioning”
Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 18.5.3, “Partitioning Limitations Relating to Functions”
Section C.5.4.6, “Time Zone Problems”

UpdateXML ()

Section 12.11, “XML Functions”

UPPER ()

Section 12.10, “Cast Functions and Operators”
Section 10.1.7.9, “Collation and **INFORMATION_SCHEMA** Searches”
Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”
Section 10.1.8, “String Repertoire”

USER ()

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 10.1.7.5, “Collation of Expressions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”
Section 12.14, “Information Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 6.3.7, “Proxy Users”
Section 16.4.1.13, “Replication and System Functions”
Section 6.3.11, “SQL-Based MySQL Account Activity Auditing”
Section 10.1.12, “UTF-8 for Metadata”
Section 23.2.4.9.1, “Writing the Server-Side Authentication Plugin”

UTC_DATE

Section 12.7, “Date and Time Functions”

UTC_DATE ()

Section 12.7, “Date and Time Functions”

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

UTC_TIME

Section 12.7, “Date and Time Functions”

UTC_TIME ()

Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

UTC_TIMESTAMP

Section 12.7, “Date and Time Functions”

UTC_TIMESTAMP ()

Section 12.7, “Date and Time Functions”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 10.6, “MySQL Server Time Zone Support”

UUID ()

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.3.4, “Binary Log Options and Variables”
Section 19.7, “Binary Logging of Stored Programs”
Section 22.2.6, “Connector/Net Connection String Options Reference”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 12.15, “Miscellaneous Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.13, “Replication and System Functions”
Section 5.2.4.2, “Setting The Binary Log Format”

UUID_SHORT ()

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.9.3.1, “How the Query Cache Operates”
Section 12.15, “Miscellaneous Functions”

V

[index top [4313]]

VALUES ()

Section 13.2.5.3, “**INSERT ... ON DUPLICATE KEY UPDATE** Syntax”
Section 12.15, “Miscellaneous Functions”

VAR_POP ()

Section 12.16.1, “[GROUP BY \(Aggregate\) Functions](#)”

VAR_SAMP ()

Section 12.16.1, “[GROUP BY \(Aggregate\) Functions](#)”

VARIANCE ()

Section 12.16.1, “[GROUP BY \(Aggregate\) Functions](#)”

VERSION ()

Section 16.1.2.1, “[Advantages and Disadvantages of Statement-Based and Row-Based Replication](#)”

Section C.5.5.1, “[Case Sensitivity in String Searches](#)”

Section 10.1.7.5, “[Collation of Expressions](#)”

Section 12.14, “[Information Functions](#)”

Section 16.4.1.13, “[Replication and System Functions](#)”

Section 6.3.10.2, “[The Audit Log File](#)”

Section 10.1.12, “[UTF-8 for Metadata](#)”

W

[\[index top \[4313\]\]](#)

WEEK ()

Section 12.7, “[Date and Time Functions](#)”

Section 5.1.4, “[Server System Variables](#)”

WEEKDAY ()

Section 12.7, “[Date and Time Functions](#)”

Section 1.8.4, “[MySQL Extensions to Standard SQL](#)”

Section 18.5.3, “[Partitioning Limitations Relating to Functions](#)”

Section 18.2, “[Partitioning Types](#)”

WEEKOFYEAR ()

Section 12.7, “[Date and Time Functions](#)”

Within ()

Section 12.17.5.4.2, “[Functions That Test Spatial Relationships Between Geometries](#)”

X

[\[index top \[4313\]\]](#)

X ()

Section 12.17.5.2.2, “[Point Functions](#)”

Y

[\[index top \[4313\]\]](#)

Y ()

Section 12.17.5.2.2, “[Point Functions](#)”

YEAR ()

Section 18.2.4, “[HASH Partitioning](#)”

Section 18.2.1, “[RANGE Partitioning](#)”

Section 12.7, “[Date and Time Functions](#)”

Section 3.3.4.5, “[Date Calculations](#)”

Section 18.2.7, “[How MySQL Partitioning Handles NULL](#)”

Section 18.3.1, “[Management of RANGE and LIST Partitions](#)”

Section 18.4, “[Partition Pruning](#)”

Section 18.5.3, “[Partitioning Limitations Relating to Functions](#)”

Section 18.2, “[Partitioning Types](#)”

YEARWEEK ()

Section 12.7, “[Date and Time Functions](#)”

Section 18.5.3, “[Partitioning Limitations Relating to Functions](#)”

INFORMATION_SCHEMA Index

[C](#) | [E](#) | [F](#) | [G](#) | [I](#) | [K](#) | [N](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#)

C

[\[index top \[4331\]\]](#)

CHARACTER_SETS

Section 10.1.9.3, “[SHOW Statements and INFORMATION_SCHEMA](#)”

Section 20.1, “[The INFORMATION_SCHEMA CHARACTER_SETS Table](#)”

COLLATION_CHARACTER_SET_APPLICABILITY

Section 20.3, “[The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table](#)”

COLLATIONS

Section 10.1.9.3, “[SHOW Statements and INFORMATION_SCHEMA](#)”

Section 22.8.1, “[C API Data Structures](#)”

Section 8.2.4, “[Optimizing INFORMATION_SCHEMA Queries](#)”

Section 20.2, “[The INFORMATION_SCHEMA COLLATIONS Table](#)”

COLUMN_PRIVILEGES

Section 20.5, “[The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table](#)”

COLUMNS

Section 8.2.4, “[Optimizing INFORMATION_SCHEMA Queries](#)”

Section 20.4, “[The INFORMATION_SCHEMA COLUMNS Table](#)”

Section 20.11, “[The INFORMATION_SCHEMA PARAMETERS Table](#)”

Section 20.17, “[The INFORMATION_SCHEMA ROUTINES Table](#)”

E

[\[index top \[4331\]\]](#)

ENGINES

Section 17.5.9, “[Quick Reference: MySQL Cluster SQL Statements](#)”

Section 20.6, “[The INFORMATION_SCHEMA ENGINES Table](#)”

EVENTS

Section 19.4.4, “[Event Metadata](#)”

Section 19.4.2, “[Event Scheduler Configuration](#)”

Section 16.4.1.10, “[Replication of Invoked Features](#)”

Section 20.7, “[The INFORMATION_SCHEMA EVENTS Table](#)”

F

[\[index top \[4331\]\]](#)

FILES

Section 20.29, “[INFORMATION_SCHEMA Tables for MySQL Cluster](#)”

Section 17.5.12.1, “[MySQL Cluster Disk Data Objects](#)”

Section 20.29.1, “[The INFORMATION_SCHEMA FILES Table](#)”

Section 17.5.10, “[The ndbinfo MySQL Cluster Information Database](#)”

G

[\[index top \[4331\]\]](#)

GLOBAL_STATUS

Section 17.5, “[Management of MySQL Cluster](#)”

Section 17.6, “[MySQL Cluster Replication](#)”

Section 17.5.15, “[NDB API Statistics Counters and Variables](#)”

Section 20.8, “[The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables](#)”

GLOBAL_VARIABLES

Section 20.9, “[The INFORMATION_SCHEMA](#)”

[GLOBAL_VARIABLES](#) and [SESSION_VARIABLES Tables](#)”

I

[\[index top \[4331\]\]](#)

INFORMATION_SCHEMA.COLLATIONS

Section 10.4.2, “[Choosing a Collation ID](#)”

INFORMATION_SCHEMA.COLUMNS

Section 21.1, “[Performance Schema Quick Start](#)”

INFORMATION_SCHEMA.ENGINES

Section 21.1, “[Performance Schema Quick Start](#)”

Section 17.5.9, “[Quick Reference: MySQL Cluster SQL Statements](#)”

INFORMATION_SCHEMA.EVENTS

Section 13.7.5.19, “[SHOW EVENTS Syntax](#)”
Section 19.4.4, “[Event Metadata](#)”
Section 16.4.1.10, “[Replication of Invoked Features](#)”
Section E.1, “[Restrictions on Stored Programs](#)”
Section 19.4.6, “[The Event Scheduler and MySQL Privileges](#)”

INFORMATION_SCHEMA.FILES

Section 13.1.3, “[ALTER LOGFILE GROUP Syntax](#)”
Section 13.1.8, “[ALTER TABLESPACE Syntax](#)”
Section 13.1.14, “[CREATE LOGFILE GROUP Syntax](#)”
Section 13.1.18, “[CREATE TABLESPACE Syntax](#)”
Section 17.4.9, “[ndb_desc — Describe NDB Tables](#)”
Section 17.5.12.3, “[MySQL Cluster Disk Data Storage Requirements](#)”

INFORMATION_SCHEMA.GLOBAL_STATUS

Section 17.5.15, “[NDB API Statistics Counters and Variables](#)”

INFORMATION_SCHEMA.PARTITIONS

Section 18.2.5, “[KEY Partitioning](#)”
Section 18.2.3.1, “[RANGE COLUMNS partitioning](#)”
Section 18.2.7, “[How MySQL Partitioning Handles NULL](#)”
Section 18.3.4, “[Obtaining Information About Partitions](#)”
Section 20.12, “[The INFORMATION_SCHEMA PARTITIONS Table](#)”

INFORMATION_SCHEMA.PLUGINS

Section 13.7.3.3, “[INSTALL PLUGIN Syntax](#)”
Section 5.1.8.1, “[Installing and Uninstalling Plugins](#)”
Section 6.3.10.1, “[Installing the Audit Log Plugin](#)”
Section 6.3.6.2.1, “[Installing the PAM Authentication Plugin](#)”
Section 6.3.6.3.1, “[Installing the Windows Authentication Plugin](#)”
Section 5.1.8.2, “[Obtaining Server Plugin Information](#)”
Section 23.2.1, “[Plugin API Characteristics](#)”
Section 23.2.2, “[Plugin API Components](#)”
Section 16.3.8.2, “[Semisynchronous Replication Installation and Configuration](#)”
Section 23.2.4.2.1, “[Server Plugin Library and Plugin Descriptors](#)”
Section 8.11.6.1, “[Thread Pool Components and Installation](#)”
Section 23.2.4.6, “[Writing INFORMATION_SCHEMA Plugins](#)”
Section 23.2.4.8, “[Writing Audit Plugins](#)”
Section 23.2.4.5, “[Writing Daemon Plugins](#)”
Section 23.2.4.4, “[Writing Full-Text Parser Plugins](#)”
Section 23.2.4.9.1, “[Writing the Server-Side Authentication Plugin](#)”

INFORMATION_SCHEMA.PROCESSLIST

Section 20.14, “[The INFORMATION_SCHEMA PROCESSLIST Table](#)”
Section 21.7.5.2, “[The threads Table](#)”

INFORMATION_SCHEMA.ROUTINES

Chapter 20, [INFORMATION_SCHEMA Tables](#)
Section 22.2.6, “[Connector/Net Connection String Options Reference](#)”
Section 20.17, “[The INFORMATION_SCHEMA ROUTINES Table](#)”

INFORMATION_SCHEMA.SESSION_STATUS

Section 17.5.15, “[NDB API Statistics Counters and Variables](#)”

INFORMATION_SCHEMA.TABLE_CONSTRAINTS

Section 20.16, “[The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table](#)”

INFORMATION_SCHEMA.TABLES

Chapter 20, [INFORMATION_SCHEMA Tables](#)
Section 17.5.13.3, “[Adding MySQL Cluster Data Nodes Online: Detailed Example](#)”

INFORMATION_SCHEMA.TRIGGERS

Section 13.7.5.39, “[SHOW TRIGGERS Syntax](#)”

INFORMATION_SCHEMA.VIEWS

Section 19.5.3, “[Updatable and Insertable Views](#)”

INNODB_CMP

Section 20.28.1, “[The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables](#)”

INNODB_CMPMEM

Section 20.28.2, “[The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables](#)”

INNODB_LOCK_WAITS

Section 20.28.5, “[The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table](#)”

INNODB_LOCKS

Section 20.28.4, “[The INFORMATION_SCHEMA INNODB_LOCKS Table](#)”

INNODB_TRX

Section 20.28.3, “[The INFORMATION_SCHEMA INNODB_TRX Table](#)”

K

[\[index top \[4331\]\]](#)

KEY_COLUMN_USAGE

Section 8.2.4, “Optimizing INFORMATION_SCHEMA Queries”

Section 20.10, “The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table”

N

[\[index top \[4331\]\]](#)

NDB_TRANSID_MYSQL_CONNECTION_MAP

Section 17.5.10.15, “The ndbinfo server_operations Table”

Section 17.5.10.16, “The ndbinfo server_transactions Table”

ndb_transid_mysql_connection_map

Section 20.29, “INFORMATION_SCHEMA Tables for MySQL Cluster”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

P

[\[index top \[4331\]\]](#)

PARAMETERS

Section 20.11, “The INFORMATION_SCHEMA PARAMETERS Table”

Section 20.17, “The INFORMATION_SCHEMA ROUTINES Table”

PARTITIONS

Section 18.2.7, “How MySQL Partitioning Handles NULL”

Section 18.3.4, “Obtaining Information About Partitions”

Section 8.2.4, “Optimizing INFORMATION_SCHEMA Queries”

Chapter 18, *Partitioning*

Section 20.12, “The INFORMATION_SCHEMA PARTITIONS Table”

PLUGINS

Section 5.1.8.2, “Obtaining Server Plugin Information”

Section 20.6, “The INFORMATION_SCHEMA ENGINES Table”

Section 20.13, “The INFORMATION_SCHEMA PLUGINS Table”

PROCESSLIST

Section 13.7.5.30, “SHOW PROCESSLIST Syntax”

Section 8.12.5, “Examining Thread Information”

Section 20.14, “The INFORMATION_SCHEMA PROCESSLIST Table”

PROFILING

Section 13.7.5.31, “SHOW PROFILE Syntax”

Section 20.15, “The INFORMATION_SCHEMA PROFILING Table”

R

[\[index top \[4331\]\]](#)

REFERENTIAL_CONSTRAINTS

Section 8.2.4, “Optimizing INFORMATION_SCHEMA Queries”

Section 20.16, “The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table”

ROUTINES

Section 13.7.5.29, “SHOW PROCEDURE STATUS Syntax”

Section 19.2.3, “Stored Routine Metadata”

Section 20.11, “The INFORMATION_SCHEMA PARAMETERS Table”

Section 20.17, “The INFORMATION_SCHEMA ROUTINES Table”

S

[\[index top \[4331\]\]](#)

SCHEMA_PRIVILEGES

Section 20.19, “The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table”

SCHEMATA

Section 6.2.2, “Privilege System Grant Tables”

Section 20.18, “The INFORMATION_SCHEMA SCHEMATA Table”

SESSION_STATUS

Section 17.5, “Management of MySQL Cluster”

Section 17.6, “MySQL Cluster Replication”

Section 17.5.15, “NDB API Statistics Counters and Variables”

Section 20.8, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”

SESSION_VARIABLES

Section 20.9, “The `INFORMATION_SCHEMA` `GLOBAL_VARIABLES` and `SESSION_VARIABLES` Tables”

STATISTICS

Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”
Section 20.20, “The `INFORMATION_SCHEMA` `STATISTICS` Table”

T

[[index top \[4331\]](#)]

TABLE_CONSTRAINTS

Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”
Section 20.23, “The `INFORMATION_SCHEMA` `TABLE_CONSTRAINTS` Table”

TABLE_PRIVILEGES

Section 20.24, “The `INFORMATION_SCHEMA` `TABLE_PRIVILEGES` Table”

TABLES

Chapter 20, *INFORMATION_SCHEMA Tables*
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”
Section 20.21, “The `INFORMATION_SCHEMA` `TABLES` Table”

TABLESPACES

Section 20.22, “The `INFORMATION_SCHEMA` `TABLESPACES` Table”

TP_THREAD_GROUP_STATE

Section 20.30.2, “The `INFORMATION_SCHEMA` `TP_THREAD_GROUP_STATE` Table”
Section 20.30, “Thread Pool `INFORMATION_SCHEMA` Tables”
Section 8.11.6.1, “Thread Pool Components and Installation”

TP_THREAD_GROUP_STATS

Section 20.30.1, “The `INFORMATION_SCHEMA` `TP_THREAD_STATE` Table”

Section 20.30, “Thread Pool `INFORMATION_SCHEMA` Tables”

Section 8.11.6.1, “Thread Pool Components and Installation”

TP_THREAD_STATE

Section 20.30, “Thread Pool `INFORMATION_SCHEMA` Tables”
Section 8.11.6.1, “Thread Pool Components and Installation”

TRIGGERS

Section 13.7.5.13, “SHOW CREATE TRIGGER Syntax”
Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”
Section 20.25, “The `INFORMATION_SCHEMA` `TRIGGERS` Table”
Section 19.3.2, “Trigger Metadata”

U

[[index top \[4331\]](#)]

USER_PRIVILEGES

Section 20.26, “The `INFORMATION_SCHEMA` `USER_PRIVILEGES` Table”

V

[[index top \[4331\]](#)]

VIEWS

Section 13.7.5.14, “SHOW CREATE VIEW Syntax”
Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”
Section 20.27, “The `INFORMATION_SCHEMA` `VIEWS` Table”
Section 19.5.4, “View Metadata”

Join Types Index

[A](#) | [C](#) | [E](#) | [F](#) | [I](#) | [R](#) | [S](#) | [U](#)

A

[[index top](#) [4335]]

ALL

Section 8.13.6, “Nested-Loop Join Algorithms”

C

[[index top](#) [4335]]

const

Section 8.8.2, “EXPLAIN Output Format”
Section 8.13.9, “ORDER BY Optimization”
Section 13.2.9, “SELECT Syntax”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”

E

[[index top](#) [4335]]

eq_ref

Section 8.8.2, “EXPLAIN Output Format”
Section 14.10.1, “MERGE Table Advantages and Disadvantages”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”

F

[[index top](#) [4335]]

fulltext

Section 8.8.2, “EXPLAIN Output Format”

I

[[index top](#) [4335]]

index

Section 8.8.2, “EXPLAIN Output Format”
Section 8.13.6, “Nested-Loop Join Algorithms”

index_merge

Section 8.8.2, “EXPLAIN Output Format”

Section 8.13.2, “Index Merge Optimization”

index_subquery

Section 8.8.2, “EXPLAIN Output Format”
Section 13.2.10.10, “Optimizing Subqueries”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”

R

[[index top](#) [4335]]

range

Section 8.8.2, “EXPLAIN Output Format”
Section 8.13.2, “Index Merge Optimization”
Section 8.13.10.1, “Loose Index Scan”
Section 8.13.6, “Nested-Loop Join Algorithms”
Section 8.13.1, “Range Optimization”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”

ref

Section 8.8.2, “EXPLAIN Output Format”
Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
Section 14.10.1, “MERGE Table Advantages and Disadvantages”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”

ref_or_null

Section 8.8.2, “EXPLAIN Output Format”
Section 8.13.4, “IS NULL Optimization”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”

S

[[index top](#) [4335]]

system

Section 8.8.2, “EXPLAIN Output Format”
Section 13.2.9, “SELECT Syntax”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”

U

[[index top](#) [4335]]

unique_subquery

Section 8.8.2, “EXPLAIN Output Format”

Section 13.2.10.10, "Optimizing Subqueries"
Section 8.13.12, "Optimizing Subqueries with `EXISTS`
Strategy"

Operator Index

[Symbols](#) | [A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [I](#) | [L](#) | [N](#) | [O](#) | [R](#) | [X](#)

Symbols

[\[index top \[4337\]\]](#)

–

[Section 12.6.1, “Arithmetic Operators”](#)
[Section 12.10, “Cast Functions and Operators”](#)
[Section 12.7, “Date and Time Functions”](#)
[Section 11.1.1, “Numeric Type Overview”](#)
[Section 18.5, “Restrictions and Limitations on Partitioning”](#)

!

[Section 9.5, “Expression Syntax”](#)
[Section 12.3.3, “Logical Operators”](#)
[Section 12.3.1, “Operator Precedence”](#)

!=

[Section 12.3.2, “Comparison Functions and Operators”](#)
[Section 12.3.1, “Operator Precedence”](#)
[Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”](#)
[Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”](#)

%

[Section 12.6.1, “Arithmetic Operators”](#)

&

[Section 13.1.17, “CREATE TABLE Syntax”](#)
[Section 12.12, “Bit Functions”](#)
[Section 18.5, “Restrictions and Limitations on Partitioning”](#)

&&

[Section 12.3.3, “Logical Operators”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)

>

[Section 8.8.2, “EXPLAIN Output Format”](#)
[Section 12.3.2, “Comparison Functions and Operators”](#)
[Section 8.3.8, “Comparison of B-Tree and Hash Indexes”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 12.3.1, “Operator Precedence”](#)
[Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”](#)
[Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”](#)

>>

[Section 12.12, “Bit Functions”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 18.5, “Restrictions and Limitations on Partitioning”](#)

>=

[Section 8.8.2, “EXPLAIN Output Format”](#)
[Section 12.3.2, “Comparison Functions and Operators”](#)
[Section 8.3.8, “Comparison of B-Tree and Hash Indexes”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 12.3.1, “Operator Precedence”](#)
[Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”](#)
[Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”](#)

<

[Section 8.8.2, “EXPLAIN Output Format”](#)
[Section 12.3.2, “Comparison Functions and Operators”](#)
[Section 8.3.8, “Comparison of B-Tree and Hash Indexes”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 12.3.1, “Operator Precedence”](#)
[Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”](#)
[Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”](#)
[Section 3.3.4.6, “Working with NULL Values”](#)

<>

[Section 8.8.2, “EXPLAIN Output Format”](#)
[Section 12.3.2, “Comparison Functions and Operators”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 12.3.1, “Operator Precedence”](#)
[Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”](#)
[Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”](#)
[Section 3.3.4.6, “Working with NULL Values”](#)

<<

[Section 12.12, “Bit Functions”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 18.5, “Restrictions and Limitations on Partitioning”](#)

<=

[Section 8.8.2, “EXPLAIN Output Format”](#)
[Section 12.3.2, “Comparison Functions and Operators”](#)
[Section 8.3.8, “Comparison of B-Tree and Hash Indexes”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 12.3.1, “Operator Precedence”](#)

Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”

<=>

Section 8.8.2, “**EXPLAIN** Output Format”
Section 12.3.2, “Comparison Functions and Operators”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”
Section 12.2, “Type Conversion in Expression Evaluation”

Section 12.6.1, “Arithmetic Operators”
Section 11.1.1, “Numeric Type Overview”
Section 18.5, “Restrictions and Limitations on Partitioning”

+

Section 12.6.1, “Arithmetic Operators”
Section 12.10, “Cast Functions and Operators”
Section 12.7, “Date and Time Functions”
Section 11.1.1, “Numeric Type Overview”
Section 18.5, “Restrictions and Limitations on Partitioning”

/

Section 12.6.1, “Arithmetic Operators”
Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.1.4, “Server System Variables”

=

Section 8.8.2, “**EXPLAIN** Output Format”
Section 13.7.4, “**SET** Syntax”
Section 12.3.4, “Assignment Operators”
Section 12.3.2, “Comparison Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section E.4, “Restrictions on Subqueries”
Section 12.5.1, “String Comparison Functions”
Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”
Section 9.4, “User-Defined Variables”
Section 3.3.4.6, “Working with **NULL** Values”

^

Section 12.12, “Bit Functions”
Section 9.5, “Expression Syntax”
Section 12.3.1, “Operator Precedence”
Section 18.5, “Restrictions and Limitations on Partitioning”

|

Section 12.12, “Bit Functions”
Section 18.5, “Restrictions and Limitations on Partitioning”

||

Section 10.1.7.3, “**COLLATE** Clause Precedence”
Section 9.5, “Expression Syntax”
Section 12.3.3, “Logical Operators”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 10.1.9.1, “Result Strings”
Section 5.1.7, “Server SQL Modes”

~

Section 12.12, “Bit Functions”
Section 18.5, “Restrictions and Limitations on Partitioning”

A

[index top [4337]]

AND

Section 13.1.17, “**CREATE TABLE** Syntax”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.13.2, “Index Merge Optimization”
Section 12.3.3, “Logical Operators”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 8.13.12, “Optimizing Subqueries with **EXISTS** Strategy”
Section E.4, “Restrictions on Subqueries”
Section 3.6.7, “Searching on Two Keys”
Section 3.3.4.2, “Selecting Particular Rows”
Section 12.5.1, “String Comparison Functions”
Section 8.13.2.1, “The Index Merge Intersection Access Algorithm”
Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”
Section 19.5.2, “View Processing Algorithms”

B

[index top [4337]]

BETWEEN

Section 8.8.2, “[EXPLAIN Output Format](#)”
Section 12.3.2, “[Comparison Functions and Operators](#)”
Section 8.3.8, “[Comparison of B-Tree and Hash Indexes](#)”
Section 8.13.1.2, “[The Range Access Method for Multiple-Part Indexes](#)”
Section 8.13.1.1, “[The Range Access Method for Single-Part Indexes](#)”
Section 12.2, “[Type Conversion in Expression Evaluation](#)”

BINARY

Section 12.10, “[Cast Functions and Operators](#)”
Section 3.3.4.7, “[Pattern Matching](#)”
Section 3.3.4.4, “[Sorting Rows](#)”
Section 10.1.7.7, “[The BINARY Operator](#)”

BINARY str

Section 12.10, “[Cast Functions and Operators](#)”

C

[\[index top \[4337\]\]](#)

CASE

Section 13.6.5.1, “[CASE Syntax](#)”
Section 12.4, “[Control Flow Functions](#)”
Section 9.5, “[Expression Syntax](#)”
Section 1.8.4, “[MySQL Extensions to Standard SQL](#)”

CASE value WHEN END

Section 12.4, “[Control Flow Functions](#)”

CASE WHEN END

Section 12.4, “[Control Flow Functions](#)”

CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END

Section 12.4, “[Control Flow Functions](#)”

D

[\[index top \[4337\]\]](#)

DIV

Section 12.6.1, “[Arithmetic Operators](#)”
Section 18.5, “[Restrictions and Limitations on Partitioning](#)”

E

[\[index top \[4337\]\]](#)

expr BETWEEN min AND max

Section 12.3.2, “[Comparison Functions and Operators](#)”

expr LIKE pat

Section 12.5.1, “[String Comparison Functions](#)”

expr NOT BETWEEN min AND max

Section 12.3.2, “[Comparison Functions and Operators](#)”

expr NOT LIKE pat

Section 12.5.1, “[String Comparison Functions](#)”

expr NOT REGEXP pat

Section 12.5.2, “[Regular Expressions](#)”

expr NOT RLIKE pat

Section 12.5.2, “[Regular Expressions](#)”

expr REGEXP pat

Section 12.5.2, “[Regular Expressions](#)”

expr RLIKE pat

Section 12.5.2, “[Regular Expressions](#)”

expr1 SOUNDS LIKE expr2

Section 12.5, “[String Functions](#)”

I

[\[index top \[4337\]\]](#)

IS

Section 12.3.1, “[Operator Precedence](#)”

IS boolean_value

Section 12.3.2, “[Comparison Functions and Operators](#)”

IS NOT boolean_value

Section 12.3.2, “[Comparison Functions and Operators](#)”

IS NOT NULL

Section 12.3.2, “[Comparison Functions and Operators](#)”
Section C.5.5.3, “[Problems with NULL Values](#)”
Section 8.13.1.1, “[The Range Access Method for Single-Part Indexes](#)”
Section 3.3.4.6, “[Working with NULL Values](#)”

IS NULL

Section 8.8.2, “[EXPLAIN Output Format](#)”

Section 8.13.4, “IS NULL Optimization”
Section 12.3.2, “Comparison Functions and Operators”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”
Section C.5.5.3, “Problems with NULL Values”
Section 5.1.4, “Server System Variables”
Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”
Section 3.3.4.6, “Working with NULL Values”

L

[index top [4337]]

LIKE

Section 13.7.5.4, “SHOW CHARACTER SET Syntax”
Section 13.7.5.5, “SHOW COLLATION Syntax”
Section 13.7.5.6, “SHOW COLUMNS Syntax”
Section 13.7.5.15, “SHOW DATABASES Syntax”
Section 13.7.5.19, “SHOW EVENTS Syntax”
Section 13.7.5.25, “SHOW OPEN TABLES Syntax”
Section 13.7.5.29, “SHOW PROCEDURE STATUS Syntax”
Section 13.7.5.36, “SHOW STATUS Syntax”
Section 13.7.5.37, “SHOW TABLE STATUS Syntax”
Section 13.7.5.38, “SHOW TABLES Syntax”
Section 13.7.5.39, “SHOW TRIGGERS Syntax”
Section 13.7.5.40, “SHOW VARIABLES Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 12.10, “Cast Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 21.2.3.2.1, “Event Pre-Filtering”
Section 20.31, “Extensions to SHOW Statements”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 3.3.4.7, “Pattern Matching”
Section 17.5.9, “Quick Reference: MySQL Cluster SQL Statements”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 6.2.3, “Specifying Account Names”
Section 12.5.1, “String Comparison Functions”
Section 9.1.1, “String Literals”
Section 5.1.5.1, “Structured System Variables”
Section 11.4.5, “The SET Type”
Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”
Section 5.1.5, “Using System Variables”

LIKE 'pattern'

Section 13.7.5, “SHOW Syntax”
Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”

LIKE ... ESCAPE

Section C.5.8, “Known Issues in MySQL”

N

[index top [4337]]

N % M

Section 12.6.1, “Arithmetic Operators”
Section 12.6.2, “Mathematical Functions”

N MOD M

Section 12.6.1, “Arithmetic Operators”
Section 12.6.2, “Mathematical Functions”

NOT

Section 12.3.3, “Logical Operators”
Section 5.1.7, “Server SQL Modes”

NOT LIKE

Section 3.3.4.7, “Pattern Matching”
Section 12.5.1, “String Comparison Functions”

NOT REGEXP

Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 3.3.4.7, “Pattern Matching”
Section 12.5.1, “String Comparison Functions”

NOT RLIKE

Section 3.3.4.7, “Pattern Matching”
Section 12.5.1, “String Comparison Functions”

O

[index top [4337]]

OR

Section 13.7.1.3, “GRANT Syntax”
Section 9.5, “Expression Syntax”
Section 8.13.2, “Index Merge Optimization”
Section 12.3.3, “Logical Operators”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”
Section 3.6.7, “Searching on Two Keys”

[Section 3.3.4.2, “Selecting Particular Rows”](#)
[Section 5.1.7, “Server SQL Modes”](#)
[Section 12.5.1, “String Comparison Functions”](#)
[Section 8.13.2.3, “The Index Merge Sort-Union Access Algorithm”](#)
[Section 8.13.2.2, “The Index Merge Union Access Algorithm”](#)
[Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”](#)
[Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”](#)

R

[\[index top \[4337\]\]](#)

REGEXP

[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 12.3.1, “Operator Precedence”](#)
[Section 3.3.4.7, “Pattern Matching”](#)
[Section 12.5.2, “Regular Expressions”](#)
[Section E.7, “Restrictions on Character Sets”](#)

RLIKE

[Section 3.3.4.7, “Pattern Matching”](#)
[Section 12.5.2, “Regular Expressions”](#)
[Section E.7, “Restrictions on Character Sets”](#)

X

[\[index top \[4337\]\]](#)

XOR

[Section 12.16.1, “GROUP BY \(Aggregate\) Functions”](#)
[Section 12.3.3, “Logical Operators”](#)

Option Index

[Symbols](#) | [A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#) | [Z](#)

Symbols

[[index top](#) [4343]]

--

[Section 1.8.5.5, "'--' as the Start of a Comment"](#)
[Section 4.8.2, "replace — A String-Replacement Utility"](#)

-#

[Section 4.6.3.1, "myisamchk General Options"](#)
[Section 4.5.1.1, "mysql Options"](#)
[Section 4.6.5, "myisampack — Generate Compressed, Read-Only MyISAM Tables"](#)
[Section 4.6.7, "mysqlbinlog — Utility for Processing Binary Log Files"](#)
[Section 4.5.2, "mysqladmin — Client for Administering a MySQL Server"](#)
[Section 4.5.3, "mysqlcheck — A Table Maintenance Program"](#)
[Section 4.5.4, "mysqldump — A Database Backup Program"](#)
[Section 4.5.5, "mysqlimport — A Data Import Program"](#)
[Section 4.5.6, "mysqlshow — Display Database, Table, and Column Information"](#)
[Section 4.5.7, "mysqlslap — Load Emulation Client"](#)
[Section 4.7.3, "my_print_defaults — Display Options from Option Files"](#)
[Section 4.4.1, "comp_err — Compile MySQL Error Message File"](#)
[Section 4.8.2, "replace — A String-Replacement Utility"](#)
[Section 5.1.3, "Server Command Options"](#)
[Section 23.4.3, "The DEBUG Package"](#)

/i

[Section 2.3.4.2, "Automating MySQL Installation on Microsoft Windows using the MSI Package"](#)

/passive

[Section 2.3.4.2, "Automating MySQL Installation on Microsoft Windows using the MSI Package"](#)

/quiet

[Section 2.3.4.2, "Automating MySQL Installation on Microsoft Windows using the MSI Package"](#)

/uninstall

[Section 2.3.4.2, "Automating MySQL Installation on Microsoft Windows using the MSI Package"](#)

/x

[Section 2.3.4.2, "Automating MySQL Installation on Microsoft Windows using the MSI Package"](#)

-1

[Section 4.5.3, "mysqlcheck — A Table Maintenance Program"](#)

-?

[Section 4.6.3.1, "myisamchk General Options"](#)
[Section 4.5.1.1, "mysql Options"](#)
[Section 17.4.24, "Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs"](#)
[Section 17.4.6, "ndb_config — Extract MySQL Cluster Configuration Information"](#)
[Section 4.6.2, "myisam_ftdump — Display Full-Text Index information"](#)
[Section 4.6.4, "myisamlog — Display MyISAM Log File Contents"](#)
[Section 4.6.5, "myisampack — Generate Compressed, Read-Only MyISAM Tables"](#)
[Section 4.6.14, "mysql_waitpid — Kill Process and Wait for Its Termination"](#)
[Section 4.6.15, "mysql_zap — Kill Processes That Match a Pattern"](#)
[Section 4.6.6, "mysqlaccess — Client for Checking Access Privileges"](#)
[Section 4.6.7, "mysqlbinlog — Utility for Processing Binary Log Files"](#)
[Section 4.6.9, "mysqlhotcopy — A Database Backup Program"](#)
[Section 4.5.2, "mysqladmin — Client for Administering a MySQL Server"](#)
[Section 4.5.3, "mysqlcheck — A Table Maintenance Program"](#)
[Section 4.5.4, "mysqldump — A Database Backup Program"](#)
[Section 4.5.5, "mysqlimport — A Data Import Program"](#)
[Section 4.5.6, "mysqlshow — Display Database, Table, and Column Information"](#)
[Section 4.5.7, "mysqlslap — Load Emulation Client"](#)
[Section 4.7.3, "my_print_defaults — Display Options from Option Files"](#)
[Section 4.4.1, "comp_err — Compile MySQL Error Message File"](#)
[Section 4.4.4, "mysql_plugin — Configure MySQL Server Plugins"](#)
[Section 4.8.1, "perror — Explain Error Codes"](#)
[Section 4.8.2, "replace — A String-Replacement Utility"](#)
[Section 4.8.3, "resolveip — Resolve Host name to IP Address or Vice Versa"](#)
[Section 5.1.3, "Server Command Options"](#)

Section 4.2.3.1, “Using Options on the Command Line”

--

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

A

[index top [4343]]

-A

Section 4.5.1.1, “mysql Options”

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.6.3.4, “Other myisamchk Options”

-a

Section 7.6.4, “MyISAM Table Optimization”

Section 17.5.2, “Commands in the MySQL Cluster Management Client”

Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.5.7, “mysqlslap — Load Emulation Client”

Section 4.6.3.4, “Other myisamchk Options”

--abort-slave-event-count

Section 16.1.3.3, “Replication Slave Options and Variables”

--add-drop-database

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 7.4.1, “Dumping Data in SQL Format with mysqldump”

--add-drop-table

Section 4.5.4, “mysqldump — A Database Backup Program”

--add-drop-trigger

Section 4.5.4, “mysqldump — A Database Backup Program”

--add-locks

Section 4.5.4, “mysqldump — A Database Backup Program”

--addtodest

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

--all

Section 1.4, “What Is New in MySQL 5.5”

--all-databases

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”

Section 7.4.1, “Dumping Data in SQL Format with mysqldump”

Section 9.2.3, “Mapping of Identifiers to File Names”

Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”

Section 7.4.2, “Reloading SQL-Format Backups”

Section 17.5.10, “The ndbinfo MySQL Cluster Information Database”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

--all-in-1

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--all-tablespaces

Section 4.5.4, “mysqldump — A Database Backup Program”

--allow-keywords

Section 4.5.4, “mysqldump — A Database Backup Program”

--allow-suspicious-udfs

Section 5.1.3, “Server Command Options”

Section 23.3.2.6, “User-Defined Function Security Precautions”

--allowold

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

--analyze

Section 4.6.3.1, “myisamchk General Options”

Section 7.6.4, “MyISAM Table Optimization”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.6.3.4, “Other `myisamchk` Options”

--ansi

Section 1.8.3, “Running MySQL in ANSI Mode”

Section 5.1.3, “Server Command Options”

antonio

Section 6.3.6.2.2.3, “Unix Password Authentication with Proxy Users and Group Mapping”

--append

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--apply-slave-statements

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--audit-log

Section 6.3.10.4, “Audit Log Plugin Options and Variables”

Section 6.3.10.1, “Installing the Audit Log Plugin”

--auto-generate-sql

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-generate-sql-add-autoincrement

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-generate-sql-execute-number

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-generate-sql-guid-primary

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-generate-sql-load-type

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-generate-sql-secondary-indexes

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-generate-sql-unique-query-number

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-generate-sql-unique-write-number

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-generate-sql-write-number

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--auto-rehash

Section 4.5.1.2, “`mysql` Commands”

Section 4.5.1.1, “`mysql` Options”

--auto-repair

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--auto-vertical-output

Section 4.5.1.1, “`mysql` Options”

--autocommit

Section 5.1.4, “Server System Variables”

B

[index top [4343]]

-B

Section 4.6.3.3, “`myisamchk` Repair Options”

Section 4.5.1.1, “`mysql` Options”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

-b

Section 4.5.1.1, “`mysql` Options”

Section 17.4.9, “`ndb_desc` — Describe NDB Tables”

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 4.6.3.4, “Other `myisamchk` Options”

Section 5.1.3, “Server Command Options”

--back_log

Section 2.7, “Installing MySQL on Solaris and OpenSolaris”

--backup

Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

backup_path

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--backup_path

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--base64-output

Section 4.6.7.2, “mysqlbinlog Row Event Display”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

--basedir

Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.3.3, “mysql.server — MySQL Server Startup Script”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”
Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”
Section 2.11.1, “Unix Postinstallation Procedures”

basedir

Section 2.3.7.2, “Creating an Option File”
Section 2.3.6.13, “MySQL Server Instance Config Wizard: Creating an Instance from the Command Line”

Section 2.11.1.2, “Starting and Stopping MySQL Automatically”

Section 2.3.8, “Troubleshooting a Microsoft Windows MySQL Server Installation”

--batch

Section 4.5.1.3, “mysql History File”
Section 4.5.1.1, “mysql Options”

--big-tables

Section 5.1.3, “Server Command Options”

--bind-address

Section 4.5.1.1, “mysql Options”
Section C.5.2.2, “Can't connect to [local] MySQL server”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 5.1.9.2, “Configuring the MySQL Server to Permit IPv6 Connections”
Section 5.1.9.4, “Connecting Using IPv6 Nonlocal Host Addresses”
Section 5.1.9.3, “Connecting Using the IPv6 Local Host Address”
Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”
Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 5.1.9, “IPv6 Support”
Section 5.1.9.5, “Obtaining an IPv6 Address from a Broker”
Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 5.1.3, “Server Command Options”

--binlog-do-db

Section 16.1.3.4, “Binary Log Options and Variables”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.2.4, “The Binary Log”

--binlog-format

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.2.4.1, “Binary Logging Formats”

Section 17.6.2, “General Requirements for MySQL Cluster Replication”

Section 5.1.3, “Server Command Options”

Section 5.2.4.2, “Setting The Binary Log Format”

Section 17.1.6.6, “Unsupported or Missing Features in MySQL Cluster”

--binlog-ignore-db

Section 16.1.3.4, “Binary Log Options and Variables”

Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.2.4, “The Binary Log”

--binlog-row-event-max-size

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.2.4.2, “Setting The Binary Log Format”

--binlog_format

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

--blob-info

Section 17.4.9, “`ndb_desc` — Describe NDB Tables”

--block-search

Section 4.6.3.4, “Other `myisamchk` Options”

--bootstrap

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 5.1.3, “Server Command Options”

--brief

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

C

[index top [4343]]

-C

Section 4.6.3.2, “`myisamchk` Check Options”

Section 4.5.1.1, “`mysql` Options”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

Section 5.1.3, “Server Command Options”

-C

Section 4.6.3.2, “`myisamchk` Check Options”

Section 4.5.1.1, “`mysql` Options”

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”

Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”

Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”

Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”

Section 2.3.6.13, “MySQL Server Instance Config Wizard: Creating an Instance from the Command Line”

--cflags

Section 2.10.5, “Dealing with Problems Compiling MySQL”

Section 4.7.2, “mysql_config — Get Compile Options for Compiling Clients”

--character-set-client-handshake

Section 5.1.3, “Server Command Options”

Section 10.1.14.7.1, “The cp932 Character Set”

--character-set-filesystem

Section 5.1.3, “Server Command Options”

--character-set-server

Section 10.5, “Character Set Configuration”

Section 10.1.5, “Configuring the Character Set and Collation for Applications”

Section 16.4.1.2, “Replication and Character Sets”

Section 10.1.3.1, “Server Character Set and Collation”

Section 5.1.3, “Server Command Options”

Section 1.4, “What Is New in MySQL 5.5”

--character-sets-dir

Section 4.6.3.3, “myisamchk Repair Options”

Section 4.5.1.1, “mysql Options”

Section C.5.2.17, “Can't initialize character set”

Section 10.5, “Character Set Configuration”

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.5.5, “mysqlimport — A Data Import Program”

Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

Section 5.1.3, “Server Command Options”

--character_set_server

Section 2.10.4, “MySQL Source-Configuration Options”

--charset

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--check

Section 4.6.3.2, “myisamchk Check Options”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--check-only-changed

Section 4.6.3.2, “myisamchk Check Options”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--check-upgrade

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”

--checkpoint

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

--chroot

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

Section 5.1.3, “Server Command Options”

CMAKE_INSTALL_PREFIX

Section 23.2.4.3, “Compiling and Installing Plugin Libraries”

Section 2.10.3, “Installing MySQL from a Development Source Tree”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 5.3.3, “Running Multiple MySQL Instances on Unix”

--collation-server

Section 10.5, “Character Set Configuration”

Section 10.1.5, “Configuring the Character Set and Collation for Applications”

Section 16.4.1.2, “Replication and Character Sets”

Section 10.1.3.1, “Server Character Set and Collation”

Section 5.1.3, “Server Command Options”

Section 1.4, “What Is New in MySQL 5.5”

--collation_server

Section 2.10.4, “MySQL Source-Configuration Options”

--column-names

Section 4.5.1.1, “mysql Options”

Section 4.2.3.2, “Program Option Modifiers”

--column-type-info

Section 4.5.1.1, “mysql Options”

--columns

Section 4.5.5, “mysqlimport — A Data Import Program”

--comments

Section 4.5.1.1, “mysql Options”

Section 4.5.4, “mysqldump — A Database Backup Program”

--commit

Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”

Section 4.5.7, “mysqlslap — Load Emulation Client”

--comp

Section 4.2.3, “Specifying Program Options”

--compact

Section 4.5.4, “mysqldump — A Database Backup Program”

--compatible

Section 4.5.4, “mysqldump — A Database Backup Program”

COMPILATION_COMMENT

Section 5.1.4, “Server System Variables”

--complete-insert

Section 4.5.4, “mysqldump — A Database Backup Program”

--compr

Section 4.2.3, “Specifying Program Options”

--compress

Section 4.5.1.1, “mysql Options”

Section 13.2.6, “LOAD DATA INFILE Syntax”

Section 6.3.8.1, “Basic SSL Concepts”

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.5.5, “mysqlimport — A Data Import Program”

Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

Section 4.5.7, “mysqlslap — Load Emulation Client”

Section 4.2.3, “Specifying Program Options”

--concurrency

Section 4.5.7, “mysqlslap — Load Emulation Client”

--config-cache

Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”

Section 17.3.2, “MySQL Cluster Configuration Files”

--config-dir

Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”

--config-file

Section 17.4.6, “ndb_config — Extract MySQL Cluster Configuration Information”

Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”

Section 4.7.3, “my_print_defaults — Display Options from Option Files”

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

Section 17.2.4, “Initial Startup of MySQL Cluster”

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”

Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”

Section 17.3.2.1, “MySQL Cluster Configuration: Basic Example”

Section 1.4, “What Is New in MySQL 5.5”

--config_from_node

Section 17.4.6, “ndb_config — Extract MySQL Cluster Configuration Information”

--configdir

Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”

Section 17.3.2, “MySQL Cluster Configuration Files”

--configinfo

Section 17.4.6, “ndb_config — Extract MySQL Cluster Configuration Information”

--connect-delay

Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”

--connect-retries

Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”

--connect-string

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

--connections

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

--console

Section 14.3.14.3, “InnoDB General Troubleshooting”

Section 14.3.14.2, “`SHOW ENGINE INNODB STATUS` and the InnoDB Monitors”

Section 14.3.3.2, “Creating the InnoDB Tablespace”

Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”

Section C.5.4.1.1, “Resetting the Root Password: Windows Systems”

Section 5.1.3, “Server Command Options”

Section 2.3.7.4, “Starting the Server for the First Time”

Section 5.2.2, “The Error Log”

--copy

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--core-file

Section 23.4.1.4, “Debugging `mysqld` under `gdb`”

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

Section 5.1.3, “Server Command Options”

--core-file-size

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

Section 5.1.3, “Server Command Options”

--correct-checksum

Section 4.6.3.3, “`myisamchk` Repair Options”

--count

Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

--create

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--create-and-drop-schema

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--create-options

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 1.4, “What Is New in MySQL 5.5”

--create-schema

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--csv

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

D

[index top [4343]]

-D

Section 4.6.3.3, “`myisamchk` Repair Options”

Section 4.5.1.1, “`mysql` Options”

Section C.5.2.17, “`Can't initialize character set`”

Section 10.3, “Adding a Character Set”

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”

Section 2.1.2.2, “Choosing a Distribution Format”

Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”

Section 23.4.1.1, “Compiling MySQL for Debugging”

Section 22.7.1, “Compiling Programs with `libmysqld`”

Section 23.4.2, “Debugging a MySQL Client”

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

Section 2.10.2, “Installing MySQL from a Standard Source Distribution”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 23.1.1, “MySQL Threads”

Chapter 18, *Partitioning*

Section 21.2.1, “Performance Schema Build Configuration”

Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 14.8, “The `ARCHIVE` Storage Engine”
Section 14.9, “The `BLACKHOLE` Storage Engine”
Section 14.12, “The `EXAMPLE` Storage Engine”
Section 14.11, “The `FEDERATED` Storage Engine”
Section 5.4, “Tracing `mysqld` Using DTrace”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

-d

Section 4.6.3.1, “`myisamchk` General Options”
Section 17.4.13, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”
Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”
Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”
Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.6.3.4, “Other `myisamchk` Options”
Section 5.1.4, “Server System Variables”

--daemon

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”

--data-file-length

Section 4.6.3.3, “`myisamchk` Repair Options”

--database

Section 4.5.1.1, “`mysql` Options”
Section 17.4.13, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”
Section 17.4.22, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--databases

Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.1, “Dumping Data in SQL Format with `mysqldump`”
Section 7.4.5.1, “Making a Copy of a Database”
Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”

--datadir

Section 2.3.7.2, “Creating an Option File”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 5.3.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 5.3.1, “Setting Up Multiple Data Directories”
Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”
Section 2.11.1, “Unix Postinstallation Procedures”
Section 4.2.3.3, “Using Option Files”

datadir

Section 2.3.7.2, “Creating an Option File”

Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”

Section 2.11.1.2, “Starting and Stopping MySQL Automatically”

Section 2.3.8, “Troubleshooting a Microsoft Windows MySQL Server Installation”

Section E.10.5, “Windows Platform Limitations”

--db

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--debug

Section 4.6.3.1, “`myisamchk` General Options”

Section 4.5.1.1, “`mysql` Options”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 23.4.1.1, “Compiling MySQL for Debugging”

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”

Section 2.3.7.5, “Starting MySQL from the Windows Command Line”

Section 23.4.3, “The DEBUG Package”

--debug-check

Section 4.5.1.1, “`mysql` Options”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

--debug-info

Section 4.5.1.1, “`mysql` Options”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

--debug-sync-timeout

Section 2.10.4, “MySQL Source-Configuration Options”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

--default-auth

Section 4.5.1.1, “`mysql` Options”

Section 22.8.10, “C API Client Plugin Functions”

Section 23.2.4.2.3, “Client Plugin Descriptors”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 6.3.6, “Pluggable Authentication”
Section 6.3.6.1, “The Native Authentication Plugins”
Section 23.2.4.9.3, “Using the Authentication Plugins”

--default-character-set

Section 4.5.1.1, “`mysql` Options”
Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 10.5, “Character Set Configuration”
Section 10.1.5, “Configuring the Character Set and Collation for Applications”
Section 10.1.4, “Connection Character Sets and Collations”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 6.3.1, “User Names and Passwords”
Section 1.4, “What Is New in MySQL 5.5”

--default-collation

Section 5.1.3, “Server Command Options”
Section 1.4, “What Is New in MySQL 5.5”

--default-storage-engine

Section 14.3.4, “`InnoDB` Startup Options and System Variables”
Section 14.3.2, “Configuring `InnoDB`”
Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 14.1, “Setting the Storage Engine”
Section 1.4, “What Is New in MySQL 5.5”

default-storage-engine

Section 14.1, “Setting the Storage Engine”

--default-table-type

Section 1.4, “What Is New in MySQL 5.5”

--default-time-zone

Section 10.6, “MySQL Server Time Zone Support”
Section 16.4.1.30, “Replication and Time Zones”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

--default.key_buffer_size

Section 5.1.5.1, “Structured System Variables”

DEFAULT_CHARSET

Section 10.1.5, “Configuring the Character Set and Collation for Applications”
Section 10.1.3.1, “Server Character Set and Collation”

DEFAULT_COLLATION

Section 10.1.5, “Configuring the Character Set and Collation for Applications”
Section 10.1.3.1, “Server Character Set and Collation”

--defaults-extra-file

Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.2.3.3, “Using Option Files”
Section 1.4, “What Is New in MySQL 5.5”

--defaults-file

Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”
Section 14.3.2, “Configuring `InnoDB`”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 22.7.3, “Options with the Embedded Server”
Section C.5.4.1.1, “Resetting the Root Password: Windows Systems”
Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 5.3.3, “Running Multiple MySQL Instances on Unix”
Section 5.3.2.2, “Starting Multiple MySQL Instances as Windows Services”

Section 5.3.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”

Section 2.3.7.7, “Starting MySQL as a Windows Service”

Section 2.3.6.1, “Starting the MySQL Server Instance Configuration Wizard”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

--defaults-group-suffix

Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”

Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”

Section 2.13, “Environment Variables”

--delay

Section 17.4.2, “`ndbinfo_select_all` — Select From ndbinfo Tables”

--delay-key-write

Section 14.5.1, “MyISAM Startup Options”

Section 8.10.5, “External Locking”

Section 5.1.3, “Server Command Options”

Section 1.4, “What Is New in MySQL 5.5”

Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

--delay-key-write-for-all-tables

Section 1.4, “What Is New in MySQL 5.5”

--delay_key_write

Section 5.1.4, “Server System Variables”

Section 5.1.5, “Using System Variables”

--delayed-insert

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--delete

Section 17.4.13, “`ndb_index_stat` — NDB Index Statistics Utility”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--delete-master-logs

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--delimiter

Section 4.5.1.1, “`mysql` Options”

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--demangle

Section 23.4.1.5, “Using a Stack Trace”

--des-key-file

Section 13.7.6.3, “`FLUSH` Syntax”

Section 12.13, “Encryption and Compression Functions”

Section 5.1.3, “Server Command Options”

--descending

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

--description

Section 4.6.3.4, “Other `myisamchk` Options”

--detach

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--disable

Section 4.2.3.2, “Program Option Modifiers”

--disable-auto-rehash

Section 4.5.1.1, “`mysql` Options”

--disable-indexes

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--disable-keys

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--disable-log-bin

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--disable-named-commands

Section 4.5.1.1, “`mysql` Options”

--disable-plugin_name

Section 5.1.8.1, “Installing and Uninstalling Plugins”

DISABLE_GRANT_OPTIONS

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

--disconnect-slave-event-count

Section 16.1.3.3, “Replication Slave Options and Variables”

--disk

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

--dont_ignore_systab_0

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--dryrun

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

--dump

Section 17.4.13, “`ndb_index_stat` — NDB Index Statistics Utility”

Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”

--dump-date

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--dump-slave

Section 4.5.4, “`mysqldump` — A Database Backup Program”

E

[index top [4343]]

-E

Section 4.5.1.1, “`mysql` Options”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

-e

Section 4.6.3.2, “`myisamchk` Check Options”

Section 4.6.3.1, “`myisamchk` General Options”

Section 4.6.3.3, “`myisamchk` Repair Options”

Section 4.5.1.1, “`mysql` Options”

Section 13.2.7, “`LOAD XML` Syntax”

Section 17.4.5, “`ndb_mgm` — The MySQL Cluster Management Client”

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”

Section 7.6.2, “How to Check `MyISAM` Tables for Errors”

Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”

Section 4.6.3.5, “Obtaining Table Information with `myisamchk`”

Section 17.2.6, “Safe Shutdown and Restart of MySQL Cluster”

Section 4.2.3.1, “Using Options on the Command Line”

Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”

--embedded

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

--enable-cleartext-plugin

Section 4.5.1.1, “`mysql` Options”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 6.3.6.4, “The Cleartext Client-Side Authentication Plugin”

--enable-locking

Section 5.1.3, “Server Command Options”

Section 1.4, “What Is New in MySQL 5.5”

--enable-named-pipe

Section C.5.2.2, “Can't connect to [local] MySQL server”

Section 4.2.2, “Connecting to the MySQL Server”

Section 2.3.7.3, “Selecting a MySQL Server Type”

Section 5.1.3, “Server Command Options”

--enable-plugin_name

Section 5.1.8.1, “Installing and Uninstalling Plugins”

--enable-pstack

Section 5.1.3, “Server Command Options”

Section 1.4, “What Is New in MySQL 5.5”

--engine

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--engine-condition-pushdown

Section 17.3.2.2, “Recommended Starting Configuration for MySQL Cluster”

event-scheduler

Section 19.4.2, “Event Scheduler Configuration”

--event-scheduler

Section 19.4.2, “Event Scheduler Configuration”

Section 5.1.3, “Server Command Options”

--events

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 7.4.5.3, “Dumping Stored Programs”

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

--example

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

--exclude-*

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--exclude-databases

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--exclude-missing-columns

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--exclude-tables

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--excludedbs

Section 17.4.22, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--excludetables

Section 17.4.22, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--execute

Section 4.5.1.3, “mysql History File”

Section 4.5.1.1, “mysql Options”

Section 17.4.5, “ndb_mgm — The MySQL Cluster Management Client”

Section 4.2.3.1, “Using Options on the Command Line”

Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”

--exit-info

Section 5.1.3, “Server Command Options”

--extend-check

Section 4.6.3.2, “myisamchk Check Options”

Section 4.6.3.1, “myisamchk General Options”

Section 4.6.3.3, “myisamchk Repair Options”

--extended

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--extended-insert

Section 4.5.4, “mysqldump — A Database Backup Program”

--external-locking

Section 14.5.1, “MyISAM Startup Options”

Section 8.10.5, “External Locking”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 8.11.1, “System Factors and Startup Parameter Tuning”

Section 1.4, “What Is New in MySQL 5.5”

--extra-file

Section 4.7.3, “my_print_defaults — Display Options from Option Files”

--extra-node-info

Section 17.4.9, “ndb_desc — Describe NDB Tables”

--extra-partition-info

Section 17.4.9, “ndb_desc — Describe NDB Tables”

Section 17.5.10.4, “The ndbinfo

cluster_operations Table”

Section 17.5.10.15, “The ndbinfo

server_operations Table”

F

[index top [4343]]

-F

Section 4.6.3.2, “myisamchk Check Options”

Section 4.5.1.2, “mysql Commands”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.7, “mysqlslap — Load Emulation Client”
Section 4.4.1, “comp_err — Compile MySQL Error Message File”

-f

Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Options”
Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 17.4.6, “ndb_config — Extract MySQL Cluster Configuration Information”
Section 17.4.4, “ndb_mgmd — The MySQL Cluster Management Server Daemon”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.15, “mysqlzap — Kill Processes That Match a Pattern”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 17.2.4, “Initial Startup of MySQL Cluster”
Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 23.4.1.5, “Using a Stack Trace”

--fast

Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--federated

Section 14.11, “The FEDERATED Storage Engine”

--fields

Section 17.4.6, “ndb_config — Extract MySQL Cluster Configuration Information”

--fields-enclosed-by

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”

--fields-escaped-by

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”

--fields-optionally-enclosed-by

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”

--fields-terminated-by

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”

--fields-xxx

Section 4.5.4, “mysqldump — A Database Backup Program”

--first-slave

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 1.4, “What Is New in MySQL 5.5”

--fix-db-names

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--fix-table-names

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--flush

Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

--flush-logs

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 7.3.1, “Establishing a Backup Policy”
Section 5.2, “MySQL Server Logs”

--flush-privileges

Section 4.5.4, “mysqldump — A Database Backup Program”

--flush_time

Section 23.1.1, “MySQL Threads”

--flushlog

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

--force

Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.10, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 23.1.2, “The MySQL Test Suite”
Section 3.5, “Using mysql in Batch Mode”

--force-if-open

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

--force-read

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

--foreground

Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”

--format

Section 17.4.22, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--fs

Section 17.4.12, “ndb_error_reporter — NDB Error-Reporting Utility”

G

[index top [4343]]

-G

Section 4.5.1.1, “mysql Options”

-g

Section 4.5.1.1, “mysql Options”
Section 23.4.1.1, “Compiling MySQL for Debugging”
Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”

--gci

Section 17.4.19, “ndb_select_all — Print Rows from an NDB Table”

--gdb

Section 23.4.1.4, “Debugging mysqld under gdb”
Section 5.1.3, “Server Command Options”

--general-log

Section 5.1.3, “Server Command Options”

--general_log

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 5.2.3, “The General Query Log”

--general_log_file

Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

Section 5.2.3, “The General Query Log”

H

[index top [4343]]

-H

Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Options”
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.4.1, “comp_err — Compile MySQL Error Message File”

-h

Section 4.5.1.1, “mysql Options”
Section 4.2.2, “Connecting to the MySQL Server”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.6.9, “mysqlhotcopy — A Database Backup Program”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — Load Emulation Client”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.2.1, “Invoking MySQL Programs”
Section 5.1.3, “Server Command Options”
Section 1.2, “Typographical and Syntax Conventions”
Section 4.2.3.1, “Using Options on the Command Line”

HAVE EMBEDDED PRIVILEGE CONTROL

Section 22.7, “libmysqld, the Embedded MySQL Server Library”

--header

Section 17.4.19, “ndb_select_all — Print Rows from an NDB Table”

--header_file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--HELP

Section 4.6.3.1, “myisamchk General Options”

--help

Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Options”
Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 17.4.6, “ndb_config — Extract MySQL Cluster Configuration Information”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.10, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.11, “mysql_find_rows — Extract SQL Statements from Files”
Section 4.6.13, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.6.14, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.6.15, “mysql_zap — Kill Processes That Match a Pattern”
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.6.9, “mysqlhotcopy — A Database Backup Program”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — Load Emulation Client”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.4.1, “comp_err — Compile MySQL Error Message File”

Section 4.4.4, “[mysql_plugin](#) — Configure MySQL Server Plugins”
Section 4.4.7, “[mysql_upgrade](#) — Check and Upgrade MySQL Tables”
Section 4.8.1, “[perror](#) — Explain Error Codes”
Section 4.8.3, “[resolveip](#) — Resolve Host name to IP Address or Vice Versa”
Section 4.3.4, “[mysqld_multi](#) — Manage Multiple MySQL Servers”
Section 4.3.2, “[mysqld_safe](#) — MySQL Server Startup Script”
Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”
Section 4.1, “Overview of MySQL Programs”
Section 5.1.3, “Server Command Options”
Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”
Section 8.11.2, “Tuning Server Parameters”
Chapter 3, *Tutorial*
Section 2.11.1, “Unix Postinstallation Procedures”
Section 4.2.3.3, “Using Option Files”
Section 4.2.3.1, “Using Options on the Command Line”

--hex

Section 17.4.18, “[ndb_restore](#) — Restore a MySQL Cluster Backup”

--hex-blob

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

--hexdump

Section 4.6.7.1, “[mysqlbinlog](#) Hex Dump Format”
Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”

--host

Section 4.5.1.1, “[mysql](#) Options”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 5.1.9.2, “Configuring the MySQL Server to Permit IPv6 Connections”
Section 4.2.2, “Connecting to the MySQL Server”
Section 17.4.6, “[ndb_config](#) — Extract MySQL Cluster Configuration Information”
Section 4.6.10, “[mysql_convert_table_format](#) — Convert Tables to Use a Given Storage Engine”
Section 4.6.13, “[mysql_setpermission](#) — Interactively Set Permissions in Grant Tables”
Section 4.6.6, “[mysqlaccess](#) — Client for Checking Access Privileges”
Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”

Section 4.6.9, “[mysqlhotcopy](#) — A Database Backup Program”
Section 4.5.2, “[mysqladmin](#) — Client for Administering a MySQL Server”
Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”
Section 4.5.5, “[mysqlimport](#) — A Data Import Program”
Section 4.5.6, “[mysqlshow](#) — Display Database, Table, and Column Information”
Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.3.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 5.1.3, “Server Command Options”
Section 1.2, “Typographical and Syntax Conventions”
Section 5.3.4, “Using Client Programs in a Multiple-Server Environment”
Section 4.2.3.3, “Using Option Files”
Section 4.2.3.1, “Using Options on the Command Line”

--hostname

Section 17.4.22, “[ndb_size.pl](#) — NDBCLUSTER Size Requirement Estimator”

--howto

Section 4.6.6, “[mysqlaccess](#) — Client for Checking Access Privileges”

--html

Section 4.5.1.1, “[mysql](#) Options”

|

[[index top \[4343\]](#)]

-I

Section 22.8.17, “Building Client Programs”
Section 4.6.4, “[myisamlog](#) — Display MyISAM Log File Contents”
Section 4.6.14, “[mysql_waitpid](#) — Kill Process and Wait for Its Termination”
Section 4.6.15, “[mysql_zap](#) — Kill Processes That Match a Pattern”
Section 4.8.1, “[perror](#) — Explain Error Codes”
Section 4.8.2, “[replace](#) — A String-Replacement Utility”
Section 4.8.3, “[resolveip](#) — Resolve Host name to IP Address or Vice Versa”

-i

Section 4.6.3.2, “[myisamchk](#) Check Options”
Section 4.5.1.1, “[mysql](#) Options”

Section 17.5.2, “Commands in the MySQL Cluster Management Client”

Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”

Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 7.6.2, “How to Check MyISAM Tables for Errors”

Section 23.4.1.3, “Using `pdb` to create a Windows crashdump”

--i-am-a-dummy

Section 4.5.1.1, “`mysql` Options”

Section 4.5.1.6.2, “Using the `--safe-updates` Option”

--id

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

--ignore

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--ignore-builtin-innodb

Section 14.3.4, “InnoDB Startup Options and System Variables”

--ignore-lines

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--ignore-spaces

Section 4.5.1.1, “`mysql` Options”

--ignore-table

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--in_file

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

--include

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

--include-*

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--include-databases

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--include-master-host-port

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--include-tables

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--info

Section 4.8.1, “`perror` — Explain Error Codes”

Section 4.8.3, “`resolveip` — Resolve Host name to IP Address or Vice Versa”

--Information

Section 4.6.11, “`mysql_find_rows` — Extract SQL Statements from Files”

--information

Section 4.6.3.2, “`myisamchk` Check Options”

--init-command

Section 4.5.1.1, “`mysql` Options”

--init-file

Section 2.10.4, “MySQL Source-Configuration Options”

Section 21.2.3, “Performance Schema Runtime Configuration”

Section C.5.4.1.2, “Resetting the Root Password: Unix Systems”

Section C.5.4.1.1, “Resetting the Root Password: Windows Systems”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 14.6, “The MEMORY Storage Engine”

--init_connect

Section 10.1.5, “Configuring the Character Set and Collation for Applications”

--initial

Section 17.5.13.2, “Adding MySQL Cluster Data Nodes Online: Basic procedure”

Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”

Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”
Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”
Section 17.4.3, “`ndbmtd` — The MySQL Cluster Data Node Daemon (Multi-Threaded)”
Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”
Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 17.6.3, “Known Issues in MySQL Cluster Replication”
Section 17.1.6.10, “Limitations Relating to Multiple MySQL Cluster Nodes”
Section 17.3.2, “MySQL Cluster Configuration Files”
Section 17.3.3.1, “MySQL Cluster Data Node Configuration Parameters”
Section 17.5.12.3, “MySQL Cluster Disk Data Storage Requirements”
Section 17.3.3.2, “MySQL Cluster Management Node Configuration Parameters”
Section 17.3.3.3, “MySQL Cluster SQL Node and API Node Configuration Parameters”
Section 17.3.3.4, “Other MySQL Cluster Configuration Parameters”
Section 17.3.3, “Overview of MySQL Cluster Configuration Parameters”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”
Section 17.5.1, “Summary of MySQL Cluster Start Phases”

--initial-start

Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”

--innodb

Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 14.3.2, “Configuring InnoDB”

innodb-status-file

Section 14.3.14.2, “`SHOW ENGINE INNODB STATUS` and the InnoDB Monitors”

--innodb-status-file

Section 14.3.4, “InnoDB Startup Options and System Variables”

--innodb-xxx

Section 5.1.3, “Server Command Options”

--innodb_checksums

Section 14.3.4, “InnoDB Startup Options and System Variables”

--innodb_file_per_table

Section 5.1.3, “Server Command Options”
Section 14.3.3, “Using Per-Table Tablespace”
Section 8.11.3.1.3, “Using Symbolic Links for Databases on Windows”

innodb_file_per_table

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 5.1.3, “Server Command Options”

--innodb_rollback_on_timeout

Section 14.3.13, “InnoDB Error Handling”
Section 14.3.4, “InnoDB Startup Options and System Variables”

--innodb_support_xa

Section 5.2.4, “The Binary Log”

--insert-ignore

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--install

Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”
Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”
Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”
Section 5.1.3, “Server Command Options”
Section 5.3.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.7.7, “Starting MySQL as a Windows Service”

--install-manual

Section 5.1.3, “Server Command Options”

Section 5.3.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.7.7, “Starting MySQL as a Windows Service”

--interactive

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”

--iterations

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

J

[index top [4343]]

-j

Section 4.6.5, “`mysampack` — Generate Compressed, Read-Only MyISAM Tables”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--join

Section 4.6.5, “`mysampack` — Generate Compressed, Read-Only MyISAM Tables”

K

[index top [4343]]

-K

Section 4.5.4, “`mysqldump` — A Database Backup Program”

-k

Section 4.6.3.3, “`myisamchk` Repair Options”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

--keep_files_on_create

Section 13.1.17, “`CREATE TABLE` Syntax”

--keepold

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

--key_buffer_size

Section 5.1.3, “Server Command Options”

--keys

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

--keys-used

Section 4.6.3.3, “`myisamchk` Repair Options”

L

[index top [4343]]

-L

Section 4.5.1.1, “`mysql` Options”

Section 22.8.17, “Building Client Programs”

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 2.14.3, “Problems Using the Perl `DBI/DBD` Interface”

Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”

Section 10.2, “Setting the Error Message Language”

-l

Section 4.6.3.3, “`myisamchk` Repair Options”

Section 22.8.3.39, “`mysql_library_end()`”

Section 22.8.17, “Building Client Programs”

Section 22.8.9, “C API Embedded Server Function Descriptions”

Section 22.8.2, “C API Function Overview”

Section 22.7.1, “Compiling Programs with `libmysqld`”

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”

Section 17.4.2, “`ndbinfo_select_all` — Select From `ndbinfo` Tables”

Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 22.8.17.1, “Problems Linking to the MySQL Client Library”

Section 2.14.3, “Problems Using the Perl `DBI/DBD` Interface”

Section 5.1.3, “Server Command Options”

Section 5.2.3, “The General Query Log”

--language

Section 5.1.3, “Server Command Options”

Section 10.2, “Setting the Error Message Language”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

--large-pages

Section 8.11.4.2, “Enabling Large Page Support”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

--lc-messages

Section 5.1.3, “Server Command Options”

--lc-messages-dir

Section 5.1.3, “Server Command Options”

--ldata

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

--ledir

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--length

Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”

--libmysqld-libs

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

--libs

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

--libs_r

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

--line-numbers

Section 4.5.1.1, “`mysql` Options”

--lines-terminated-by

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 7.4.3, “Dumping Data in Delimited-Text Format with `mysqldump`”

--loadqueries

Section 17.4.22, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

--local

Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”

--local-infile

Section 4.5.1.1, “`mysql` Options”
Section 13.2.7, “`LOAD XML` Syntax”
Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”

--local-load

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--local-service

Section 2.3.7.7, “Starting MySQL as a Windows Service”

--lock

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

--lock-all-tables

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 1.4, “What Is New in MySQL 5.5”

--lock-tables

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”

--log

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 5.1.3, “Server Command Options”
Section 5.2.3, “The General Query Log”

--log-bin

Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”
Section 13.4.1.1, “`PURGE BINARY LOGS` Syntax”
Section 7.3.3, “Backup Strategy Summary”
Section 16.1.3.4, “Binary Log Options and Variables”
Section 19.7, “Binary Logging of Stored Programs”
Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 16.4.5, “How to Report Replication Bugs or Problems”
Section C.5.8, “Known Issues in MySQL”
Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.3, “Running Multiple MySQL Instances on One Machine”

Section 5.1.4, “Server System Variables”

Section 16.3.6, “Switching Masters During Failover”

Section 5.2.4, “The Binary Log”

Section 16.4.4, “Troubleshooting Replication”

Section 16.4.3, “Upgrading a Replication Setup”

Section 7.3.2, “Using Backups for Recovery”

--log-bin-index

Section 16.1.3.4, “Binary Log Options and Variables”

Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”

Section 5.2.4, “The Binary Log”

--log-bin-trust-function-creators

Section 16.1.3.4, “Binary Log Options and Variables”

Section 19.7, “Binary Logging of Stored Programs”

Section 1.4, “What Is New in MySQL 5.5”

--log-bin-trust-routine-creators

Section 1.4, “What Is New in MySQL 5.5”

--log-bin-use-v1-events

Section 16.1.3.4, “Binary Log Options and Variables”

--log-bin-use-v1-row-events

Section 16.1.3.4, “Binary Log Options and Variables”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 17.1.4, “MySQL Cluster Development History”

Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--log-error

Section 13.7.6.3, “`FLUSH` Syntax”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

Section 4.2.3.6, “Option Defaults, Options Expecting Values, and the = Sign”

Section 5.3, “Running Multiple MySQL Instances on One Machine”

Section 5.1.3, “Server Command Options”

Section 5.2.6, “Server Log Maintenance”

Section 2.3.7.5, “Starting MySQL from the Windows Command Line”

Section 2.3.7.4, “Starting the Server for the First Time”

Section 5.2.2, “The Error Log”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

--log-isam

Section 5.1.3, “Server Command Options”

--log-long-format

Section 5.1.3, “Server Command Options”

Section 1.4, “What Is New in MySQL 5.5”

--log-name

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”

--log-output

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”

Section 5.1.3, “Server Command Options”

Section 5.2.3, “The General Query Log”

Section 5.2.5, “The Slow Query Log”

--log-queries-not-using-indexes

Section 5.1.3, “Server Command Options”

--log-short-format

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.1.3, “Server Command Options”

Section 5.2.5, “The Slow Query Log”

--log-slave-updates

Section 16.4.5, “How to Report Replication Bugs or Problems”

Section 16.3.5, “Improving Replication Performance”

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 16.3.6, “Switching Masters During Failover”

Section 5.2.4, “The Binary Log”

log-slave-updates

Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”

--log-slow-admin-statements

Section 5.1.3, “Server Command Options”
Section 5.2.5, “The Slow Query Log”

--log-slow-queries

Section 5.1.3, “Server Command Options”
Section 5.2.5, “The Slow Query Log”

--log-slow-slave-statements

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.2.5, “The Slow Query Log”

--log-tc

Section 5.1.3, “Server Command Options”

--log-tc-size

Section 5.1.3, “Server Command Options”
Section 5.1.6, “Server Status Variables”

--log-update

Section 1.4, “What Is New in MySQL 5.5”

--log-warnings

Section C.5.2.9, “MySQL server has gone away”
Section C.5.2.11, “Communication Errors and Aborted Connections”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.1.3, “Server Command Options”
Section 5.2.2, “The Error Log”
Section 1.4, “What Is New in MySQL 5.5”

--loops

Section 17.4.13, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”
Section 17.4.2, “`ndbinfo_select_all` — Select From `ndbinfo` Tables”

--loose

Section 4.2.3.2, “Program Option Modifiers”

--loose-opt_name

Section 4.2.3.3, “Using Option Files”

--lossy-conversions

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--low-priority

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--low-priority-updates

Section 13.2.5, “INSERT Syntax”
Section 8.10.3, “Concurrent Inserts”
Section 5.1.3, “Server Command Options”
Section 8.10.2, “Table Locking Issues”

--lower-case-table-names

Section 9.2.2, “Identifier Case Sensitivity”

M

[index top [4343]]

-m

Section 4.6.3.2, “`myisamchk` Check Options”
Section 4.5.1.1, “`mysql` Options”
Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”

--malloc-lib

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--master-connect-retry

Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-data

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.3.1, “Establishing a Backup Policy”
Section 5.2, “MySQL Server Logs”

--master-host

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-info-file

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 16.2.2.2, “Slave Status Logs”

--master-password

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-port

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-retry-count

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 16.1.3.3, “Replication Slave Options and Variables”

--master-ssl

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-ssl-ca

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-ssl-capath

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-ssl-cert

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-ssl-cipher

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-ssl-key

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-user

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 5.5”

--master-xxx

Section 1.4, “What Is New in MySQL 5.5”

--max

Section 4.2.3.5, “Using Options to Set Program Variables”

--max-binlog-dump-events

Section 16.1.3.4, “Binary Log Options and Variables”

--max-binlog-size

Section 16.1.3.3, “Replication Slave Options and Variables”

--max-record-length

Section 4.6.3.3, “myisamchk Repair Options”
Section 13.7.2.5, “REPAIR TABLE Syntax”

--max-relay-log-size

Section 16.1.3.3, “Replication Slave Options and Variables”

--max-seeks-for-key

Section 8.2.1.4, “How to Avoid Full Table Scans”
Section C.5.6, “Optimizer-Related Issues”

--max_a

Section 4.2.3.5, “Using Options to Set Program Variables”

--max_join_size

Section 4.5.1.6.2, “Using the `--safe-updates` Option”

--maximum

Section 4.2.3.2, “Program Option Modifiers”

--maximum-query_cache_size

Section 4.2.3.2, “Program Option Modifiers”
Section 8.9.3.3, “Query Cache Configuration”
Section 5.1.5, “Using System Variables”

--maximum-var_name

Section 5.1.3, “Server Command Options”
Section 5.1.5, “Using System Variables”

--medium-check

Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--memlock

Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 14.3.3.1, “Using Raw Devices for the Shared Tablespace”

--method

Section 4.6.9, “[mysqlhotcopy](#) — A Database Backup Program”

--min-examined-row-limit

Section 5.1.3, “Server Command Options”

--my-plugin

Section 5.1.8.1, “Installing and Uninstalling Plugins”

--my-print-defaults

Section 4.4.4, “[mysql_plugin](#) — Configure MySQL Server Plugins”

--my_plugin

Section 5.1.8.1, “Installing and Uninstalling Plugins”

--mycnf

Section 17.4.6, “[ndb_config](#) — Extract MySQL Cluster Configuration Information”

Section 17.4.4, “[ndb_mgmd](#) — The MySQL Cluster Management Server Daemon”

--myisam-block-size

Section 8.9.2.5, “Key Cache Block Size”

Section 5.1.3, “Server Command Options”

--myisam-recover

Section 5.1.3, “Server Command Options”

--myisam-recover-options

Section 14.5.1, “[MyISAM](#) Startup Options”

Section 8.6.1, “Optimizing [MyISAM](#) Queries”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 7.6.5, “Setting Up a [MyISAM](#) Table Maintenance Schedule”

Section C.5.2.19, “Table-Corruption Issues”

Section 14.5, “The [MyISAM](#) Storage Engine”

Section 23.4.1.6, “Using Server Logs to Find Causes of Errors in [mysqld](#)”

--myisam_sort_buffer_size

Section 4.6.3.6, “[myisamchk](#) Memory Usage”

MYSQL_MAINTAINER_MODE

Section 2.10.5, “Dealing with Problems Compiling MySQL”

MYSQL_TCP_PORT

Section 2.10.3, “Installing MySQL from a Development Source Tree”

Section 2.10.4, “MySQL Source-Configuration Options”

MYSQL_UNIX_ADDR

Section C.5.4.5, “How to Protect or Change the MySQL Unix Socket File”

Section 2.10.3, “Installing MySQL from a Development Source Tree”

Section 2.10.4, “MySQL Source-Configuration Options”

--mysqldadmin

Section 4.3.4, “[mysqld_multi](#) — Manage Multiple MySQL Servers”

--mysqld

Section 4.4.4, “[mysql_plugin](#) — Configure MySQL Server Plugins”

Section 4.3.4, “[mysqld_multi](#) — Manage Multiple MySQL Servers”

Section 4.3.2, “[mysqld_safe](#) — MySQL Server Startup Script”

--mysqld-version

Section 4.3.2, “[mysqld_safe](#) — MySQL Server Startup Script”

N

[[index top \[4343\]](#)]

-N

Section 4.5.1.1, “[mysql](#) Options”

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

Section 4.4.1, “[comp_err](#) — Compile MySQL Error Message File”

-n

Section 4.6.3.3, “[myisamchk](#) Repair Options”

Section 4.5.1.1, “[mysql](#) Options”

Section 17.5.2, “Commands in the MySQL Cluster Management Client”

Section 17.4.9, “[ndb_desc](#) — Describe NDB Tables”

Section 17.4.18, “[ndb_restore](#) — Restore a MySQL Cluster Backup”

Section 17.4.23, “[ndb_waiter](#) — Wait for MySQL Cluster to Reach a Given Status”

Section 17.4.1, “[nbd](#) — The MySQL Cluster Data Node Daemon”

Section 17.4.17, “[nbd_redo_log_reader](#) — Check and Print Content of Cluster Redo Log”

Section 4.6.8, “[mysqldumpslow](#) — Summarize Slow Query Log Files”

Section 4.6.9, “[mysqlhotcopy](#) — A Database Backup Program”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”
Section 4.7.3, “[my_print_defaults](#) — Display Options from Option Files”
Section 4.7.4, “[resolve_stack_dump](#) — Resolve Numeric Stack Trace Dump to Symbols”

--name_file

Section 4.4.1, “[comp_err](#) — Compile MySQL Error Message File”

--named-commands

Section 4.5.1.1, “[mysql](#) Options”

--ndb

Section 4.8.1, “[perror](#) — Explain Error Codes”

--ndb-batch-size

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-blob-read-batch-bytes

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-blob-write-batch-bytes

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-cluster-connection-pool

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-connectstring

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 17.4.6, “[ndb_config](#) — Extract MySQL Cluster Configuration Information”
Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”
Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”
Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 17.3.2.2, “Recommended Starting Configuration for MySQL Cluster”

ndb-connectstring

Section 17.6.5, “Preparing the MySQL Cluster for Replication”

--ndb-deferred-constraints

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-distribution

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-force-send

Section 17.3.2.2, “Recommended Starting Configuration for MySQL Cluster”

--ndb-index-stat-enable

Section 17.3.2.2, “Recommended Starting Configuration for MySQL Cluster”

--ndb-log-apply-status

Section 17.3.4.3, “MySQL Cluster System Variables”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-log-empty-epochs

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-log-orig

Section 17.6.4, “MySQL Cluster Replication Schema and Tables”

--ndb-log-transaction-id

Section 16.1.3.4, “Binary Log Options and Variables”
Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-log-update-as-write

Section 17.6.3, “Known Issues in MySQL Cluster Replication”
Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”
Section 5.2.4, “The Binary Log”

--ndb-log-updated-only

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

--ndb-mgmd-host

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-nodegroup-map

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--ndb-nodeid

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”

Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-optimized-node-selection

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

ndb-transid-mysql-connection-map

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-transid-mysql-connection-map

Section 20.29.2, “The `INFORMATION_SCHEMA.ndb_transid_mysql_connection_map` Table”

--ndb-use-exact-count

Section 17.3.2.2, “Recommended Starting Configuration for MySQL Cluster”

--ndb-wait-connected

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb-wait-setup

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndb_optimization_delay

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--ndbcluster

Section 13.7.5.17, “`SHOW ENGINES` Syntax”

Section 17.2.2.1, “Installing MySQL Cluster on Windows from a Binary Release”

Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”

Section 17.3, “MySQL Cluster Configuration”

Section 17.1.1, “MySQL Cluster Core Concepts”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

Section 17.5.4, “MySQL Server Usage for MySQL Cluster”

Section 17.3.2.2, “Recommended Starting Configuration for MySQL Cluster”

Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”

--ndbinfo-database

Section 17.3.4.3, “MySQL Cluster System Variables”

--ndbinfo-table-prefix

Section 17.3.4.3, “MySQL Cluster System Variables”

net_retry_count

Section 16.2.1, “Replication Implementation Details”

net_write_timeout

Section 16.2.1, “Replication Implementation Details”

--new

Section 4.2.3.3, “Using Option Files”

--nice

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--no-auto-rehash

Section 4.5.1.1, “`mysql` Options”

--no-autocommit

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--no-beep

Section 4.5.1.1, “`mysql` Options”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

--no-binlog

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--no-contact

Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”

--no-create-db

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--no-create-info

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”

--no-data

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”

--no-defaults

Section 6.2.7, “Causes of Access-Denied Errors”

Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”

Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--no-log

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

--no-named-commands

Section 4.5.1.1, “`mysql` Options”

Section 1.4, “What Is New in MySQL 5.5”

--no-nodeid-checks

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”

--no-pager

Section 4.5.1.1, “`mysql` Options”

Section 1.4, “What Is New in MySQL 5.5”

--no-restore-disk-objects

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--no-set-names

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--no-symlinks

Section 4.6.3.3, “`myisamchk` Repair Options”

--no-tablespaces

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--no-tee

Section 4.5.1.1, “`mysql` Options”

Section 1.4, “What Is New in MySQL 5.5”

--no-upgrade

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--nodaemon

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”

Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”

--nodata

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

--nodeid

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

--nodes

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

--noindices

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

--nostart

Section 17.5.2, “Commands in the MySQL Cluster Management Client”

Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”

--not-started

Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”

--nowait-nodes

Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”
Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”

--number-char-cols

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--number-int-cols

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--number-of-queries

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--numeric-dump-file

Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”

O

[[index top \[4343\]](#)]

-O

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 1.4, “What Is New in MySQL 5.5”

-o

Section 4.6.3.3, “`myisamchk` Repair Options”
Section 4.5.1.1, “`mysql` Options”
Section 22.7.1, “Compiling Programs with `libmysqld`”
Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”
Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 8.11.3, “Optimizing Disk I/O”

--offset

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--old-alter-table

Section 5.1.3, “Server Command Options”

--old-style-user-limits

Section 5.1.3, “Server Command Options”
Section 6.3.4, “Setting Account Resource Limits”

--old_server

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

ON

Section 3.3.4.9, “Using More Than one Table”

--one-database

Section 4.5.1.1, “`mysql` Options”

--one-thread

Section 5.1.3, “Server Command Options”

--only-print

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

open-files-limit

Section C.5.2.7, “Too many connections”

--open-files-limit

Section C.5.2.18, “`'File' Not Found` and Similar Errors”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

--opt

Section 8.5.4, “Bulk Data Loading for InnoDB Tables”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--opt_name

Section 4.2.3.3, “Using Option Files”

--optimize

Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”

--order

Section 17.4.19, “[ndb_select_all](#) — Print Rows from an NDB Table”

--order-by-primary

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

--out_dir

Section 4.4.1, “[comp_err](#) — Compile MySQL Error Message File”

--out_file

Section 4.4.1, “[comp_err](#) — Compile MySQL Error Message File”

P

[[index top \[4343\]](#)]

-P

Section 4.5.1.1, “[mysql Options](#)”

Section 4.2.2, “[Connecting to the MySQL Server](#)”

Section 17.4.4, “[ndb_mgmd](#) — The MySQL Cluster Management Server Daemon”

Section 4.6.6, “[mysqlaccess](#) — Client for Checking Access Privileges”

Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”

Section 4.6.9, “[mysqlhotcopy](#) — A Database Backup Program”

Section 4.5.2, “[mysqladmin](#) — Client for Administering a MySQL Server”

Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

Section 4.5.5, “[mysqlimport](#) — A Data Import Program”

Section 4.5.6, “[mysqlshow](#) — Display Database, Table, and Column Information”

Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”

Section 4.4.4, “[mysql_plugin](#) — Configure MySQL Server Plugins”

Section 4.2.1, “[Invoking MySQL Programs](#)”

Section 5.1.3, “[Server Command Options](#)”

-P

Section 4.6.3.3, “[myisamchk Repair Options](#)”

Section 4.5.1.1, “[mysql Options](#)”

Section 18.2.5, “[KEY Partitioning](#)”

Section 17.5.13.3, “[Adding MySQL Cluster Data Nodes Online: Detailed Example](#)”

Section 6.3.2, “[Adding User Accounts](#)”

Section 6.2.7, “[Causes of Access-Denied Errors](#)”

Section 4.2.2, “[Connecting to the MySQL Server](#)”

Section 17.4.9, “[ndb_desc](#) — Describe NDB Tables”

Section 17.4.18, “[ndb_restore](#) — Restore a MySQL Cluster Backup”

Section 17.4.19, “[ndb_select_all](#) — Print Rows from an NDB Table”

Section 17.4.21, “[ndb_show_tables](#) — Display List of NDB Tables”

Section 4.6.1, “[innochecksum](#) — Offline InnoDB File Checksum Utility”

Section 4.6.4, “[myisamlog](#) — Display MyISAM Log File Contents”

Section 4.6.6, “[mysqlaccess](#) — Client for Checking Access Privileges”

Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”

Section 4.6.9, “[mysqlhotcopy](#) — A Database Backup Program”

Section 4.5.2, “[mysqladmin](#) — Client for Administering a MySQL Server”

Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

Section 4.5.5, “[mysqlimport](#) — A Data Import Program”

Section 4.5.6, “[mysqlshow](#) — Display Database, Table, and Column Information”

Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”

Section 4.4.4, “[mysql_plugin](#) — Configure MySQL Server Plugins”

Section 6.1.2.1, “[End-User Guidelines for Password Security](#)”

Section 4.2.1, “[Invoking MySQL Programs](#)”

Section C.5.2.5, “[Password Fails When Entered Interactively](#)”

Section 2.11.2, “[Securing the Initial MySQL Accounts](#)”

Section 2.3.7.7, “[Starting MySQL as a Windows Service](#)”

Section 2.3.7.5, “[Starting MySQL from the Windows Command Line](#)”

Section 2.3.7.8, “[Testing The MySQL Installation](#)”

Section 17.5.10.4, “[The ndbinfo cluster_operations Table](#)”

Section 17.5.10.15, “[The ndbinfo server_operations Table](#)”

Section 2.3.9, “[Upgrading MySQL on Windows](#)”

Section 6.3.1, “[User Names and Passwords](#)”

Section 4.2.3.1, “[Using Options on the Command Line](#)”

Section 2.3.10, “[Windows Postinstallation Procedures](#)”

--pager

Section 4.5.1.2, “mysql Commands”

Section 4.5.1.1, “mysql Options”

--parallel-recover

Section 4.6.3.3, “myisamchk Repair Options”

parallelism

Section 17.4.19, “ndb_select_all — Print Rows from an NDB Table”

--parallelism

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--parsable

Section 17.4.21, “ndb_show_tables — Display List of NDB Tables”

--partition

Section 5.1.3, “Server Command Options”

--password

Section 4.5.1.1, “mysql Options”

Section 6.3.2, “Adding User Accounts”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 4.2.2, “Connecting to the MySQL Server”

Section 4.6.10, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”

Section 4.6.13, “mysql_setpermission — Interactively Set Permissions in Grant Tables”

Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.5.5, “mysqlimport — A Data Import Program”

Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

Section 4.5.7, “mysqlslap — Load Emulation Client”

Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

Section 6.1.2.1, “End-User Guidelines for Password Security”

Section 7.3, “Example Backup and Recovery Strategy”

Section 4.2.1, “Invoking MySQL Programs”

Section C.5.2.5, “Password Fails When Entered Interactively”

Section 6.3.6, “Pluggable Authentication”

Section 6.3.6.6, “The Test Authentication Plugin”

Section 6.3.1, “User Names and Passwords”

Section 4.2.3.1, “Using Options on the Command Line”

--

performance_schema_max_mutex_classes

Section 21.5, “Performance Schema Status Monitoring”

--

performance_schema_max_mutex_instances

Section 21.5, “Performance Schema Status Monitoring”

pid-file

Section 2.11.1.2, “Starting and Stopping MySQL Automatically”

--pid-file

Section 4.3.3, “mysql.server — MySQL Server Startup Script”

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

Section 5.3, “Running Multiple MySQL Instances on One Machine”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

--pipe

Section 4.5.1.1, “mysql Options”

Section 4.2.2, “Connecting to the MySQL Server”

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.5.5, “mysqlimport — A Data Import Program”

Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

Section 4.5.7, “mysqlslap — Load Emulation Client”

Section 2.3.7.8, “Testing The MySQL Installation”

--plan

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--plugin

Section 5.1.3, “Server Command Options”

--plugin-dir

Section 4.5.1.1, “`mysql` Options”

Section 22.8.10, “C API Client Plugin Functions”

Section 23.2.4.2.3, “Client Plugin Descriptors”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

Section 6.3.6, “Pluggable Authentication”

Section E.9, “Restrictions on Pluggable Authentication”

Section 23.2.4.9.3, “Using the Authentication Plugins”

--plugin-ini

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

--plugin-

`innodb_file_per_table`

Section 5.1.3, “Server Command Options”

--plugin-load

Section 13.7.3.3, “`INSTALL PLUGIN` Syntax”

Section 6.3.10.4, “Audit Log Plugin Options and Variables”

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 5.1.8.1, “Installing and Uninstalling Plugins”

Section 6.3.10.1, “Installing the Audit Log Plugin”

Section 6.3.6.2.1, “Installing the PAM Authentication Plugin”

Section 6.3.6.3.1, “Installing the Windows Authentication Plugin”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 6.3.6, “Pluggable Authentication”

Section 23.2.2, “Plugin API Components”

Section 23.2.4.2, “Plugin Data Structures”

Section 5.1.3, “Server Command Options”

Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”

Section 23.2, “The MySQL Plugin API”

Section 8.11.6.1, “Thread Pool Components and Installation”

Section 23.2.4.9.3, “Using the Authentication Plugins”

--plugin-sql-mode

Section 5.1.3, “Server Command Options”

--plugin-xxx

Section 5.1.3, “Server Command Options”

--plugin_dir

Section 2.10.4, “MySQL Source-Configuration Options”

Section 23.2.2, “Plugin API Components”

--plugin_name

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 5.1.8.1, “Installing and Uninstalling Plugins”

Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

--plugindir

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

--port

Section 4.5.1.1, “`mysql` Options”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 4.2.2, “Connecting to the MySQL Server”

Section 4.6.10, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”

Section 4.6.13, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

Section 4.2.1, “Invoking MySQL Programs”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 5.3, “Running Multiple MySQL Instances on One Machine”

Section 5.3.3, “Running Multiple MySQL Instances on Unix”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”

Section 5.3.4, “Using Client Programs in a Multiple-Server Environment”

--port-open-timeout

Section 5.1.3, “Server Command Options”

--position

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 1.4, “What Is New in MySQL 5.5”

--post-query

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--post-system

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--pre-query

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

--pre-system

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

PREFIX

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”

--preserve-trailing-spaces

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--preview

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--print

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--print-defaults

Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”

Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

Section 2.12.1, “Upgrading MySQL”

--print-full-config

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”

--print_*

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--print_data

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--print_log

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--print_meta

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--progress-frequency

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--promote-attributes

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--prompt

Section 4.5.1.2, “`mysql` Commands”

Section 4.5.1.1, “`mysql` Options”

--protocol

Section 4.5.1.1, “`mysql` Options”

Section 4.2.2, “Connecting to the MySQL Server”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “[mysqlshow](#) — Display Database, Table, and Column Information”
Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”
Section 5.3.3, “[Running Multiple MySQL Instances on Unix](#)”
Section 2.3.7.4, “[Starting the Server for the First Time](#)”
Section 2.3.7.8, “[Testing The MySQL Installation](#)”
Section 5.3.4, “[Using Client Programs in a Multiple-Server Environment](#)”

Q

[[index top \[4343\]](#)]

-Q

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

-q

Section 4.6.3.3, “[myisamchk Repair Options](#)”
Section 4.5.1.1, “[mysql Options](#)”
Section 17.4.6, “[ndb_config](#) — Extract MySQL Cluster Configuration Information”
Section 4.6.9, “[mysqlhotcopy](#) — A Database Backup Program”
Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”
Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”

--query

Section 17.4.6, “[ndb_config](#) — Extract MySQL Cluster Configuration Information”
Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”
Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”

--query-cache-size

Section 8.10.5, “[External Locking](#)”

--quick

Section 4.6.3.6, “[myisamchk Memory Usage](#)”
Section 4.6.3.3, “[myisamchk Repair Options](#)”
Section 4.5.1.1, “[mysql Options](#)”
Section C.5.2.8, “[Out of memory](#)”
Section 4.5.1, “[mysql](#) — The MySQL Command-Line Tool”
Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”
Section 7.6.1, “[Using myisamchk for Crash Recovery](#)”

Section 4.2.3.3, “[Using Option Files](#)”

--quiet

Section 4.6.9, “[mysqlhotcopy](#) — A Database Backup Program”

--quote-names

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

R

[[index top \[4343\]](#)]

-R

Section 7.6.4, “[MyISAM Table Optimization](#)”
Section 17.4.18, “[ndb_restore](#) — Restore a MySQL Cluster Backup”
Section 4.6.4, “[myisamlog](#) — Display MyISAM Log File Contents”
Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”
Section 4.6.3.4, “[Other myisamchk Options](#)”

-r

Section 4.6.3.2, “[myisamchk Check Options](#)”
Section 4.6.3.3, “[myisamchk Repair Options](#)”
Section 4.5.1.1, “[mysql Options](#)”
Section 23.3.2, “[Adding a New User-Defined Function](#)”
Section 17.4.6, “[ndb_config](#) — Extract MySQL Cluster Configuration Information”
Section 4.6.4, “[myisamlog](#) — Display MyISAM Log File Contents”
Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”
Section 4.6.8, “[mysqldumpslow](#) — Summarize Slow Query Log Files”
Section 4.5.2, “[mysqladmin](#) — Client for Administering a MySQL Server”
Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”
Section 4.5.5, “[mysqlimport](#) — A Data Import Program”
Section 7.6.3, “[How to Repair MyISAM Tables](#)”
Section 2.2, “[Installing MySQL from Generic Binaries on Unix/Linux](#)”
Section 5.1.3, “[Server Command Options](#)”

--raw

Section 4.5.1.1, “[mysql Options](#)”

--read-from-remote-server

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--read-only

Section 4.6.3.2, “`myisamchk` Check Options”
Section 16.1.3.3, “Replication Slave Options and Variables”

--rebuild-indexes

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--reconnect

Section 4.5.1.1, “`mysql` Options”

--record_log_pos

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

--recover

Section 4.6.3.2, “`myisamchk` Check Options”
Section 4.6.3.1, “`myisamchk` General Options”
Section 4.6.3.6, “`myisamchk` Memory Usage”
Section 4.6.3.3, “`myisamchk` Repair Options”

--regexp

Section 4.6.11, “`mysql_find_rows` — Extract SQL Statements from Files”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

--relative

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

--relay-log

Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”
Section 16.3.5, “Improving Replication Performance”
Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”
Section 4.2.3.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 16.2.2.1, “The Slave Relay Log”

--relay-log-index

Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 16.2.2.1, “The Slave Relay Log”

--relay-log-info-file

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 16.2.2.2, “Slave Status Logs”

--relay-log-purge

Section 16.1.3.3, “Replication Slave Options and Variables”

--relay-log-recovery

Section 16.1.3.3, “Replication Slave Options and Variables”

--relay-log-space-limit

Section 16.1.3.3, “Replication Slave Options and Variables”

--relnotes

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--reload

Section 17.5.13.2, “Adding MySQL Cluster Data Nodes Online: Basic procedure”
Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”
Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 17.1.6.10, “Limitations Relating to Multiple MySQL Cluster Nodes”
Section 17.3.2, “MySQL Cluster Configuration Files”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”

--remove

Section 17.4.4, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”
Section 17.2.2.4, “Installing MySQL Cluster Processes as Windows Services”
Section 5.1.3, “Server Command Options”
Section 5.3.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.7.7, “Starting MySQL as a Windows Service”

--repair

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--replace

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--replicate-*

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

Section 16.2.3.3, “Replication Rule Application”

Section 16.1.3.3, “Replication Slave Options and Variables”

--replicate-*-db

Section 16.2.3.3, “Replication Rule Application”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section E.1, “Restrictions on Stored Programs”

--replicate-*-table

Section 16.2.3.3, “Replication Rule Application”

--replicate-do-*

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

--replicate-do-db

Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”

Section 13.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”

Section 16.1.3.4, “Binary Log Options and Variables”

Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 16.3.4, “Replicating Different Databases to Different Slaves”

Section 16.4.1.24, “Replication and Reserved Words”

Section 16.4.1.21, “Replication and Temporary Tables”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.2.4, “The Binary Log”

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-do-table

Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”

Section 16.2.3.2, “Evaluation of Table-Level Replication Options”

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 16.4.1.24, “Replication and Reserved Words”

Section 16.4.1.21, “Replication and Temporary Tables”

Section 16.2.3.3, “Replication Rule Application”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 14.9, “The `BLACKHOLE` Storage Engine”

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-ignore-*

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

--replicate-ignore-db

Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”

Section 13.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”

Section 16.1.3.4, “Binary Log Options and Variables”

Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 16.4.1.24, “Replication and Reserved Words”

Section 16.2.3.3, “Replication Rule Application”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.2.4, “The Binary Log”

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-ignore-table

Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”

Section 16.2.3.2, “Evaluation of Table-Level Replication Options”

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 16.4.1.24, “Replication and Reserved Words”

Section 16.4.1.21, “Replication and Temporary Tables”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 14.9, “The `BLACKHOLE` Storage Engine”

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-rewrite-db

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-same-server-id

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 16.1.3.3, “Replication Slave Options and Variables”

--replicate-wild-do-table

Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
Section 16.2.3.2, “Evaluation of Table-Level Replication Options”
Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 16.3.4, “Replicating Different Databases to Different Slaves”
Section 16.4.1.21, “Replication and Temporary Tables”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section E.1, “Restrictions on Stored Programs”

--replicate-wild-ignore-table

Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
Section 16.2.3.2, “Evaluation of Table-Level Replication Options”
Section 17.6.3, “Known Issues in MySQL Cluster Replication”
Section 16.4.1.21, “Replication and Temporary Tables”
Section 16.1.3.3, “Replication Slave Options and Variables”

replication-ignore-table

Section 16.4.1.33, “Replication and Views”

--report-host

Section 13.7.5.34, “SHOW SLAVE HOSTS Syntax”
Section 16.1.4.1, “Checking Replication Status”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.1.4, “Server System Variables”

--report-password

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.1.4, “Server System Variables”

--report-port

Section 13.7.5.34, “SHOW SLAVE HOSTS Syntax”
Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.1.4, “Server System Variables”

--report-user

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.1.4, “Server System Variables”

--resetmaster

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

--resetslave

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

--restore-epoch

Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”

--restore-privilege-tables

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”
Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”

--restore_data

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--restore_epoch

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”
Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”

--restore_meta

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--result-file

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”

--rewrite-database

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

--rhost

Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”

--rollback

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--routines

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 7.4.5.3, “Dumping Stored Programs”

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

--rowid

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

--rows

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

Section 4.6.11, “`mysql_find_rows` — Extract SQL Statements from Files”

--rpm

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

S

[index top [4343]]

-s

Section 4.5.1.2, “`mysql` Commands”

Section 4.5.1.1, “`mysql` Options”

Section 7.6.4, “`MyISAM` Table Optimization”

Section 4.2.2, “Connecting to the MySQL Server”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

Section 4.2.1, “Invoking MySQL Programs”

Section 4.6.3.4, “Other `myisamchk` Options”

-s

Section 4.6.3.1, “`myisamchk` General Options”

Section 4.5.1.1, “`mysql` Options”

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”

Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”

Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

Section 4.6.15, “`mysql_zap` — Kill Processes That Match a Pattern”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

Section 4.8.1, “`perror` — Explain Error Codes”

Section 4.8.2, “`replace` — A String-Replacement Utility”

Section 4.8.3, “`resolveip` — Resolve Host name to IP Address or Vice Versa”

Section 7.6.2, “How to Check `MyISAM` Tables for Errors”

Section 7.6.3, “How to Repair `MyISAM` Tables”

Section 5.1.3, “Server Command Options”

Section 7.6.5, “Setting Up a `MyISAM` Table Maintenance Schedule”

--safe-mode

Section 5.1.3, “Server Command Options”

--safe-recover

Section 4.6.3.1, “`myisamchk` General Options”

Section 4.6.3.6, “`myisamchk` Memory Usage”

Section 4.6.3.3, “`myisamchk` Repair Options”

--safe-show-database

Section 5.1.3, “Server Command Options”

Section 1.4, “What Is New in MySQL 5.5”

--safe-updates

Section 4.5.1.2, “`mysql` Commands”

Section 4.5.1.1, “mysql Options”
Section 4.5.1.6.2, “Using the `--safe-updates` Option”

--safe-user-create

Section 5.1.3, “Server Command Options”

--savequeries

Section 17.4.22, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

--secure-auth

Section 4.5.1.1, “mysql Options”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 5.1.3, “Server Command Options”

--secure-file-priv

Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 2.11.1, “Unix Postinstallation Procedures”

--select_limit

Section 4.5.1.6.2, “Using the `--safe-updates` Option”

server-id

Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”
Section 16.1, “Replication Configuration”
Section 16.1.3.2, “Replication Master Options and Variables”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 16.1.1.1, “Setting the Replication Master Configuration”
Section 16.1.1.2, “Setting the Replication Slave Configuration”
Section 16.1.1.8, “Setting Up Replication with Existing Data”

--server-id

Section 13.7.5.34, “`SHOW SLAVE HOSTS` Syntax”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 17.6.2, “General Requirements for MySQL Cluster Replication”
Section 16.1.3, “Replication and Binary Logging Options and Variables”
Section 5.1.4, “Server System Variables”
Section 16.4.4, “Troubleshooting Replication”

--server-id-bits

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 17.3.4.3, “MySQL Cluster System Variables”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--service-startup-timeout

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

--set-auto-increment

Section 4.6.3.4, “Other `myisamchk` Options”

--set-character-set

Section 4.6.3.3, “`myisamchk` Repair Options”

--set-charset

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--set-collation

Section 4.6.3.3, “`myisamchk` Repair Options”

--set-variable

Section 4.5.1.1, “mysql Options”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.2.3.5, “Using Options to Set Program Variables”
Section 1.4, “What Is New in MySQL 5.5”

set-variable

Section 4.2.3.5, “Using Options to Set Program Variables”

--shared-memory

Section 4.2.2, “Connecting to the MySQL Server”
Section 5.1.3, “Server Command Options”
Section 5.3.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.3.7.4, “Starting the Server for the First Time”

--shared-memory-base-name

Section 22.8.3.49, “`mysql_options()`”
Section 4.2.2, “Connecting to the MySQL Server”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”
Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 5.1.3, “Server Command Options”
Section 5.3.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 5.3.4, “Using Client Programs in a Multiple-Server Environment”

--short-form

Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”

--show-slave-auth-info

Section 16.1.3.3, “Replication Slave Options and Variables”

--show-table-type

Section 4.5.6, “[mysqlshow](#) — Display Database, Table, and Column Information”

--show-temp-status

Section 17.4.21, “[ndb_show_tables](#) — Display List of NDB Tables”

--show-warnings

Section 4.5.1.1, “[mysql](#) Options”

--sigint-ignore

Section 4.5.1.1, “[mysql](#) Options”

--silent

Section 4.6.3.1, “[myisamchk](#) General Options”

Section 4.5.1.1, “[mysql](#) Options”

Section 4.6.5, “[myisampack](#) — Generate Compressed, Read-Only MyISAM Tables”

Section 4.5.2, “[mysqladmin](#) — Client for Administering a MySQL Server”

Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”

Section 4.5.5, “[mysqlimport](#) — A Data Import Program”

Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”

Section 4.8.1, “[perror](#) — Explain Error Codes”

Section 4.8.3, “[resolveip](#) — Resolve Host name to IP Address or Vice Versa”

Section 4.3.4, “[mysqld_multi](#) — Manage Multiple MySQL Servers”

Section 7.6.5, “Setting Up a [MyISAM](#) Table Maintenance Schedule”

--single-transaction

Section 14.3.7, “Backing Up and Recovering an [InnoDB](#) Database”

Section 7.2, “Database Backup Methods”

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

Section 7.3.1, “Establishing a Backup Policy”

--single-user

Section 17.4.23, “[ndb_waiter](#) — Wait for MySQL Cluster to Reach a Given Status”

--skip

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

Section 4.2.3.2, “Program Option Modifiers”

Section 5.1.3, “Server Command Options”

--skip-add-drop-table

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

--skip-add-locks

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

--skip-auto-rehash

Section 14.3.14.4, “Troubleshooting [InnoDB](#) Data Dictionary Operations”

--skip-broken-objects

Section 17.4.18, “[ndb_restore](#) — Restore a MySQL Cluster Backup”

--skip-character-set-client-handshake

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 10.1.14.7.1, “The [cp932](#) Character Set”

--skip-column-names

Section 4.5.1.1, “[mysql](#) Options”

--skip-comments

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

--skip-compact

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

--skip-concurrent-insert

Section 5.1.3, “Server Command Options”

--skip-config-cache

Section 17.4.4, “[ndb_mgmd](#) — The MySQL Cluster Management Server Daemon”

--skip-disable-keys

Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

--skip-dump-date

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-engine_name

Section 13.7.5.17, “SHOW ENGINES Syntax”

--skip-event-scheduler

Section 5.1.3, “Server Command Options”

--skip-events

Section 7.4.5.3, “Dumping Stored Programs”

--skip-extended-insert

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-external-locking

Section 8.10.5, “External Locking”
Section 8.12.5.2, “General Thread States”
Section 5.1.3, “Server Command Options”
Section 8.11.1, “System Factors and Startup Parameter Tuning”
Section 1.4, “What Is New in MySQL 5.5”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”

--skip-grant-tables

Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 13.7.3.3, “INSTALL PLUGIN Syntax”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 23.3.2.5, “Compiling and Installing User-Defined Functions”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”
Section 19.4.2, “Event Scheduler Configuration”
Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 6.3.6, “Pluggable Authentication”
Section 2.11.1.1, “Problems Running mysql_install_db”
Section C.5.4.1.3, “Resetting the Root Password: Generic Instructions”
Section 5.1.3, “Server Command Options”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”
Section 4.2.3.1, “Using Options on the Command Line”
Section 6.2.6, “When Privilege Changes Take Effect”

--skip-host-cache

Section 6.2.7, “Causes of Access-Denied Errors”
Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”
Section 5.1.3, “Server Command Options”

--skip-innodb

Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 14.3.2, “Configuring InnoDB”
Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”

--skip-innodb-checksums

Section 14.3.4, “InnoDB Startup Options and System Variables”

--skip-

innodb_adaptive_hash_index

Section 14.3.4, “InnoDB Startup Options and System Variables”

--skip-innodb_checksums

Section 14.3.4, “InnoDB Startup Options and System Variables”

--skip-innodb_doublewrite

Section 14.3.4, “InnoDB Startup Options and System Variables”

--skip-kill-mysqld

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--skip-line-numbers

Section 4.5.1.1, “mysql Options”

--skip-lock-tables

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-locking

Section 1.4, “What Is New in MySQL 5.5”

--skip-log-warnings

Section 16.1.3.3, “Replication Slave Options and Variables”

--skip-name-resolve

Section 6.2.7, “Causes of Access-Denied Errors”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”
Section 2.11.2, “Securing the Initial MySQL Accounts”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 2.3.7.8, “Testing The MySQL Installation”

--skip-named-commands

Section 4.5.1.1, “`mysql` Options”
Section 1.4, “What Is New in MySQL 5.5”

--skip-ndbcluster

Section 17.3.4.3, “MySQL Cluster System Variables”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

--skip-networking

Section C.5.2.2, “Can't connect to [local] MySQL server”
Section C.5.2.9, “MySQL server has gone away”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”
Section 6.3.6, “Pluggable Authentication”
Section C.5.4.1.3, “Resetting the Root Password: Generic Instructions”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 16.4.3, “Upgrading a Replication Setup”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

skip-networking

Section 16.1.1.1, “Setting the Replication Master Configuration”
Section 16.4.4, “Troubleshooting Replication”

--skip-new

Section 13.7.2.4, “`OPTIMIZE TABLE` Syntax”
Section 23.4.1, “Debugging a MySQL Server”
Section 5.1.4, “Server System Variables”

--skip-opt

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-pager

Section 4.5.1.1, “`mysql` Options”

Section 1.4, “What Is New in MySQL 5.5”

--skip-partition

Chapter 18, *Partitioning*
Section 5.1.3, “Server Command Options”

--skip-plugin-innodb_file_per_table

Section 5.1.3, “Server Command Options”

--skip-plugin_name

Section 5.1.8.1, “Installing and Uninstalling Plugins”

--skip-quick

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-quote-names

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-reconnect

Section 4.5.1.1, “`mysql` Options”
Section 22.8.12, “Controlling Automatic Reconnection Behavior”
Section 4.5.1.6.3, “Disabling `mysql` Auto-Reconnect”

--skip-routines

Section 7.4.5.3, “Dumping Stored Programs”

--skip-safemalloc

Section 5.1.3, “Server Command Options”

--skip-set-charset

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-show-database

Section 13.7.5.15, “`SHOW DATABASES` Syntax”
Section 6.2.1, “Privileges Provided by MySQL”
Section 5.1.3, “Server Command Options”

--skip-slave-start

Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”
Section 13.4.2.5, “`START SLAVE` Syntax”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 16.3.7, “Setting Up Replication Using SSL”
Section 16.1.1.8, “Setting Up Replication with Existing Data”
Section 16.4.4, “Troubleshooting Replication”

Section 16.4.3, “Upgrading a Replication Setup”

skip-slave-start

Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”

Section 17.6.5, “Preparing the MySQL Cluster for Replication”

Section 17.6.6, “Starting MySQL Cluster Replication (Single Replication Channel)”

--skip-ssl

Section 6.3.8.4, “SSL Command Options”

--skip-stack-trace

Section 23.4.1.4, “Debugging `mysqld` under `gdb`”

Section 5.1.3, “Server Command Options”

--skip-super-large-pages

Section 8.11.4.2, “Enabling Large Page Support”

Section 1.4.4, “Enhanced Solaris Support”

Section 5.1.3, “Server Command Options”

--skip-symbolic-links

Section 13.1.17, “`CREATE TABLE` Syntax”

Section 6.1.3, “Making MySQL Secure Against Attackers”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 8.11.3.1.2, “Using Symbolic Links for `MyISAM` Tables on Unix”

Section 8.11.3.1.3, “Using Symbolic Links for Databases on Windows”

Section 1.4, “What Is New in MySQL 5.5”

--skip-symlink

Section 1.4, “What Is New in MySQL 5.5”

--skip-syslog

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

Section 5.2.2, “The Error Log”

--skip-table-check

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--skip-tee

Section 4.5.1.1, “`mysql` Options”

Section 1.4, “What Is New in MySQL 5.5”

--skip-thread-priority

Section 5.1.3, “Server Command Options”

--skip-triggers

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 7.4.5.3, “Dumping Stored Programs”

--skip-tz-utc

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-unknown-objects

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

--skip-use-db

Section 4.6.11, “`mysql_find_rows` — Extract SQL Statements from Files”

--skip-write-binlog

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

--skip-xxx

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip_grant_tables

Section 4.2.3.1, “Using Options on the Command Line”

--slave-load-tmpdir

Section 16.3.1.2, “Backing Up Raw Data from a Slave”

Section 7.2, “Database Backup Methods”

Section 16.1.3.3, “Replication Slave Options and Variables”

slave-max-allowed-packet

Section 16.1.3.2, “Replication Master Options and Variables”

--slave-max-allowed-packet

Section 16.1.3.2, “Replication Master Options and Variables”

--slave-net-timeout

Section 16.1.3.3, “Replication Slave Options and Variables”

--slave-skip-errors

Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 16.4.1.26, “Slave Errors During Replication”

--slave_compressed_protocol

Section 16.1.3.3, “Replication Slave Options and Variables”

--sleep

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

--slow-query-log

Section 5.1.3, “Server Command Options”

--slow-start-timeout

Section 5.1.3, “Server Command Options”

--slow_query_log

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 5.2.5, “The Slow Query Log”

--slow_query_log_file

Section 5.3, “Running Multiple MySQL Instances on One Machine”

Section 5.1.3, “Server Command Options”

Section 5.2.5, “The Slow Query Log”

--socket

Section 4.5.1.1, “`mysql` Options”

Section C.5.2.2, “Can't connect to [local] MySQL server”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 4.2.2, “Connecting to the MySQL Server”

Section 4.6.10, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”

Section 4.6.13, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

Section C.5.4.5, “How to Protect or Change the MySQL Unix Socket File”

Section 4.2.1, “Invoking MySQL Programs”

Section 2.10.4, “MySQL Source-Configuration Options”

Section 5.3, “Running Multiple MySQL Instances on One Machine”

Section 5.3.3, “Running Multiple MySQL Instances on Unix”

Section 5.1.3, “Server Command Options”

Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”

Section 2.3.7.8, “Testing The MySQL Installation”

Section 5.3.4, “Using Client Programs in a Multiple-Server Environment”

socket

Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”

--sort-index

Section 7.6.4, “`MyISAM` Table Optimization”

Section 4.6.3.4, “Other `myisamchk` Options”

--sort-records

Section 7.6.4, “`MyISAM` Table Optimization”

Section 4.6.3.4, “Other `myisamchk` Options”

--sort-recover

Section 4.6.3.1, “`myisamchk` General Options”

Section 4.6.3.6, “`myisamchk` Memory Usage”

Section 4.6.3.3, “`myisamchk` Repair Options”

--spassword

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--sporadic-binlog-dump-fail

Section 16.1.3.4, “Binary Log Options and Variables”

--sql-bin-update-same

Section 1.4, “What Is New in MySQL 5.5”

--sql-mode

Chapter 12, *Functions and Operators*

Section 1.8.2, “Selecting SQL Modes”
Section 5.1.3, “Server Command Options”
Section 5.1.7, “Server SQL Modes”

sql-mode

Section 5.1.7, “Server SQL Modes”

--srcdir

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

--ssl

Section 4.5.1.1, “`mysql` Options”
Section 4.2.2, “Connecting to the MySQL Server”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”
Section 5.1.3, “Server Command Options”
Section 6.3.8.4, “SSL Command Options”

--ssl*

Section 4.5.1.1, “`mysql` Options”
Section 4.2.2, “Connecting to the MySQL Server”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”
Section 5.1.3, “Server Command Options”

--ssl-ca

Section 13.7.1.3, “`GRANT` Syntax”
Section 6.3.8.5, “Setting Up SSL Certificates and Keys for MySQL”
Section 6.3.8.4, “SSL Command Options”
Section 6.3.8.3, “Using SSL Connections”

--ssl-capath

Section 13.7.1.3, “`GRANT` Syntax”
Section 6.3.8.4, “SSL Command Options”

--ssl-cert

Section 13.7.1.3, “`GRANT` Syntax”
Section 6.3.8.5, “Setting Up SSL Certificates and Keys for MySQL”
Section 6.3.8.4, “SSL Command Options”
Section 6.3.8.3, “Using SSL Connections”

--ssl-cipher

Section 6.3.8.4, “SSL Command Options”

--ssl-key

Section 13.7.1.3, “`GRANT` Syntax”
Section 6.3.8.5, “Setting Up SSL Certificates and Keys for MySQL”
Section 6.3.8.4, “SSL Command Options”
Section 6.3.8.3, “Using SSL Connections”

--ssl-verify-server-cert

Section 6.3.8.4, “SSL Command Options”

--ssl-xxx

Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”
Section 6.3.8.2, “Configuring MySQL for SSL”
Section 5.1.4, “Server System Variables”
Section 6.3.8.4, “SSL Command Options”

--standalone

Section 23.4.1.2, “Creating Trace Files”
Section 5.1.3, “Server Command Options”
Section 2.3.7.5, “Starting MySQL from the Windows Command Line”

--start-datetime

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”

--start-position

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”
Section 1.4, “What Is New in MySQL 5.5”

--start_row

Section 4.6.11, “`mysql_find_rows` — Extract SQL Statements from Files”

--statefile

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

--stats

Section 4.6.2, “[mysam_ftdump](#) — Display Full-Text Index information”

--status

Section 4.5.6, “[mysqlshow](#) — Display Database, Table, and Column Information”

--stop-datetime

Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”

Section 7.5.1, “Point-in-Time Recovery Using Event Times”

--stop-position

Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”

Section 7.5.2, “Point-in-Time Recovery Using Event Positions”

--suffix

Section 4.6.9, “[mysqlhotcopy](#) — A Database Backup Program”

--super-large-pages

Section 8.11.4.2, “Enabling Large Page Support”

Section 1.4.4, “Enhanced Solaris Support”

Section 5.1.3, “Server Command Options”

--superuser

Section 4.6.6, “[mysqlaccess](#) — Client for Checking Access Privileges”

--symbolic-links

Section 5.1.3, “Server Command Options”

Section 1.4, “What Is New in MySQL 5.5”

--symbols-file

Section 4.7.4, “[resolve_stack_dump](#) — Resolve Numeric Stack Trace Dump to Symbols”

--sys-*

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

--sys-check

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

--sys-create

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

sys-create-if-not-exist

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

--sys-create-if-not-valid

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

--sys-drop

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

--sys-skip-events

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

--sys-skip-tables

Section 17.4.13, “[ndb_index_stat](#) — NDB Index Statistics Utility”

SYSCONFDIR

Section 4.2.3.3, “Using Option Files”

--sysdate-is-now

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”

Section 12.7, “Date and Time Functions”

Section 16.4.1.13, “Replication and System Functions”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

--syslog

Section 4.3.2, “[mysqld_safe](#) — MySQL Server Startup Script”

Section 5.2.2, “The Error Log”

--syslog-tag

Section 4.3.2, “[mysqld_safe](#) — MySQL Server Startup Script”

--system

Section 17.4.6, “[ndb_config](#) — Extract MySQL Cluster Configuration Information”

T

[[index top \[4343\]](#)]

-T

Section 4.6.3.2, “[myisamchk](#) Check Options”

Section 4.5.1.1, “[mysql](#) Options”

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 5.1.3, “Server Command Options”

-t

Section 4.6.3.3, “`myisamchk` Repair Options”
Section 4.5.1.1, “`mysql` Options”
Section 17.4.8, “`ndb_delete_all` — Delete All Rows from an NDB Table”
Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”
Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”
Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.15, “`mysql_zap` — Kill Processes That Match a Pattern”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 2.3.6.13, “MySQL Server Instance Config Wizard: Creating an Instance from the Command Line”
Section 5.1.3, “Server Command Options”

--tab

Section 7.1, “Backup and Recovery Types”
Section 7.2, “Database Backup Methods”
Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.3, “Dumping Data in Delimited-Text Format with `mysqldump`”

Section 7.4, “Using `mysqldump` for Backups”

--table

Section 4.5.1.1, “`mysql` Options”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--table_open_cache

Section 8.4.3.1, “How MySQL Opens and Closes Tables”

--tables

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--tc-heuristic-recover

Section 5.1.3, “Server Command Options”

--tcp-ip

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

--tee

Section 4.5.1.2, “`mysql` Commands”
Section 4.5.1.1, “`mysql` Options”

--temp-pool

Section 5.1.3, “Server Command Options”

--test

Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

Text

Section 1.2, “Typographical and Syntax Conventions”

--thread_cache_size

Section 23.4.1.4, “Debugging `mysqld` under `gdb`”

--thread_handling

Section 5.1.3, “Server Command Options”

--thread_stack

Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”

--timeout

Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”

--timezone

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 10.6, “MySQL Server Time Zone Support”
Section 16.4.1.30, “Replication and Time Zones”
Section 5.1.4, “Server System Variables”
Section C.5.4.6, “Time Zone Problems”

tmpdir

Section 2.3, “Installing MySQL on Microsoft Windows”

--tmpdir

Section 4.6.3.6, “`myisamchk` Memory Usage”
Section 4.6.3.3, “`myisamchk` Repair Options”
Section C.5.2.13, “Can't create/write to file”
Section 4.6.5, “`mysampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 5.3, “Running Multiple MySQL Instances on One Machine”
Section 5.1.3, “Server Command Options”
Section C.5.4.4, “Where MySQL Stores Temporary Files”

--to-last-log

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--transaction-isolation

Section 13.3.6, “`SET TRANSACTION` Syntax”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 14.3.9, “The InnoDB Transaction Model and Locking”

--transactional

Section 17.4.8, “`ndb_delete_all` — Delete All Rows from an NDB Table”

--triggers

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.5.3, “Dumping Stored Programs”

--try-reconnect

Section 17.4.5, “`ndb_mgm` — The MySQL Cluster Management Client”

--tupscan

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

--type

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”
Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”
Section 4.6.10, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”

--tz-utc

Section 4.5.4, “`mysqldump` — A Database Backup Program”

U

[index top [4343]]

-U

Section 4.6.3.2, “`myisamchk` Check Options”
Section 4.5.1.1, “`mysql` Options”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

-u

Section 4.6.3.3, “`myisamchk` Repair Options”
Section 4.5.1.1, “`mysql` Options”
Section 4.2.2, “Connecting to the MySQL Server”
Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”
Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

Section 4.2.1, “Invoking MySQL Programs”

Section 5.1.3, “Server Command Options”

Section 2.3.7.8, “Testing The MySQL Installation”

Section 6.3.1, “User Names and Passwords”

Section 2.3.10, “Windows Postinstallation Procedures”

--unbuffered

Section 4.5.1.1, “`mysql` Options”

--unpack

Section 4.6.3.3, “`myisamchk` Repair Options”

Section 14.5.3, “MyISAM Table Storage Formats”

Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

--unqualified

Section 17.4.21, “`ndb_show_tables` — Display List of NDB Tables”

--update

Section 17.4.13, “`ndb_index_stat` — NDB Index Statistics Utility”

--update-state

Section 4.6.3.2, “`myisamchk` Check Options”

Section 7.6.3, “How to Repair MyISAM Tables”

Section 14.5, “The MyISAM Storage Engine”

--upgrade-system-tables

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

--usage

Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”

--use-frm

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--use-mysqld_safe

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

--use-symbolic-links

Section 1.4, “What Is New in MySQL 5.5”

--use-threads

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--useHexFormat

Section 17.4.19, “`ndb_select_all` — Print Rows from an NDB Table”

--user

Section 4.5.1.1, “`mysql` Options”

Section C.5.2.18, “`'File' Not Found` and Similar Errors”

Section 4.2.2, “Connecting to the MySQL Server”

Section 4.6.10, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”

Section 4.6.13, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.7, “`mysqlslap` — Load Emulation Client”

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

Section 7.3, “Example Backup and Recovery Strategy”

Section 6.1.5, “How to Run MySQL as a Normal User”

Section 4.2.1, “Invoking MySQL Programs”

Section 6.1.3, “Making MySQL Secure Against Attackers”

Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”

Section 4.2.3.6, “Option Defaults, Options Expecting Values, and the = Sign”

Section 6.3.6, “Pluggable Authentication”

Section C.5.4.1.2, “Resetting the Root Password: Unix Systems”

Section 5.1.3, “Server Command Options”
Section 6.3.6.5, “The Socket Peer-Credential Authentication Plugin”
Section 6.3.6.6, “The Test Authentication Plugin”
Section 2.11.1, “Unix Postinstallation Procedures”
Section 6.3.1, “User Names and Passwords”
Section 4.2.3.3, “Using Option Files”

V

[[index top \[4343\]](#)]

-v

Section 4.6.3.1, “[myisamchk](#) General Options”
Section 4.5.1.1, “[mysql](#) Options”
Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 17.4.6, “[ndb_config](#) — Extract MySQL Cluster Configuration Information”
Section 4.6.4, “[myisamlog](#) — Display MyISAM Log File Contents”
Section 4.6.5, “[myisampack](#) — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.14, “[mysql_waitpid](#) — Kill Process and Wait for Its Termination”
Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”
Section 4.5.2, “[mysqladmin](#) — Client for Administering a MySQL Server”
Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”
Section 4.5.5, “[mysqlimport](#) — A Data Import Program”
Section 4.5.6, “[mysqlshow](#) — Display Database, Table, and Column Information”
Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”
Section 4.7.3, “[my_print_defaults](#) — Display Options from Option Files”
Section 4.7.4, “[resolve_stack_dump](#) — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.4.1, “[comp_err](#) — Compile MySQL Error Message File”
Section 4.4.4, “[mysql_plugin](#) — Configure MySQL Server Plugins”
Section 4.8.1, “[perror](#) — Explain Error Codes”
Section 4.8.2, “[replace](#) — A String-Replacement Utility”
Section 4.8.3, “[resolveip](#) — Resolve Host name to IP Address or Vice Versa”
Section 5.1.3, “Server Command Options”
Section 4.2.3.1, “Using Options on the Command Line”

-v

Section 4.6.3.1, “[myisamchk](#) General Options”
Section 4.5.1.1, “[mysql](#) Options”
Section 4.6.7.2, “[mysqlbinlog](#) Row Event Display”
Section 4.6.1, “[innochecksum](#) — Offline InnoDB File Checksum Utility”
Section 4.6.2, “[myisam_ftdump](#) — Display Full-Text Index information”
Section 4.6.4, “[myisamlog](#) — Display MyISAM Log File Contents”
Section 4.6.5, “[myisampack](#) — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.14, “[mysql_waitpid](#) — Kill Process and Wait for Its Termination”
Section 4.6.6, “[mysqlaccess](#) — Client for Checking Access Privileges”
Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”
Section 4.6.8, “[mysqldumpslow](#) — Summarize Slow Query Log Files”
Section 4.5.2, “[mysqladmin](#) — Client for Administering a MySQL Server”
Section 4.5.3, “[mysqlcheck](#) — A Table Maintenance Program”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”
Section 4.5.5, “[mysqlimport](#) — A Data Import Program”
Section 4.5.6, “[mysqlshow](#) — Display Database, Table, and Column Information”
Section 4.5.7, “[mysqlslap](#) — Load Emulation Client”
Section 4.7.3, “[my_print_defaults](#) — Display Options from Option Files”
Section 4.4.4, “[mysql_plugin](#) — Configure MySQL Server Plugins”
Section 4.8.1, “[perror](#) — Explain Error Codes”
Section 4.8.2, “[replace](#) — A String-Replacement Utility”
Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 4.6.3.5, “Obtaining Table Information with [myisamchk](#)”
Section 5.1.3, “Server Command Options”
Section 4.2.3.1, “Using Options on the Command Line”

--var_name

Section 4.6.3.1, “[myisamchk](#) General Options”
Section 4.5.1.1, “[mysql](#) Options”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 4.6.7, “[mysqlbinlog](#) — Utility for Processing Binary Log Files”
Section 4.5.2, “[mysqladmin](#) — Client for Administering a MySQL Server”
Section 4.5.4, “[mysqldump](#) — A Database Backup Program”

Section 5.1.3, “Server Command Options”
Section 1.4, “What Is New in MySQL 5.5”

--verbose

Section 4.6.3.1, “`myisamchk` General Options”
Section 4.5.1.1, “`mysql` Options”
Section 4.6.7.2, “`mysqlbinlog` Row Event Display”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.4.13, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.10, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”
Section 4.6.14, “`mysql_waitpid` — Kill Process and Wait for Its Termination”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.8.1, “`perror` — Explain Error Codes”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Section 4.6.3.4, “Other `myisamchk` Options”
Section 5.1.3, “Server Command Options”
Section 2.11.1.3, “Starting and Troubleshooting the MySQL Server”
Section 8.11.2, “Tuning Server Parameters”

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”
Section 4.2.3.3, “Using Option Files”
Section 4.2.3.1, “Using Options on the Command Line”

--version

Section 4.6.3.1, “`myisamchk` General Options”
Section 4.5.1.1, “`mysql` Options”
Section 17.4.24, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 17.4.6, “`ndb_config` — Extract MySQL Cluster Configuration Information”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.10, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”
Section 4.6.14, “`mysql_waitpid` — Kill Process and Wait for Its Termination”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.7.2, “`mysql_config` — Get Compile Options for Compiling Clients”
Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.8.1, “`perror` — Explain Error Codes”
Section 4.8.3, “`resolveip` — Resolve Host name to IP Address or Vice Versa”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 5.1.3, “Server Command Options”
Section 4.2.3.1, “Using Options on the Command Line”

--vertical

Section 4.5.1.1, “`mysql` Options”

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 1.7, “How to Report Bugs or Problems”

W

[[index top](#) [4343]]

-W

Section 4.5.1.1, “`mysql` Options”
Section 4.2.2, “Connecting to the MySQL Server”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — Load Emulation Client”
Section 5.1.3, “Server Command Options”

-w

Section 4.6.3.1, “`myisamchk` General Options”
Section 4.5.1.1, “`mysql` Options”
Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”
Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--wait

Section 4.6.3.1, “`myisamchk` General Options”
Section 4.5.1.1, “`mysql` Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

--wait-nodes

Section 17.4.23, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”

--warnings

Section 1.4, “What Is New in MySQL 5.5”

--where

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--windows

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

--with-mysql-sock

Section 22.9.4, “MySQL Functions (PDO_MYSQL) (MySQL (PDO))”

--with-ndb-sci

Section 17.3.2.11, “SCI Transport Connections in MySQL Cluster”

--with-ndbcluster

Section 17.5.4, “MySQL Server Usage for MySQL Cluster”

--with-pdo-mysql

Section 22.9.4, “MySQL Functions (PDO_MYSQL) (MySQL (PDO))”

--with-plugins

Section 17.5.4, “MySQL Server Usage for MySQL Cluster”

--with-pstack

Section 1.4, “What Is New in MySQL 5.5”

--with-zlib-dir

Section 22.9.4, “MySQL Functions (PDO_MYSQL) (MySQL (PDO))”

WITH_BUNDLED_MEMCACHED

Section 2.10.4, “MySQL Source-Configuration Options”

WITH_CLASSPATH

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”
Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”

WITH_DEBUG

Section 2.10.4, “MySQL Source-Configuration Options”

WITH_NDB_JAVA

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”

Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”

WITH_NDBCLUSTER

Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”

WITH_NDBCLUSTER_STORAGE_ENGINE

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”

Section 17.2.2.2, “Compiling and Installing MySQL Cluster from Source on Windows”

Section 2.10.4, “MySQL Source-Configuration Options”

WITH_PERFSCHEMA_STORAGE_ENGINE

Section 21.2.1, “Performance Schema Build Configuration”

WITH_ZLIB

Section 2.10.4, “MySQL Source-Configuration Options”

--write-binlog

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”

X

[index top [4343]]

-X

Section 4.5.1.2, “mysql Commands”

Section 4.5.1.1, “mysql Options”

Section 4.5.4, “mysqldump — A Database Backup Program”

-x

Section 17.4.19, “ndb_select_all — Print Rows from an NDB Table”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.5.7, “mysqlslap — Load Emulation Client”

--xml

Section 4.5.1.1, “mysql Options”

Section 13.2.7, “LOAD XML Syntax”

Section 17.4.6, “ndb_config — Extract MySQL Cluster Configuration Information”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 12.11, “XML Functions”

Y

[index top [4343]]

-Y

Section 4.5.4, “mysqldump — A Database Backup Program”

-y

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.5.7, “mysqlslap — Load Emulation Client”

Z

[index top [4343]]

-z

Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”

Section 17.4.19, “ndb_select_all — Print Rows from an NDB Table”

Privileges Index

[A](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [I](#) | [L](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#)

A

[[index top \[4397\]](#)]

ALL

Section 13.7.1.3, “[GRANT Syntax](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”

ALL PRIVILEGES

Section 6.2.1, “[Privileges Provided by MySQL](#)”

ALTER

Section 13.1.1, “[ALTER DATABASE Syntax](#)”
Section 13.1.7, “[ALTER TABLE Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”
Section 13.1.32, “[RENAME TABLE Syntax](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 20.24, “[The INFORMATION_SCHEMA TABLE_PRIVILEGES Table](#)”

ALTER ROUTINE

Section 13.1.4, “[ALTER FUNCTION Syntax](#)”
Section 13.1.5, “[ALTER PROCEDURE Syntax](#)”
Section 13.1.15, “[CREATE PROCEDURE and CREATE FUNCTION Syntax](#)”
Section 13.1.26, “[DROP PROCEDURE and DROP FUNCTION Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”
Section 19.7, “[Binary Logging of Stored Programs](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 5.1.4, “[Server System Variables](#)”
Section 19.2.2, “[Stored Routines and MySQL Privileges](#)”

C

[[index top \[4397\]](#)]

CREATE

Section 13.1.7, “[ALTER TABLE Syntax](#)”
Section 13.1.10, “[CREATE DATABASE Syntax](#)”
Section 13.1.17, “[CREATE TABLE Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”
Section 13.1.32, “[RENAME TABLE Syntax](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”

CREATE ROUTINE

Section 13.1.15, “[CREATE PROCEDURE and CREATE FUNCTION Syntax](#)”

Section 13.7.1.3, “[GRANT Syntax](#)”
Section 19.7, “[Binary Logging of Stored Programs](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 5.1.4, “[Server System Variables](#)”
Section 19.2.2, “[Stored Routines and MySQL Privileges](#)”

CREATE TABLESPACE

Section 13.7.1.3, “[GRANT Syntax](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”

CREATE TEMPORARY TABLES

Section 13.1.17, “[CREATE TABLE Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”

CREATE USER

Section 13.7.1.1, “[CREATE USER Syntax](#)”
Section 13.7.1.2, “[DROP USER Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”
Section 13.7.1.4, “[RENAME USER Syntax](#)”
Section 13.7.1.5, “[REVOKE Syntax](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”

CREATE VIEW

Section 13.1.9, “[ALTER VIEW Syntax](#)”
Section 13.1.20, “[CREATE VIEW Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section E.5, “[Restrictions on Views](#)”
Section 20.24, “[The INFORMATION_SCHEMA TABLE_PRIVILEGES Table](#)”

D

[[index top \[4397\]](#)]

DELETE

Section 13.1.17, “[CREATE TABLE Syntax](#)”
Section 13.2.2, “[DELETE Syntax](#)”
Section 13.7.3.2, “[DROP FUNCTION Syntax](#)”
Section 13.7.1.2, “[DROP USER Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”
Section 13.2.8, “[REPLACE Syntax](#)”
Section 13.7.3.4, “[UNINSTALL PLUGIN Syntax](#)”
Section 6.2.5, “[Access Control, Stage 2: Request Verification](#)”
Section 23.3.2.5, “[Compiling and Installing User-Defined Functions](#)”
Section 5.1.8.1, “[Installing and Uninstalling Plugins](#)”
Section 17.5.11.2, “[MySQL Cluster and MySQL Privileges](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 14.10, “[The MERGE Storage Engine](#)”

Section 23.3.2.6, “User-Defined Function Security Precautions”

DROP

Section 13.1.7, “ALTER TABLE Syntax”
Section 13.1.9, “ALTER VIEW Syntax”
Section 13.1.20, “CREATE VIEW Syntax”
Section 13.1.21, “DROP DATABASE Syntax”
Section 13.1.28, “DROP TABLE Syntax”
Section 13.1.31, “DROP VIEW Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 13.1.32, “RENAME TABLE Syntax”
Section 13.1.33, “TRUNCATE TABLE Syntax”
Section 18.3.1, “Management of RANGE and LIST Partitions”
Section 21.6, “Performance Schema General Table Characteristics”
Section 6.2.1, “Privileges Provided by MySQL”
Section 20.24, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”
Section 6.2, “The MySQL Access Privilege System”

E

[index top [4397]]

EVENT

Section 13.1.2, “ALTER EVENT Syntax”
Section 13.1.11, “CREATE EVENT Syntax”
Section 13.1.22, “DROP EVENT Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 13.7.5.9, “SHOW CREATE EVENT Syntax”
Section 13.7.5.19, “SHOW EVENTS Syntax”
Section 19.4.1, “Event Scheduler Overview”
Section 19.4.3, “Event Syntax”
Section 6.2.1, “Privileges Provided by MySQL”
Section 19.4.6, “The Event Scheduler and MySQL Privileges”

EXECUTE

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 19.6, “Access Control for Stored Programs and Views”
Section 6.2.1, “Privileges Provided by MySQL”
Section 5.1.4, “Server System Variables”
Section 19.2.2, “Stored Routines and MySQL Privileges”

F

[index top [4397]]

FILE

Section 13.7.1.3, “GRANT Syntax”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 13.2.7, “LOAD XML Syntax”
Section 13.2.9.1, “SELECT ... INTO Syntax”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.2.2, “Privilege System Grant Tables”
Section 6.2.1, “Privileges Provided by MySQL”
Section 12.5, “String Functions”
Section 11.4.3, “The BLOB and TEXT Types”

G

[index top [4397]]

GRANT OPTION

Section 13.7.1.3, “GRANT Syntax”
Section 13.7.1.5, “REVOKE Syntax”
Section 6.2.1, “Privileges Provided by MySQL”
Section 20.5, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”

I

[index top [4397]]

INDEX

Section 13.1.7, “ALTER TABLE Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 6.2.1, “Privileges Provided by MySQL”
Section 20.24, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”

INSERT

Section 13.1.7, “ALTER TABLE Syntax”
Section 13.7.2.1, “ANALYZE TABLE Syntax”
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 13.7.1.1, “CREATE USER Syntax”
Section 13.1.20, “CREATE VIEW Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 13.2.5, “INSERT Syntax”
Section 13.7.2.4, “OPTIMIZE TABLE Syntax”
Section 13.1.32, “RENAME TABLE Syntax”
Section 13.7.2.5, “REPAIR TABLE Syntax”
Section 13.2.8, “REPLACE Syntax”
Section 19.6, “Access Control for Stored Programs and Views”

Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 6.3.2, “Adding User Accounts”
Section 23.3.2.5, “Compiling and Installing User-Defined Functions”
Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 14.2.1, “Pluggable Storage Engine Architecture”
Section 6.2.1, “Privileges Provided by MySQL”
Section 5.1.3, “Server Command Options”
Section 20.5, “The `INFORMATION_SCHEMA` `COLUMN_PRIVILEGES` Table”
Section 20.24, “The `INFORMATION_SCHEMA` `TABLE_PRIVILEGES` Table”
Section 19.4.6, “The Event Scheduler and MySQL Privileges”
Section 23.3.2.6, “User-Defined Function Security Precautions”

L

[[index top \[4397\]](#)]

LOCK TABLES

Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.7.1.3, “`GRANT` Syntax”
Section 13.3.5, “`LOCK TABLES` and `UNLOCK TABLES` Syntax”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.1, “Privileges Provided by MySQL”

P

[[index top \[4397\]](#)]

PROCESS

Section 13.7.1.3, “`GRANT` Syntax”
Chapter 20, *INFORMATION_SCHEMA Tables*
Section 13.7.6.4, “`KILL` Syntax”
Section 14.3.14.2, “`SHOW ENGINE INNODB STATUS` and the InnoDB Monitors”
Section 13.7.5.30, “`SHOW PROCESSLIST` Syntax”
Section 6.3.2, “Adding User Accounts”
Section 19.4.2, “Event Scheduler Configuration”
Section 8.12.5, “Examining Thread Information”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 6.2.1, “Privileges Provided by MySQL”
Section 20.14, “The `INFORMATION_SCHEMA` `PROCESSLIST` Table”

Section 21.7.5.2, “The `threads` Table”

PROXY

Section 13.7.1.3, “`GRANT` Syntax”
Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”
Section 6.2.2, “Privilege System Grant Tables”
Section 6.2.1, “Privileges Provided by MySQL”
Section 6.3.7, “Proxy Users”
Section 6.3.6.2.2.3, “Unix Password Authentication with Proxy Users and Group Mapping”
Section 6.3.6.3.2, “Using the Windows Authentication Plugin”

PROXY ... WITH GRANT OPTION

Section 6.3.7, “Proxy Users”

R

[[index top \[4397\]](#)]

REFERENCES

Section 13.7.1.3, “`GRANT` Syntax”
Section 6.2.1, “Privileges Provided by MySQL”
Section 20.5, “The `INFORMATION_SCHEMA` `COLUMN_PRIVILEGES` Table”
Section 20.24, “The `INFORMATION_SCHEMA` `TABLE_PRIVILEGES` Table”

RELOAD

Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.7.1.3, “`GRANT` Syntax”
Section 22.8.3.55, “`mysql_refresh()`”
Section 22.8.3.56, “`mysql_reload()`”
Section 13.7.6.6, “`RESET` Syntax”
Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 6.3.2, “Adding User Accounts”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 12.13, “Encryption and Compression Functions”
Section 6.2.2, “Privilege System Grant Tables”
Section 6.2.1, “Privileges Provided by MySQL”

REPLICATION CLIENT

Section 13.7.1.3, “`GRANT` Syntax”
Section 13.7.5.2, “`SHOW BINARY LOGS` Syntax”
Section 13.7.5.24, “`SHOW MASTER STATUS` Syntax”
Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”
Section 6.2.1, “Privileges Provided by MySQL”

REPLICATION SLAVE

- Section 13.7.1.3, “GRANT Syntax”
- Section 16.1.1.3, “Creating a User for Replication”
- Section 6.2.1, “Privileges Provided by MySQL”
- Section 16.3.7, “Setting Up Replication Using SSL”

S

[index top [4397]]

SELECT

- Section 13.7.2.1, “ANALYZE TABLE Syntax”
- Section 13.7.2.3, “CHECKSUM TABLE Syntax”
- Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
- Section 13.1.17, “CREATE TABLE Syntax”
- Section 13.1.19, “CREATE TRIGGER Syntax”
- Section 13.1.20, “CREATE VIEW Syntax”
- Section 13.2.2, “DELETE Syntax”
- Section 13.7.1.3, “GRANT Syntax”
- Section 13.2.5, “INSERT Syntax”
- Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”
- Section 13.7.2.4, “OPTIMIZE TABLE Syntax”
- Section 13.7.2.5, “REPAIR TABLE Syntax”
- Section 13.7.5.14, “SHOW CREATE VIEW Syntax”
- Section 13.7.5.22, “SHOW GRANTS Syntax”
- Section 13.2.11, “UPDATE Syntax”
- Section 19.6, “Access Control for Stored Programs and Views”
- Section 6.2.5, “Access Control, Stage 2: Request Verification”
- Section 22.2.6, “Connector/Net Connection String Options Reference”
- Section 4.6.9, “mysqlhotcopy — A Database Backup Program”
- Section 4.5.4, “mysqldump — A Database Backup Program”
- Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”
- Section 21.6, “Performance Schema General Table Characteristics”
- Section 6.2.1, “Privileges Provided by MySQL”
- Section E.5, “Restrictions on Views”
- Section 20.5, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”
- Section 20.24, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”
- Section 14.10, “The MERGE Storage Engine”
- Section 19.4.6, “The Event Scheduler and MySQL Privileges”
- Section 6.2, “The MySQL Access Privilege System”
- Section 19.3.1, “Trigger Syntax”

SHOW DATABASES

- Section 13.7.1.3, “GRANT Syntax”
- Section 13.7.5.15, “SHOW DATABASES Syntax”
- Section 6.2.1, “Privileges Provided by MySQL”
- Section 5.1.3, “Server Command Options”
- Section 5.1.4, “Server System Variables”

SHOW VIEW

- Section 13.7.1.3, “GRANT Syntax”
- Section 13.7.5.14, “SHOW CREATE VIEW Syntax”
- Section 4.5.4, “mysqldump — A Database Backup Program”
- Section 6.2.1, “Privileges Provided by MySQL”
- Section E.5, “Restrictions on Views”
- Section 20.27, “The INFORMATION_SCHEMA VIEWS Table”

SHUTDOWN

- Section 13.7.1.3, “GRANT Syntax”
- Section 22.8.3.65, “mysql_shutdown()”
- Section 6.2.5, “Access Control, Stage 2: Request Verification”
- Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
- Section 6.2.2, “Privilege System Grant Tables”
- Section 6.2.1, “Privileges Provided by MySQL”
- Section 5.1.12, “The Shutdown Process”

SUPER

- Section 13.1.4, “ALTER FUNCTION Syntax”
- Section 13.1.6, “ALTER SERVER Syntax”
- Section 13.1.9, “ALTER VIEW Syntax”
- Section 13.7.6.1, “BINLOG Syntax”
- Section 13.1.11, “CREATE EVENT Syntax”
- Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
- Section 13.1.16, “CREATE SERVER Syntax”
- Section 13.1.19, “CREATE TRIGGER Syntax”
- Section 13.1.20, “CREATE VIEW Syntax”
- Section 13.1.27, “DROP SERVER Syntax”
- Section 13.7.1.3, “GRANT Syntax”
- Section 13.7.6.4, “KILL Syntax”
- Section 22.8.3.12, “mysql_dump_debug_info()”
- Section 13.7.1.6, “SET PASSWORD Syntax”
- Section 13.4.1.3, “SET sql_log_bin Syntax”
- Section 13.3.6, “SET TRANSACTION Syntax”
- Section 13.7.4, “SET Syntax”
- Section 13.7.5.2, “SHOW BINARY LOGS Syntax”
- Section 13.7.5.24, “SHOW MASTER STATUS Syntax”
- Section 13.7.5.30, “SHOW PROCESSLIST Syntax”
- Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
- Section 13.4.2.5, “START SLAVE Syntax”
- Section 13.4.2.6, “STOP SLAVE Syntax”

Section C.5.2.7, “[Too many connections](#)”
Section 19.6, “[Access Control for Stored Programs and Views](#)”
Section 6.3.5, “[Assigning Account Passwords](#)”
Section 16.1.3.4, “[Binary Log Options and Variables](#)”
Section 19.7, “[Binary Logging of Stored Programs](#)”
Section 10.1.5, “[Configuring the Character Set and Collation for Applications](#)”
Section 4.6.7, “[mysqlbinlog — Utility for Processing Binary Log Files](#)”
Section 12.13, “[Encryption and Compression Functions](#)”
Section 12.9.6, “[Fine-Tuning MySQL Full-Text Search](#)”
Section 16.1.1, “[How to Set Up Replication](#)”
Section 6.1.3, “[Making MySQL Secure Against Attackers](#)”
Section 10.7, “[MySQL Server Locale Support](#)”
Section 10.6, “[MySQL Server Time Zone Support](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 16.1.2, “[Replication Formats](#)”
Section 16.1.3.3, “[Replication Slave Options and Variables](#)”
Section 16.3.8.2, “[Semisynchronous Replication Installation and Configuration](#)”
Section 5.1.7, “[Server SQL Modes](#)”
Section 5.1.4, “[Server System Variables](#)”
Section 5.2.4.2, “[Setting The Binary Log Format](#)”
Section 5.2.4, “[The Binary Log](#)”
Section 19.4.6, “[The Event Scheduler and MySQL Privileges](#)”
Section 5.1.5, “[Using System Variables](#)”

T

[\[index top \[4397\]\]](#)

TRIGGER

Section 13.1.19, “[CREATE TRIGGER Syntax](#)”
Section 13.1.30, “[DROP TRIGGER Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”
Section 13.7.5.39, “[SHOW TRIGGERS Syntax](#)”
Section 19.6, “[Access Control for Stored Programs and Views](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 20.25, “[The INFORMATION_SCHEMA TRIGGERS Table](#)”

U

[\[index top \[4397\]\]](#)

UPDATE

Section 13.1.17, “[CREATE TABLE Syntax](#)”
Section 13.1.19, “[CREATE TRIGGER Syntax](#)”
Section 13.7.1.3, “[GRANT Syntax](#)”

Section 13.2.5, “[INSERT Syntax](#)”
Section 13.7.1.4, “[RENAME USER Syntax](#)”
Section 13.7.1.5, “[REVOKE Syntax](#)”
Section 13.2.11, “[UPDATE Syntax](#)”
Section 19.6, “[Access Control for Stored Programs and Views](#)”
Section 17.5.11.2, “[MySQL Cluster and MySQL Privileges](#)”
Section 21.6, “[Performance Schema General Table Characteristics](#)”
Section 21.2.3, “[Performance Schema Runtime Configuration](#)”
Section 21.7.1, “[Performance Schema Setup Tables](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 20.5, “[The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table](#)”
Section 20.24, “[The INFORMATION_SCHEMA TABLE_PRIVILEGES Table](#)”
Section 14.10, “[The MERGE Storage Engine](#)”
Section 19.3.1, “[Trigger Syntax](#)”

USAGE

Section 13.7.1.3, “[GRANT Syntax](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”

SQL Modes Index

[A](#) | [D](#) | [E](#) | [H](#) | [I](#) | [M](#) | [N](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#)

A

[\[index top \[4403\]\]](#)

ALLOW_INVALID_DATES

Section 12.7, “Date and Time Functions”
Section 11.3, “Date and Time Types”
Section C.5.5.2, “Problems Using `DATE` Columns”
Section 5.1.7, “Server SQL Modes”
Section 11.3.1, “The `DATE`, `DATETIME`, and `TIMESTAMP` Types”

ANSI

Section 13.7.5.14, “`SHOW CREATE VIEW` Syntax”
Section 9.2.4, “Function Name Parsing and Resolution”
Section 5.1.7, “Server SQL Modes”
Section 20.27, “The `INFORMATION_SCHEMA VIEWS` Table”

ANSI_QUOTES

Section 14.3.5.4, “`FOREIGN KEY` Constraints”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 9.2, “Schema Object Names”
Section 5.1.7, “Server SQL Modes”
Section 9.1.1, “String Literals”

D

[\[index top \[4403\]\]](#)

DB2

Section 5.1.7, “Server SQL Modes”

E

[\[index top \[4403\]\]](#)

ERROR_FOR_DIVISION_BY_ZERO

Section 12.18.3, “Expression Handling”
Section 12.18.5, “Precision Math Examples”
Section 5.1.7, “Server SQL Modes”

H

[\[index top \[4403\]\]](#)

HIGH_NOT_PRECEDENCE

Section 9.5, “Expression Syntax”
Section 12.3.1, “Operator Precedence”
Section 5.1.7, “Server SQL Modes”

I

[\[index top \[4403\]\]](#)

IGNORE_SPACE

Section 4.5.1.1, “`mysql` Options”
Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 9.2.4, “Function Name Parsing and Resolution”
Section 5.1.7, “Server SQL Modes”

M

[\[index top \[4403\]\]](#)

MAXDB

Section 11.1.2, “Date and Time Type Overview”
Section 5.1.7, “Server SQL Modes”
Section 11.3.1, “The `DATE`, `DATETIME`, and `TIMESTAMP` Types”

MSSQL

Section 5.1.7, “Server SQL Modes”

MYSQL323

Section 5.1.7, “Server SQL Modes”

MYSQL40

Section 5.1.7, “Server SQL Modes”

N

[\[index top \[4403\]\]](#)

NO_AUTO_CREATE_USER

Section 13.7.1.3, “`GRANT` Syntax”
Section 6.3.2, “Adding User Accounts”
Section 5.1.7, “Server SQL Modes”

NO_AUTO_VALUE_ON_ZERO

Section 13.1.17, “`CREATE TABLE` Syntax”
Section 5.1.7, “Server SQL Modes”
Section 3.6.9, “Using `AUTO_INCREMENT`”

NO_BACKSLASH_ESCAPES

Section 5.1.7, “Server SQL Modes”

Section 12.5.1, “String Comparison Functions”
Section 9.1.1, “String Literals”

NO_DIR_IN_CREATE

Section 13.1.17, “CREATE TABLE Syntax”
Section 16.4.1.9, “Replication and DIRECTORY Table Options”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.7, “Server SQL Modes”
Section 18.2.6, “Subpartitioning”
Section 5.2.4, “The Binary Log”

NO_ENGINE_SUBSTITUTION

Section 13.1.7, “ALTER TABLE Syntax”
Section 13.1.17, “CREATE TABLE Syntax”
Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 5.1.7, “Server SQL Modes”
Section 14.1, “Setting the Storage Engine”
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”

NO_FIELD_OPTIONS

Section 5.1.7, “Server SQL Modes”

NO_KEY_OPTIONS

Section 5.1.7, “Server SQL Modes”

NO_TABLE_OPTIONS

Section 5.1.7, “Server SQL Modes”

NO_UNSIGNED_SUBTRACTION

Section 12.6.1, “Arithmetic Operators”
Section 12.10, “Cast Functions and Operators”
Section 11.1.1, “Numeric Type Overview”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.1.7, “Server SQL Modes”

NO_ZERO_DATE

Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP”
Section 12.10, “Cast Functions and Operators”
Section 11.3, “Date and Time Types”
Section C.5.5.2, “Problems Using DATE Columns”
Section 5.1.7, “Server SQL Modes”

NO_ZERO_IN_DATE

Section 13.1.17, “CREATE TABLE Syntax”
Section 11.3, “Date and Time Types”
Section C.5.5.2, “Problems Using DATE Columns”
Section 5.1.7, “Server SQL Modes”

O

[index top [4403]]

ONLY_FULL_GROUP_BY

Section 3.3.4.8, “Counting Rows”
Section 12.16.3, “MySQL Extensions to GROUP BY”
Section 5.1.7, “Server SQL Modes”

ORACLE

Section 5.1.7, “Server SQL Modes”

P

[index top [4403]]

PAD_CHAR_TO_FULL_LENGTH

Section 5.1.7, “Server SQL Modes”
Section 11.1.3, “String Type Overview”
Section 11.4.1, “The CHAR and VARCHAR Types”

PIPES_AS_CONCAT

Section 9.5, “Expression Syntax”
Section 12.3.1, “Operator Precedence”
Section 5.1.7, “Server SQL Modes”

POSTGRESQL

Section 5.1.7, “Server SQL Modes”

R

[index top [4403]]

REAL_AS_FLOAT

Section 11.1.1, “Numeric Type Overview”
Section 11.2, “Numeric Types”
Section 5.1.7, “Server SQL Modes”

S

[index top [4403]]

STRICT_ALL_TABLES

Section 6.3.2, “Adding User Accounts”
Section 1.8.6.2, “Constraints on Invalid Data”
Section 12.18.3, “Expression Handling”
Section 5.1.7, “Server SQL Modes”

STRICT_TRANS_TABLES

Section 6.3.2, “Adding User Accounts”
Section 1.8.6.2, “Constraints on Invalid Data”

[Section 12.18.3, "Expression Handling"](#)
[Section 5.1.7, "Server SQL Modes"](#)

T

[\[index top \[4403\]\]](#)

TRADITIONAL

[Section 11.3.5, "Automatic Initialization and Updating for
TIMESTAMP"](#)

[Section 12.18.3, "Expression Handling"](#)

[Section 5.1.7, "Server SQL Modes"](#)

Statement/Syntax Index

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [K](#) | [L](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#) | [W](#) | [X](#)

A

[\[index top \[4407\]\]](#)

ALTER DATABASE

[Section 13.1.1, “ALTER DATABASE Syntax”](#)
[Section 10.1.3.2, “Database Character Set and Collation”](#)
[Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”](#)
[Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”](#)
[Section 1.8.4, “MySQL Extensions to Standard SQL”](#)
[Section 16.1.3.3, “Replication Slave Options and Variables”](#)

ALTER EVENT

[Section 13.1.2, “ALTER EVENT Syntax”](#)
[Section 13.1.11, “CREATE EVENT Syntax”](#)
[Section 19.7, “Binary Logging of Stored Programs”](#)
[Section 19.4.4, “Event Metadata”](#)
[Section 19.4.1, “Event Scheduler Overview”](#)
[Section 19.4.3, “Event Syntax”](#)
[Section 12.14, “Information Functions”](#)
[Section 16.4.1.6, “Replication of CURRENT_USER\(\)”](#)
[Section 16.4.1.10, “Replication of Invoked Features”](#)
[Section E.1, “Restrictions on Stored Programs”](#)
[Section 13.3.3, “Statements That Cause an Implicit Commit”](#)
[Section 20.7, “The INFORMATION_SCHEMA EVENTS Table”](#)
[Section 19.4.6, “The Event Scheduler and MySQL Privileges”](#)

ALTER EVENT event_name

ENABLED

[Section 16.4.1.10, “Replication of Invoked Features”](#)

ALTER FUNCTION

[Section 13.1.4, “ALTER FUNCTION Syntax”](#)
[Section 19.7, “Binary Logging of Stored Programs”](#)
[Section 13.3.3, “Statements That Cause an Implicit Commit”](#)
[Section 19.2.1, “Stored Routine Syntax”](#)

ALTER LOGFILE GROUP

[Section 13.1.3, “ALTER LOGFILE GROUP Syntax”](#)
[Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”](#)

[Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”](#)
[Section 17.5.12.1, “MySQL Cluster Disk Data Objects”](#)
[Section 20.29.1, “The INFORMATION_SCHEMA FILES Table”](#)

ALTER ONLINE TABLE

[Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”](#)

ALTER ONLINE TABLE ...

REORGANIZE PARTITION

[Section 17.5.13.2, “Adding MySQL Cluster Data Nodes Online: Basic procedure”](#)
[Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”](#)
[Section 17.5.13.1, “Adding MySQL Cluster Data Nodes Online: General Issues”](#)
[Section 17.5.2, “Commands in the MySQL Cluster Management Client”](#)
[Section 17.2.7, “Upgrading and Downgrading MySQL Cluster NDB 7.2”](#)

ALTER ONLINE TABLE tbl

REORGANIZE PARTITION

[Section 17.5.13.2, “Adding MySQL Cluster Data Nodes Online: Basic procedure”](#)

ALTER PROCEDURE

[Section 13.1.5, “ALTER PROCEDURE Syntax”](#)
[Section 19.7, “Binary Logging of Stored Programs”](#)
[Section 13.3.3, “Statements That Cause an Implicit Commit”](#)
[Section 19.2.1, “Stored Routine Syntax”](#)

ALTER SCHEMA

[Section 13.1.1, “ALTER DATABASE Syntax”](#)

ALTER SERVER

[Section 13.1.6, “ALTER SERVER Syntax”](#)
[Section 16.4.1.5, “Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER”](#)

ALTER TABLE

[Section 4.6.3.1, “myisamchk General Options”](#)
[Section 13.1.7.3, “ALTER TABLE Examples”](#)
[Section 13.1.7.2, “ALTER TABLE Online Operations”](#)
[Section 13.1.7.1, “ALTER TABLE Partition Operations”](#)
[Section 13.1.7, “ALTER TABLE Syntax”](#)
[Section 13.7.2.2, “CHECK TABLE Syntax”](#)
[Section 13.1.13, “CREATE INDEX Syntax”](#)
[Section 13.1.17, “CREATE TABLE Syntax”](#)
[Section 13.1.24, “DROP INDEX Syntax”](#)

Section 8.8.2, “`EXPLAIN` Output Format”

Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”

Section 14.3.5.4, “`FOREIGN KEY` Constraints”

Section 13.7.1.3, “`GRANT` Syntax”

Section 8.3.7, “`InnoDB` and `MyISAM` Index Statistics Collection”

Section 13.2.5.2, “`INSERT DELAYED` Syntax”

Section 13.7.6.4, “`KILL` Syntax”

Section 13.3.5, “`LOCK TABLES` and `UNLOCK TABLES` Syntax”

Section 14.10.2, “`MERGE` Table Problems”

Section 14.5.1, “`MyISAM` Startup Options”

Section 14.5.3, “`MyISAM` Table Storage Formats”

Section 22.8.3.35, “`mysql_info()`”

Section 13.7.2.4, “`OPTIMIZE TABLE` Syntax”

Section 18.2.3.1, “`RANGE COLUMNS` partitioning”

Section 18.2.1, “`RANGE` Partitioning”

Section 13.1.32, “`RENAME TABLE` Syntax”

Section 13.7.5.16, “`SHOW ENGINE` Syntax”

Section 13.7.5.41, “`SHOW WARNINGS` Syntax”

Section C.5.7.2, “`TEMPORARY` Table Problems”

Section 11.3.4, “`YEAR(2)` Limitations and Migrating to `YEAR(4)`”

Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”

Section 16.1.3.4, “Binary Log Options and Variables”

Section 10.1.3.4, “Column Character Set and Collation”

Section 10.1.13, “Column Character Set Conversion”

Section 14.3.9.2, “Consistent Nonlocking Reads”

Section 3.3.2, “Creating a Table”

Section 12.17.4.3, “Creating Spatial Columns”

Section 12.17.6.1, “Creating Spatial Indexes”

Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”

Section 14.3.12.3, “Defragmenting a Table”

Section 17.4.18, “`ndb_restore` — Restore a MySQL Cluster Backup”

Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”

Section 14.3.7.2, “Forcing `InnoDB` Recovery”

Section 12.9, “Full-Text Search Functions”

Section 8.12.5.2, “General Thread States”

Section C.5.4.3, “How MySQL Handles a Full Disk”

Section 8.9.3.1, “How the Query Cache Operates”

Section 7.6.3, “How to Repair `MyISAM` Tables”

Section 12.14, “Information Functions”

Section 17.2.3, “Initial Configuration of MySQL Cluster”

Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”

Section C.5.8, “Known Issues in MySQL”

Section 17.1.6.10, “Limitations Relating to Multiple MySQL Cluster Nodes”

Section 17.1.6.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”

Section E.10.3, “Limits on Table Size”

Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”

Section 18.3.3, “Maintenance of Partitions”

Section 18.3.2, “Management of `HASH` and `KEY` Partitions”

Section 18.3.1, “Management of `RANGE` and `LIST` Partitions”

Section 17.3.2.1, “MySQL Cluster Configuration: Basic Example”

Section 17.5.12.1, “MySQL Cluster Disk Data Objects”

Section 17.2.5, “MySQL Cluster Example with Tables and Data”

Section 17.6.4, “MySQL Cluster Replication Schema and Tables”

Section 17.3.4.3, “MySQL Cluster System Variables”

Section 1.8.4, “MySQL Extensions to Standard SQL”

Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”

Section 11.2.6, “Out-of-Range and Overflow Handling”

Section 18.1, “Overview of Partitioning in MySQL”

Section 18.3, “Partition Management”

Section 18.5.4, “Partitioning and Table-Level Locking”

Section 18.5.1, “Partitioning Keys, Primary Keys, and Unique Keys”

Section 18.5.2, “Partitioning Limitations Relating to Storage Engines”

Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”

Section 6.2.1, “Privileges Provided by MySQL”

Section C.5.7.1, “Problems with `ALTER TABLE`”

Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”

Section 16.4.1.1, “Replication and `AUTO_INCREMENT`”

Section 16.4.1.24, “Replication and Reserved Words”

Section 16.4.1.8.1, “Replication with More Columns on Master or Slave”

Section 18.5, “Restrictions and Limitations on Partitioning”

Section E.5, “Restrictions on Views”

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”

Section 5.1.3, “Server Command Options”

Section 5.1.7, “Server SQL Modes”

Section 5.1.4, “Server System Variables”

Section 5.2.4.2, “Setting The Binary Log Format”

Section 14.1, “Setting the Storage Engine”

Section 13.1.17.2, “Silent Column Specification Changes”

Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 11.1.3, “String Type Overview”

Section 10.1.3.3, “Table Character Set and Collation”

Section 14.8, “The `ARCHIVE` Storage Engine”

Section 20.12, “The `INFORMATION_SCHEMA PARTITIONS` Table”
Section 14.6, “The `MEMORY` Storage Engine”
Section 14.5, “The `MyISAM` Storage Engine”
Section 5.2.5, “The Slow Query Log”
Section 14.3.5.3.1, “Traditional `InnoDB` Auto-Increment Locking”
Section 14.3.14.4, “Troubleshooting `InnoDB` Data Dictionary Operations”
Section 17.1.6.6, “Unsupported or Missing Features in MySQL Cluster”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 10.1.11, “Upgrading from Previous to Current Unicode Support”
Section 3.6.9, “Using `AUTO_INCREMENT`”
Section 14.3.3, “Using Per-Table Tablespaces”
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”
Section 8.11.3.1.2, “Using Symbolic Links for `MyISAM` Tables on Unix”
Section 1.4, “What Is New in MySQL 5.5”
Section C.5.4.2, “What to Do If MySQL Keeps Crashing”
Section C.5.4.4, “Where MySQL Stores Temporary Files”

ALTER TABLE ... ADD PARTITION

Section 18.3.1, “Management of `RANGE` and `LIST` Partitions”

ALTER TABLE ... ENGINE = MEMORY

Section 16.4.1.20, “Replication and `MEMORY` Tables”

ALTER TABLE ... OPTIMIZE PARTITION

Section 18.5.2, “Partitioning Limitations Relating to Storage Engines”

ALTER TABLE ... PARTITION BY

Section 18.5.1, “Partitioning Keys, Primary Keys, and Unique Keys”

ALTER TABLE ... PARTITION BY ...

Section 18.3.1, “Management of `RANGE` and `LIST` Partitions”
Section 18.5, “Restrictions and Limitations on Partitioning”

ALTER TABLE ... RENAME

Section 8.11.3.1.2, “Using Symbolic Links for `MyISAM` Tables on Unix”

ALTER TABLE ... REORGANIZE PARTITION

Section 17.5.13.1, “Adding MySQL Cluster Data Nodes Online: General Issues”

ALTER TABLE ... TRUNCATE PARTITION

Section 18.3.3, “Maintenance of Partitions”
Section 18.3, “Partition Management”
Section 1.4, “What Is New in MySQL 5.5”

ALTER TABLE ... TRUNCATE PARTITION ALL

Section 18.3.3, “Maintenance of Partitions”

ALTER TABLE

`mysql.ndb_apply_status`

ENGINE=MyISAM

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

ALTER TABLE `mysql.proc` ENGINE = NDB

Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”

ALTER TABLE `table_name` REORGANIZE PARTITION

Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”

ALTER TABLE `tbl_name` FORCE

Section 13.1.7, “`ALTER TABLE` Syntax”

ALTER TABLESPACE

Section 13.1.8, “`ALTER TABLESPACE` Syntax”
Section 13.1.18, “`CREATE TABLESPACE` Syntax”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
Section 17.5.12.1, “MySQL Cluster Disk Data Objects”
Section 20.29.1, “The `INFORMATION_SCHEMA FILES` Table”

ALTER VIEW

Section 13.1.9, “`ALTER VIEW` Syntax”
Section 13.1.20, “`CREATE VIEW` Syntax”
Section 12.14, “Information Functions”
Section 16.4.1.6, “Replication of `CURRENT_USER()`”
Section E.1, “Restrictions on Stored Programs”

Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 19.5.2, “View Processing Algorithms”

Section 19.5.1, “View Syntax”

ANALYZE TABLE

Section 4.6.3.1, “myisamchk General Options”

Section 13.1.7, “ALTER TABLE Syntax”

Section 13.7.2.1, “ANALYZE TABLE Syntax”

Section 8.8.2, “EXPLAIN Output Format”

Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”

Section 14.3.4, “InnoDB Startup Options and System Variables”

Section 14.10.2, “MERGE Table Problems”

Section 7.6, “MyISAM Table Maintenance and Crash Recovery”

Section 13.7.5.23, “SHOW INDEX Syntax”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”

Section 8.12.5.2, “General Thread States”

Section C.5.8, “Known Issues in MySQL”

Section 14.3.15, “Limits on InnoDB Tables”

Section 18.3.3, “Maintenance of Partitions”

Section 1.8.4, “MySQL Extensions to Standard SQL”

Section 8.6.1, “Optimizing MyISAM Queries”

Section 8.8.1, “Optimizing Queries with EXPLAIN”

Section 6.2.1, “Privileges Provided by MySQL”

Section 16.4.1.12, “Replication and FLUSH”

Section 18.5, “Restrictions and Limitations on Partitioning”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 5.2.5, “The Slow Query Log”

B

[index top [4407]]

BEGIN

Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”

Section 14.3.13, “InnoDB Error Handling”

Section 19.7, “Binary Logging of Stored Programs”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 16.4.1.31, “Replication and Transactions”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section E.1, “Restrictions on Stored Programs”

Section 5.1.4, “Server System Variables”

Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 14.3.9, “The InnoDB Transaction Model and Locking”

BEGIN ... END

Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”

Section 13.6.5.1, “CASE Syntax”

Section 13.1.19, “CREATE TRIGGER Syntax”

Section 13.6.7.2, “DECLARE ... HANDLER Syntax”

Section 13.6.3, “DECLARE Syntax”

Section 13.6.5.4, “LEAVE Syntax”

Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”

Section 13.6.6.1, “Cursor CLOSE Syntax”

Section 19.1, “Defining Stored Programs”

Section 19.4.1, “Event Scheduler Overview”

Section 13.6.4.1, “Local Variable DECLARE Syntax”

Section 13.6.4.2, “Local Variable Scope and Resolution”

Section 13.6, “MySQL Compound-Statement Syntax”

Section E.1, “Restrictions on Stored Programs”

Section 13.6.2, “Statement Label Syntax”

Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 19.3.1, “Trigger Syntax”

BINLOG

Section 4.6.7.2, “mysqlbinlog Row Event Display”

Section 13.7.6.1, “BINLOG Syntax”

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

C

[index top [4407]]

CACHE INDEX

Section 13.7.6.2, “CACHE INDEX Syntax”

Section 13.7.6.5, “LOAD INDEX INTO CACHE Syntax”

Section 8.9.2.4, “Index Preloading”

Section 8.9.2.2, “Multiple Key Caches”

Section 18.5, “Restrictions and Limitations on Partitioning”

Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 1.4, “What Is New in MySQL 5.5”

CALL

Section 13.2.1, “CALL Syntax”

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”

Section 22.8.3.1, “mysql_affected_rows()”

Section 22.8.3.37, “mysql_insert_id()”
Section 22.8.3.45, “mysql_more_results()”
Section 22.8.3.46, “mysql_next_result()”
Section 22.8.3.52, “mysql_real_connect()”
Section 22.8.3.64, “mysql_set_server_option()”
Section 22.8.7.17, “mysql_stmt_next_result()”
Section 19.6, “Access Control for Stored Programs and Views”
Section 19.7, “Binary Logging of Stored Programs”
Section 22.8.1, “C API Data Structures”
Section 22.8.14, “C API Prepared Statement Problems”
Section 22.8.13, “C API Support for Multiple Statement Execution”
Section 22.8.16, “C API Support for Prepared CALL Statements”
Section 13.5, “SQL Syntax for Prepared Statements”
Chapter 19, *Stored Programs and Views*
Section 19.2.1, “Stored Routine Syntax”
Section 19.3.1, “Trigger Syntax”

CALL p()

Section 13.6.7.3.3, “RESIGNAL with a Condition Value and Optional New Signal Information”

CALL stored_procedure()

Section 18.5.4, “Partitioning and Table-Level Locking”

CASE

Section 13.6.5.1, “CASE Syntax”
Section 12.4, “Control Flow Functions”
Section 13.6.5, “Flow Control Statements”

CHANGE MASTER TO

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 13.4.2.3, “RESET SLAVE Syntax”
Section 13.7.1.6, “SET PASSWORD Syntax”
Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
Section 16.3.1.2, “Backing Up Raw Data from a Slave”
Section 16.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”
Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”
Section 17.6.5, “Preparing the MySQL Cluster for Replication”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.18, “Replication and Master or Slave Shutdowns”
Section 16.1, “Replication Configuration”

Section 8.12.5.8, “Replication Slave Connection Thread States”
Section 8.12.5.6, “Replication Slave I/O Thread States”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.1.6, “Server Status Variables”
Section 16.1.1.10, “Setting the Master Configuration on the Slave”
Section 16.3.7, “Setting Up Replication Using SSL”
Section 16.1.1.8, “Setting Up Replication with Existing Data”
Section 16.1.1.7, “Setting Up Replication with New Master and Slaves”
Section 16.2.2.2, “Slave Status Logs”
Section 16.3.6, “Switching Masters During Failover”
Section 16.1.2.2, “Usage of Row-Based Logging and Replication”
Section 1.4, “What Is New in MySQL 5.5”

CHECK TABLE

Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 13.7.2.2, “CHECK TABLE Syntax”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.20, “CREATE VIEW Syntax”
Section 14.3.14.3, “InnoDB General Troubleshooting”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section C.5.2.9, “MySQL server has gone away”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 14.3.7, “Backing Up and Recovering an InnoDB Database”
Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”
Section 14.5.4.1, “Corrupted MyISAM Tables”
Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 8.10.5, “External Locking”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 18.3.3, “Maintenance of Partitions”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 14.5.4.2, “Problems from Tables Not Being Closed Properly”
Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”

Section 18.5, “Restrictions and Limitations on Partitioning”
Section E.3, “Restrictions on Server-Side Cursors”
Section E.1, “Restrictions on Stored Programs”
Section E.5, “Restrictions on Views”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 7.6.5, “Setting Up a `MyISAM` Table Maintenance Schedule”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 14.10, “The `MERGE` Storage Engine”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

CHECK TABLE ... EXTENDED

Section 13.7.2.2, “`CHECK TABLE` Syntax”

CHECK TABLE ... FOR UPGRADE

Section 13.7.2.5, “`REPAIR TABLE` Syntax”
Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”

CHECKSUM TABLE

Section 13.7.2.3, “`CHECKSUM TABLE` Syntax”
Section 13.1.17, “`CREATE TABLE` Syntax”

CLOSE

Section 13.6.6.1, “Cursor `CLOSE` Syntax”

COMMIT

Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 14.3.13, “`InnoDB` Error Handling”
Section 13.3.4, “`SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, and `RELEASE SAVEPOINT` Syntax”
Section 13.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”
Section 19.7, “Binary Logging of Stored Programs”
Section 8.5.4, “Bulk Data Loading for `InnoDB` Tables”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 14.3.9.7, “Implicit Transaction Commit and Rollback”
Section 14.3.15, “Limits on `InnoDB` Tables”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 13.3, “MySQL Transactional and Locking Statements”
Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 16.4.1.31, “Replication and Transactions”

Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 14.3.9, “The `InnoDB` Transaction Model and Locking”
Section 5.2.4, “The Binary Log”
Section 1.8.5.3, “Transaction and Atomic Operation Differences”
Section 19.3.1, “Trigger Syntax”
Section 3.6.9, “Using `AUTO_INCREMENT`”
Section 14.3.5.1, “Using `InnoDB` Transactions”

CREATE DATABASE

Section 13.1.10, “`CREATE DATABASE` Syntax”
Section 22.8.3.8, “`mysql_create_db()`”
Section 13.7.5.8, “`SHOW CREATE DATABASE` Syntax”
Section 10.1.9.3, “`SHOW` Statements and `INFORMATION_SCHEMA`”
Section 7.1, “Backup and Recovery Types”
Section 22.8.2, “C API Function Overview”
Section 10.1.5, “Configuring the Character Set and Collation for Applications”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 10.1.3.2, “Database Character Set and Collation”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.1, “Dumping Data in SQL Format with `mysqldump`”
Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 9.2.2, “Identifier Case Sensitivity”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 10.1.3.1, “Server Character Set and Collation”
Section 13.3.3, “Statements That Cause an Implicit Commit”

CREATE DATABASE dbx

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

CREATE DATABASE IF NOT EXISTS

Section 16.4.1.3, “Replication of `CREATE ... IF NOT EXISTS` Statements”

CREATE EVENT

Section 13.1.2, “`ALTER EVENT` Syntax”
Section 13.1.11, “`CREATE EVENT` Syntax”
Section 13.7.5.9, “`SHOW CREATE EVENT` Syntax”
Section 19.7, “Binary Logging of Stored Programs”
Section 19.4.4, “Event Metadata”
Section 19.4.3, “Event Syntax”
Section 12.14, “Information Functions”
Section 16.4.1.6, “Replication of `CURRENT_USER()`”
Section 16.4.1.10, “Replication of Invoked Features”
Section E.1, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “The `INFORMATION_SCHEMA` `EVENTS` Table”
Section 19.4.6, “The Event Scheduler and MySQL Privileges”

CREATE EVENT IF NOT EXISTS

Section 16.4.1.3, “Replication of `CREATE ... IF NOT EXISTS` Statements”

CREATE FUNCTION

Section 13.1.4, “`ALTER FUNCTION` Syntax”
Section 13.1.12, “`CREATE FUNCTION` Syntax”
Section 13.7.3.1, “`CREATE FUNCTION` Syntax for User-Defined Functions”
Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 13.7.3.2, “`DROP FUNCTION` Syntax”
Section 23.3, “Adding New Functions to MySQL”
Section 19.7, “Binary Logging of Stored Programs”
Section 23.3.2.5, “Compiling and Installing User-Defined Functions”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 9.2.4, “Function Name Parsing and Resolution”
Section 12.14, “Information Functions”
Section 16.4.1.6, “Replication of `CURRENT_USER()`”
Section 16.4.1.10, “Replication of Invoked Features”
Section E.1, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 19.2.1, “Stored Routine Syntax”
Section 23.3.2.1, “UDF Calling Sequences for Simple Functions”
Section 2.12.1, “Upgrading MySQL”
Section 23.3.2.6, “User-Defined Function Security Precautions”

CREATE INDEX

Section 13.1.7.2, “`ALTER TABLE` Online Operations”
Section 13.1.13, “`CREATE INDEX` Syntax”
Section 12.17.6.1, “Creating Spatial Indexes”
Section 12.9, “Full-Text Search Functions”
Section 8.7, “Optimizing for `MEMORY` Tables”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 17.1.6.6, “Unsupported or Missing Features in MySQL Cluster”

CREATE LOGFILE GROUP

Section 13.1.3, “`ALTER LOGFILE GROUP` Syntax”
Section 13.1.14, “`CREATE LOGFILE GROUP` Syntax”
Section 13.1.18, “`CREATE TABLESPACE` Syntax”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
Section 17.5.12.1, “MySQL Cluster Disk Data Objects”
Section 20.29.1, “The `INFORMATION_SCHEMA` `FILES` Table”

CREATE OR REPLACE VIEW

Section 13.1.9, “`ALTER VIEW` Syntax”
Section 13.1.20, “`CREATE VIEW` Syntax”
Section E.5, “Restrictions on Views”

CREATE PROCEDURE

Section 13.1.5, “`ALTER PROCEDURE` Syntax”
Section 13.2.1, “`CALL` Syntax”
Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 19.7, “Binary Logging of Stored Programs”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 12.14, “Information Functions”
Section 16.4.1.6, “Replication of `CURRENT_USER()`”
Section 16.4.1.10, “Replication of Invoked Features”
Section E.1, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 19.2.1, “Stored Routine Syntax”

CREATE SCHEMA

Section 13.1.10, “`CREATE DATABASE` Syntax”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”

CREATE SERVER

Section 13.1.6, “`ALTER SERVER` Syntax”

Section 13.1.16, “CREATE SERVER Syntax”
Section 14.11.3, “FEDERATED Storage Engine Notes and Tips”
Section 13.7.6.3, “FLUSH Syntax”
Section 14.11.2.2, “Creating a FEDERATED Table Using CREATE SERVER”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 14.11.2, “How to Create FEDERATED Tables”
Section 16.4.1.5, “Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER”

CREATE TABLE

Section 4.5.1.1, “mysql Options”
Section 13.1.7.3, “ALTER TABLE Examples”
Section 13.1.7.2, “ALTER TABLE Online Operations”
Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 13.1.7, “ALTER TABLE Syntax”
Section 13.1.11, “CREATE EVENT Syntax”
Section 13.1.13, “CREATE INDEX Syntax”
Section 13.1.16, “CREATE SERVER Syntax”
Section 13.1.17.1, “CREATE TABLE ... SELECT Syntax”
Section 13.1.17, “CREATE TABLE Syntax”
Section 13.8.1, “DESCRIBE Syntax”
Section 14.3.5.4, “FOREIGN KEY Constraints”
Section 18.2.4, “HASH Partitioning”
Section 13.8.3, “HELP Syntax”
Section 14.3.5.5, “InnoDB and MySQL Replication”
Section 14.3.14.3, “InnoDB General Troubleshooting”
Section 18.2.5, “KEY Partitioning”
Section 18.2.2, “LIST Partitioning”
Section 13.2.7, “LOAD XML Syntax”
Section 14.5.3, “MyISAM Table Storage Formats”
Section 18.2.3.1, “RANGE COLUMNS partitioning”
Section 18.2.1, “RANGE Partitioning”
Section 13.7.5.6, “SHOW COLUMNS Syntax”
Section 13.7.5.12, “SHOW CREATE TABLE Syntax”
Section 14.3.14.2, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”
Section 13.7.5.16, “SHOW ENGINE Syntax”
Section 13.7.5.37, “SHOW TABLE STATUS Syntax”
Section 13.7.5.41, “SHOW WARNINGS Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 13.1.33, “TRUNCATE TABLE Syntax”
Section 13.7.3.4, “UNINSTALL PLUGIN Syntax”
Section 7.1, “Backup and Recovery Types”
Section 16.1.3.4, “Binary Log Options and Variables”
Section 10.1.3.4, “Column Character Set and Collation”
Section 8.3.4, “Column Indexes”
Section 14.11.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 3.3.2, “Creating a Table”

Section 14.3.5, “Creating and Using InnoDB Tables”
Section 12.17.4.3, “Creating Spatial Columns”
Section 12.17.6.1, “Creating Spatial Indexes”
Section 7.2, “Database Backup Methods”
Section 10.1.3.2, “Database Character Set and Collation”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 1.8.5.4, “Foreign Key Differences”
Section 12.9, “Full-Text Search Functions”
Section 3.4, “Getting Information About Databases and Tables”
Section 18.2.7, “How MySQL Partitioning Handles NULL”
Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”
Section 9.2.2, “Identifier Case Sensitivity”
Section 12.14, “Information Functions”
Section 17.2.3, “Initial Configuration of MySQL Cluster”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
Section E.10.3, “Limits on Table Size”
Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”
Section 3.3.3, “Loading Data into a Table”
Section 5.2.4.4, “Logging Format for Changes to mysql Database Tables”
Section 18.3.1, “Management of RANGE and LIST Partitions”
Section 17.3.2.1, “MySQL Cluster Configuration: Basic Example”
Section 17.5.12.1, “MySQL Cluster Disk Data Objects”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 17.5.15, “NDB API Statistics Counters and Variables”
Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”
Section 8.5.6, “Optimizing InnoDB DDL Operations”
Section 8.4.1, “Optimizing Data Size”
Section 18.1, “Overview of Partitioning in MySQL”
Section 18.3, “Partition Management”
Section 18.5.1, “Partitioning Keys, Primary Keys, and Unique Keys”
Section 18.5.3, “Partitioning Limitations Relating to Functions”
Section 18.5.2, “Partitioning Limitations Relating to Storage Engines”
Section 18.2, “Partitioning Types”
Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”
Section 6.2.1, “Privileges Provided by MySQL”
Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”

Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 16.4.1.1, “Replication and `AUTO_INCREMENT`”
Section 16.4.1.9, “Replication and `DIRECTORY` Table Options”
Section 16.4.1.2, “Replication and Character Sets”
Section 16.4.1.13, “Replication and System Functions”
Section 16.4.1.4, “Replication of `CREATE TABLE ... SELECT` Statements”
Section 16.4.1.8.1, “Replication with More Columns on Master or Slave”
Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.3, “Server Command Options”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 5.2.4.2, “Setting The Binary Log Format”
Section 14.1, “Setting the Storage Engine”
Section 13.1.17.2, “Silent Column Specification Changes”
Section C.1, “Sources of Error Information”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 11.1.3, “String Type Overview”
Section 18.2.6, “Subpartitioning”
Section 10.1.3.3, “Table Character Set and Collation”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 11.4.4, “The `ENUM` Type”
Section 20.12, “The `INFORMATION_SCHEMA PARTITIONS` Table”
Section 14.6, “The `MEMORY` Storage Engine”
Section 14.5, “The `MyISAM` Storage Engine”
Section 13.2.10.1, “The Subquery as Scalar Operand”
Section 14.3.5.3.1, “Traditional `InnoDB` Auto-Increment Locking”
Section 14.3.14.4, “Troubleshooting `InnoDB` Data Dictionary Operations”
Section 17.1.6.6, “Unsupported or Missing Features in MySQL Cluster”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 7.4, “Using `mysqldump` for Backups”
Section 3.6.9, “Using `AUTO_INCREMENT`”
Section 3.3.4.9, “Using More Than one Table”
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”
Section 8.11.3.1.2, “Using Symbolic Links for `MyISAM` Tables on Unix”
Section 1.4, “What Is New in MySQL 5.5”
Section E.10.5, “Windows Platform Limitations”

CREATE TABLE ... LIKE

Section 13.1.17, “`CREATE TABLE` Syntax”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”

CREATE TABLE ... SELECT

Section 13.1.17.1, “`CREATE TABLE ... SELECT` Syntax”
Section 1.8.5.1, “`SELECT INTO TABLE` Differences”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 19.7, “Binary Logging of Stored Programs”
Section 12.10, “Cast Functions and Operators”
Section 14.3.9.2, “Consistent Nonlocking Reads”
Section C.5.8, “Known Issues in MySQL”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”
Section 16.4.2, “Replication Compatibility Between MySQL Versions”
Section 16.4.1.4, “Replication of `CREATE TABLE ... SELECT` Statements”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

CREATE TABLE ... SELECT ...

Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”
Section 18.3.1, “Management of `RANGE` and `LIST` Partitions”

CREATE TABLE IF NOT EXISTS

Section 13.1.17.1, “`CREATE TABLE ... SELECT` Syntax”
Section 16.4.1.3, “Replication of `CREATE ... IF NOT EXISTS` Statements”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

CREATE TABLE IF NOT EXISTS ... LIKE

Section 16.4.1.3, “Replication of `CREATE ... IF NOT EXISTS` Statements”

CREATE TABLE IF NOT EXISTS ... SELECT

Section 13.1.17.1, “`CREATE TABLE ... SELECT` Syntax”
Section 16.4.1.3, “Replication of `CREATE ... IF NOT EXISTS` Statements”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

CREATE TABLESPACE

Section 13.1.8, “ALTER TABLESPACE Syntax”
Section 13.1.17, “CREATE TABLE Syntax”
Section 13.1.18, “CREATE TABLESPACE Syntax”
Section 13.1.29, “DROP TABLESPACE Syntax”
Section 17.3.2.6, “Defining MySQL Cluster Data Nodes”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
Section 17.5.12.1, “MySQL Cluster Disk Data Objects”
Section 20.29.1, “The INFORMATION_SCHEMA FILES Table”

CREATE TEMPORARY TABLE

Section 13.7.1.3, “GRANT Syntax”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.3.3, “Statements That Cause an Implicit Commit”

CREATE TRIGGER

Section 13.1.19, “CREATE TRIGGER Syntax”
Section 13.7.5.13, “SHOW CREATE TRIGGER Syntax”
Section 19.7, “Binary Logging of Stored Programs”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 12.14, “Information Functions”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”
Section 16.4.1.6, “Replication of CURRENT_USER ()”
Section 16.4.1.10, “Replication of Invoked Features”
Section E.1, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 19.3.1, “Trigger Syntax”

CREATE USER

Section 13.7.1.1, “CREATE USER Syntax”
Section 13.7.6.3, “FLUSH Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 6.3.2, “Adding User Accounts”
Section 6.3.5, “Assigning Account Passwords”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 5.1.9.3, “Connecting Using the IPv6 Local Host Address”
Section 16.1.1.3, “Creating a User for Replication”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 8.11.4.1, “How MySQL Uses Memory”

Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”
Section 6.3.6.2.1, “Installing the PAM Authentication Plugin”
Section 6.3.6.3.1, “Installing the Windows Authentication Plugin”
Section 5.1.9, “IPv6 Support”
Section 5.2.4.4, “Logging Format for Changes to mysql Database Tables”
Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”
Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 6.1.2.3, “Passwords and Logging”
Section 6.3.6, “Pluggable Authentication”
Section 6.2.1, “Privileges Provided by MySQL”
Section 6.3.7, “Proxy Users”
Section 5.1.4, “Server System Variables”
Section 6.2.3, “Specifying Account Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.2, “The MySQL Access Privilege System”
Section 6.3.1, “User Names and Passwords”
Section 6.3.6.2.2, “Using the PAM Authentication Plugin”
Section 6.3.6.3.2, “Using the Windows Authentication Plugin”

CREATE VIEW

Section 13.1.9, “ALTER VIEW Syntax”
Section 13.1.20, “CREATE VIEW Syntax”
Section 13.7.5.14, “SHOW CREATE VIEW Syntax”
Section 8.12.5.2, “General Thread States”
Section 12.14, “Information Functions”
Section 18.5.4, “Partitioning and Table-Level Locking”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.6, “Replication of CURRENT_USER ()”
Section E.5, “Restrictions on Views”
Section 9.2, “Schema Object Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 20.27, “The INFORMATION_SCHEMA VIEWS Table”
Section 19.5.3, “Updatable and Insertable Views”
Section 19.5.2, “View Processing Algorithms”
Section 19.5.1, “View Syntax”

D

[index top [4407]]

DEALLOCATE PREPARE

- Section 13.5.3, “DEALLOCATE PREPARE Syntax”
- Section 13.5.1, “PREPARE Syntax”
- Section E.1, “Restrictions on Stored Programs”
- Section 5.1.6, “Server Status Variables”
- Section 13.5, “SQL Syntax for Prepared Statements”

DECLARE

- Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
- Section 13.6.3, “DECLARE Syntax”
- Section 13.6.7.4, “SIGNAL Syntax”
- Section 13.6.6.2, “Cursor DECLARE Syntax”
- Section 13.6.4.1, “Local Variable DECLARE Syntax”
- Section 13.6.4, “Variables in Stored Programs”

DECLARE ... CONDITION

- Section 13.6.7.1, “DECLARE ... CONDITION Syntax”
- Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
- Section 13.6.7.4, “SIGNAL Syntax”
- Section 13.6.7, “Condition Handling”

DECLARE ... HANDLER

- Section 13.6.7.1, “DECLARE ... CONDITION Syntax”
- Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
- Section 13.6.7, “Condition Handling”
- Section 13.6.7.4.2, “Effect of Signals on Handlers, Cursors, and Statements”

DELETE

- Section 4.5.1.1, “mysql Options”
- Section 13.1.7.1, “ALTER TABLE Partition Operations”
- Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
- Section 13.1.19, “CREATE TRIGGER Syntax”
- Section 13.1.20, “CREATE VIEW Syntax”
- Section 13.2.2, “DELETE Syntax”
- Section 14.11.3, “FEDERATED Storage Engine Notes and Tips”
- Section 14.3.5.4, “FOREIGN KEY Constraints”
- Section 13.7.1.3, “GRANT Syntax”
- Chapter 20, *INFORMATION_SCHEMA Tables*
- Section 14.3.5.5, “InnoDB and MySQL Replication”
- Section 14.3.4, “InnoDB Startup Options and System Variables”
- Section 13.2.9.2, “JOIN Syntax”
- Section 13.7.6.4, “KILL Syntax”
- Section 18.2.2, “LIST Partitioning”
- Section 14.10.2, “MERGE Table Problems”
- Section 22.8.3.1, “mysql_affected_rows()”
- Section 22.8.3.48, “mysql_num_rows()”
- Section 22.8.7.10, “mysql_stmt_execute()”

- Section 22.8.7.13, “mysql_stmt_field_count()”
- Section 22.8.7.18, “mysql_stmt_num_rows()”
- Section 18.2.1, “RANGE Partitioning”
- Section 13.7.1.5, “REVOKE Syntax”
- Section 13.3.6, “SET TRANSACTION Syntax”
- Section 13.1.33, “TRUNCATE TABLE Syntax”
- Section 6.3.2, “Adding User Accounts”
- Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
- Section 16.1.3.4, “Binary Log Options and Variables”
- Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
- Section 22.8.2, “C API Function Overview”
- Section 22.8.6, “C API Prepared Statement Function Overview”
- Section 6.2.7, “Causes of Access-Denied Errors”
- Section 14.3.9.2, “Consistent Nonlocking Reads”
- Section C.5.5.6, “Deleting Rows from Related Tables”
- Section 17.4.8, “ndb_delete_all — Delete All Rows from an NDB Table”
- Section 14.3.7.2, “Forcing InnoDB Recovery”
- Section 1.8.5.4, “Foreign Key Differences”
- Chapter 12, *Functions and Operators*
- Section 8.12.5.2, “General Thread States”
- Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”
- Section 8.9.3.1, “How the Query Cache Operates”
- Section 12.14, “Information Functions”
- Section 8.10.1, “Internal Locking Methods”
- Section C.5.8, “Known Issues in MySQL”
- Section 17.1.6.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”
- Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”
- Section 14.3.9.6, “Locks Set by Different SQL Statements in InnoDB”
- Section 5.2.4.4, “Logging Format for Changes to mysql Database Tables”
- Section 18.3.1, “Management of RANGE and LIST Partitions”
- Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”
- Section 17.5.12.1, “MySQL Cluster Disk Data Objects”
- Section 1.8.4, “MySQL Extensions to Standard SQL”
- Section 8.2.1, “Optimizing SELECT Statements”
- Section 8.2.2, “Optimizing DML Statements”
- Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
- Section 6.2.2, “Privilege System Grant Tables”
- Section 6.2.1, “Privileges Provided by MySQL”
- Section 8.12.5.4, “Query Cache Thread States”
- Section 16.4.1.14, “Replication and LIMIT”
- Section 16.4.1.20, “Replication and MEMORY Tables”
- Section 16.4.1.23, “Replication and the Query Optimizer”
- Section 16.1.3.3, “Replication Slave Options and Variables”

Section 9.3, “Reserved Words”
Section E.5, “Restrictions on Views”
Section 13.2.10.11, “Rewriting Subqueries as Joins”
Section 3.3.4.1, “Selecting All Data”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.3, “Server Command Options”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 8.2.2.3, “Speed of `DELETE` Statements”
Section 5.4.1.12, “Statement Probes”
Section 13.2.10.9, “Subquery Errors”
Section 13.2.10, “Subquery Syntax”
Section 8.10.2, “Table Locking Issues”
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 14.9, “The `BLACKHOLE` Storage Engine”
Section 20.27, “The `INFORMATION_SCHEMA VIEWS` Table”
Section 14.6, “The `MEMORY` Storage Engine”
Section 14.10, “The `MERGE` Storage Engine”
Section 5.2.4, “The Binary Log”
Section 6.2, “The MySQL Access Privilege System”
Section 19.3.1, “Trigger Syntax”
Section 19.5.3, “Updatable and Insertable Views”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 16.1.2.2, “Usage of Row-Based Logging and Replication”
Section 3.6.9, “Using `AUTO_INCREMENT`”
Section 4.5.1.6.2, “Using the `--safe-updates` Option”
Section 19.3, “Using Triggers”
Section 22.8.11.2, “What Results You Can Get from a Query”
Section 6.2.6, “When Privilege Changes Take Effect”
Section 22.8.11.1, “Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success”
Section 23.2.4.8, “Writing Audit Plugins”

DELETE FROM ... WHERE ...

Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”

DELETE FROM `t1,t2`

Section 5.4.1.12, “Statement Probes”

DESCRIBE

Section 13.1.17, “`CREATE TABLE` Syntax”
Section 13.8.1, “`DESCRIBE` Syntax”
Section 13.8.2, “`EXPLAIN` Syntax”
Section 22.8.7.28, “`mysql_stmt_store_result()`”
Section 22.8.3.69, “`mysql_store_result()`”
Section 22.8.3.71, “`mysql_use_result()`”
Section 13.7.5.6, “`SHOW COLUMNS` Syntax”

Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”
Section 3.3.2, “Creating a Table”
Section 20.31, “Extensions to `SHOW` Statements”
Section 3.4, “Getting Information About Databases and Tables”
Section 8.2.1.3, “Optimizing `LIMIT` Queries”
Section 8.11.3, “Optimizing Disk I/O”
Section 8.8.1, “Optimizing Queries with `EXPLAIN`”
Section 13.1.17.2, “Silent Column Specification Changes”
Section 3.6.6, “Using Foreign Keys”
Section 10.1.12, “UTF-8 for Metadata”

DO

Section 13.1.2, “`ALTER EVENT` Syntax”
Section 13.1.11, “`CREATE EVENT` Syntax”
Section 13.2.3, “`DO` Syntax”
Section 19.7, “Binary Logging of Stored Programs”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”
Section 12.15, “Miscellaneous Functions”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section E.1, “Restrictions on Stored Programs”
Section 13.2.10, “Subquery Syntax”
Section 20.7, “The `INFORMATION_SCHEMA EVENTS` Table”

DROP DATABASE

Section 13.1.21, “`DROP DATABASE` Syntax”
Section 22.8.3.11, “`mysql_drop_db()`”
Section 22.8.2, “C API Function Overview”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.1, “Dumping Data in SQL Format with `mysqldump`”
Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 8.9.3.1, “How the Query Cache Operates”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section E.10.5, “Windows Platform Limitations”

DROP DATABASE IF EXISTS

Section 16.4.1.7, “Replication of `DROP ... IF EXISTS` Statements”

DROP EVENT

- Section 13.1.22, “`DROP EVENT` Syntax”
- Section 19.7, “Binary Logging of Stored Programs”
- Section 19.4.3, “Event Syntax”
- Section 16.4.1.10, “Replication of Invoked Features”
- Section 13.3.3, “Statements That Cause an Implicit Commit”
- Section 19.4.6, “The Event Scheduler and MySQL Privileges”

DROP FUNCTION

- Section 13.1.4, “`ALTER FUNCTION` Syntax”
- Section 13.7.3.1, “`CREATE FUNCTION` Syntax for User-Defined Functions”
- Section 13.1.23, “`DROP FUNCTION` Syntax”
- Section 13.7.3.2, “`DROP FUNCTION` Syntax”
- Section 13.1.26, “`DROP PROCEDURE` and `DROP FUNCTION` Syntax”
- Section 23.3, “Adding New Functions to MySQL”
- Section 19.7, “Binary Logging of Stored Programs”
- Section 23.3.2.5, “Compiling and Installing User-Defined Functions”
- Section 9.2.4, “Function Name Parsing and Resolution”
- Section 16.4.1.10, “Replication of Invoked Features”
- Section 13.3.3, “Statements That Cause an Implicit Commit”
- Section 19.2.1, “Stored Routine Syntax”
- Section 2.12.1, “Upgrading MySQL”
- Section 23.3.2.6, “User-Defined Function Security Precautions”

DROP INDEX

- Section 13.1.7.2, “`ALTER TABLE` Online Operations”
- Section 13.1.7, “`ALTER TABLE` Syntax”
- Section 13.1.24, “`DROP INDEX` Syntax”
- Section 12.17.6.1, “Creating Spatial Indexes”
- Section 1.8.4, “MySQL Extensions to Standard SQL”
- Section 13.3.3, “Statements That Cause an Implicit Commit”

DROP LOGFILE GROUP

- Section 13.1.25, “`DROP LOGFILE GROUP` Syntax”
- Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”

DROP PROCEDURE

- Section 13.1.5, “`ALTER PROCEDURE` Syntax”
- Section 13.1.26, “`DROP PROCEDURE` and `DROP FUNCTION` Syntax”
- Section 19.7, “Binary Logging of Stored Programs”
- Section 16.4.1.10, “Replication of Invoked Features”
- Section 13.3.3, “Statements That Cause an Implicit Commit”

- Section 19.2.1, “Stored Routine Syntax”

DROP SCHEMA

- Section 13.1.21, “`DROP DATABASE` Syntax”
- Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
- Section 5.1.4, “Server System Variables”

DROP SERVER

- Section 13.1.27, “`DROP SERVER` Syntax”
- Section 13.7.6.3, “`FLUSH` Syntax”
- Section 8.11.4.1, “How MySQL Uses Memory”
- Section 16.4.1.5, “Replication of `CREATE SERVER`, `ALTER SERVER`, and `DROP SERVER`”

DROP TABLE

- Section 4.5.1.1, “`mysql` Options”
- Section 13.1.7, “`ALTER TABLE` Syntax”
- Section 13.1.19, “`CREATE TRIGGER` Syntax”
- Section 13.1.28, “`DROP TABLE` Syntax”
- Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”
- Section 14.3.5.4, “`FOREIGN KEY` Constraints”
- Section 13.3.5, “`LOCK TABLES` and `UNLOCK TABLES` Syntax”
- Section 14.10.2, “`MERGE` Table Problems”
- Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”
- Section 13.7.5.41, “`SHOW WARNINGS` Syntax”
- Section 13.6.7.4, “`SIGNAL` Syntax”
- Section 13.4.2.5, “`START SLAVE` Syntax”
- Section 13.1.33, “`TRUNCATE TABLE` Syntax”
- Section 13.7.3.4, “`UNINSTALL PLUGIN` Syntax”
- Section 17.5.13.1, “Adding MySQL Cluster Data Nodes Online: General Issues”
- Section 17.5.2, “Commands in the MySQL Cluster Management Client”
- Section 14.3.9.2, “Consistent Nonlocking Reads”
- Section 17.4.10, “`ndb_drop_index` — Drop Index from an NDB Table”
- Section 17.4.11, “`ndb_drop_table` — Drop an NDB Table”
- Section 4.5.4, “`mysqldump` — A Database Backup Program”
- Section 23.2.3.2, “Full-Text Parser Plugins”
- Section 8.9.3.1, “How the Query Cache Operates”
- Section 12.14, “Information Functions”
- Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
- Section 17.1.6.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”
- Section 1.8.4, “MySQL Extensions to Standard SQL”
- Section 8.5.6, “Optimizing InnoDB DDL Operations”
- Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”
- Section E.5, “Restrictions on Views”

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.4, “Server System Variables”
Section 5.2.4.2, “Setting The Binary Log Format”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 14.6, “The `MEMORY` Storage Engine”
Section 14.10, “The `MERGE` Storage Engine”
Section 14.3.14.4, “Troubleshooting `InnoDB` Data Dictionary Operations”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

DROP TABLE IF EXISTS

Section 16.4.1.7, “Replication of `DROP ... IF EXISTS` Statements”

DROP TABLE IF EXISTS

```
mysql.user mysql.db
mysql.tables_priv
mysql.columns_priv
mysql.procs_priv
```

Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”

DROP TABLE t1

Section 8.10.4, “Metadata Locking Within Transactions”

DROP TABLESPACE

Section 13.1.29, “`DROP TABLESPACE` Syntax”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”

DROP TEMPORARY TABLE IF EXISTS

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

DROP TRIGGER

Section 13.1.30, “`DROP TRIGGER` Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 16.4.1.10, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 19.3.1, “Trigger Syntax”

DROP USER

Section 13.7.1.2, “`DROP USER` Syntax”
Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.7.1.3, “`GRANT` Syntax”
Section 13.7.1.5, “`REVOKE` Syntax”

Section 8.11.4.1, “How MySQL Uses Memory”
Section 12.14, “Information Functions”
Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”
Section 6.2.1, “Privileges Provided by MySQL”
Section 6.3.3, “Removing User Accounts”
Section 16.4.1.6, “Replication of `CURRENT_USER()`”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 19.4.6, “The Event Scheduler and MySQL Privileges”

DROP USER 'x'@'localhost'

Section 23.2.4.9.3, “Using the Authentication Plugins”

DROP VIEW

Section 13.1.31, “`DROP VIEW` Syntax”
Section E.5, “Restrictions on Views”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 19.5.1, “View Syntax”

DROP VIEW IF EXISTS

Section 16.4.1.7, “Replication of `DROP ... IF EXISTS` Statements”

E

[[index top \[4407\]](#)]

EXECUTE

Section 13.2.1, “`CALL` Syntax”
Section 13.5.2, “`EXECUTE` Syntax”
Section 13.5.1, “`PREPARE` Syntax”
Section 22.8.16, “C API Support for Prepared `CALL` Statements”
Section E.1, “Restrictions on Stored Programs”
Section E.5, “Restrictions on Views”
Section 5.1.6, “Server Status Variables”
Section 13.5, “SQL Syntax for Prepared Statements”

EXPLAIN

Section 13.1.7, “`ALTER TABLE` Syntax”
Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 8.13.11, “`DISTINCT` Optimization”
Section 8.8.3, “`EXPLAIN EXTENDED` Output Format”
Section 8.8.2, “`EXPLAIN` Output Format”
Section 13.8.2, “`EXPLAIN` Syntax”
Chapter 20, *INFORMATION_SCHEMA Tables*
Section 8.13.4, “`IS NULL` Optimization”

Section 22.8.7.28, “mysql_stmt_store_result()”
Section 22.8.3.69, “mysql_store_result()”
Section 22.8.3.71, “mysql_use_result()”
Section 8.13.9, “ORDER BY Optimization”
Section 13.2.9, “SELECT Syntax”
Section 13.7.5.41, “SHOW WARNINGS Syntax”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”
Section 23.4.1, “Debugging a MySQL Server”
Section 8.13.3, “Engine Condition Pushdown Optimization”
Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”
Section 8.2.1.4, “How to Avoid Full Table Scans”
Section 13.2.9.3, “Index Hint Syntax”
Section 8.13.2, “Index Merge Optimization”
Section 8.13, “Internal Details of MySQL Optimizations”
Section 8.13.10.1, “Loose Index Scan”
Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.3.4.3, “MySQL Cluster System Variables”
Chapter 21, *MySQL Performance Schema*
Section 18.3.4, “Obtaining Information About Partitions”
Section C.5.6, “Optimizer-Related Issues”
Section 8.2.4, “Optimizing INFORMATION_SCHEMA Queries”
Section 8.2.1, “Optimizing SELECT Statements”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 13.2.10.10, “Optimizing Subqueries”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”
Section E.1, “Restrictions on Stored Programs”
Section C.5.5.7, “Solving Problems with No Matching Rows”
Section 8.2.1.1, “Speed of SELECT Statements”
Section 13.2.10.8, “Subqueries in the FROM Clause”
Section 8.13.2.1, “The Index Merge Intersection Access Algorithm”
Section 8.13.1.2, “The Range Access Method for Multiple-Part Indexes”
Section 8.8, “Understanding the Query Execution Plan”
Section 12.17.6.2, “Using a Spatial Index”
Section 23.4.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 8.3.6, “Verifying Index Usage”

EXPLAIN ... SELECT

Section 18.3.4, “Obtaining Information About Partitions”

EXPLAIN EXTENDED

Section 8.8.3, “EXPLAIN EXTENDED Output Format”
Section 8.8.2, “EXPLAIN Output Format”

Section 13.8.2, “EXPLAIN Syntax”
Section 13.7.5.41, “SHOW WARNINGS Syntax”
Section 8.13.3, “Engine Condition Pushdown Optimization”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”

EXPLAIN PARTITIONS

Section 13.8.2, “EXPLAIN Syntax”
Section 18.3.4, “Obtaining Information About Partitions”
Section 8.8.1, “Optimizing Queries with EXPLAIN”

EXPLAIN PARTITIONS SELECT

Section 18.3.4, “Obtaining Information About Partitions”

EXPLAIN PARTITIONS SELECT COUNT ()

Section 18.2.1, “RANGE Partitioning”

EXPLAIN SELECT

Section 8.8.2, “EXPLAIN Output Format”
Section 14.3.9.9, “How to Cope with Deadlocks”
Section 1.7, “How to Report Bugs or Problems”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 18.3.4, “Obtaining Information About Partitions”
Section 13.2.10.8, “Subqueries in the FROM Clause”

EXPLAIN SELECT ... ORDER BY

Section 8.13.9, “ORDER BY Optimization”

EXPLAIN tbl_name

Section 13.8.2, “EXPLAIN Syntax”
Section 8.8.1, “Optimizing Queries with EXPLAIN”

F

[index top [4407]]

FETCH

Section 13.6.6.2, “Cursor DECLARE Syntax”
Section 13.6.6.3, “Cursor FETCH Syntax”
Section E.1, “Restrictions on Stored Programs”

FETCH ... INTO var_list

Section 13.6.4, “Variables in Stored Programs”

FLUSH

Section 13.7.6.3, “FLUSH Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 13.7.6.6, “RESET Syntax”

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.3.1, “Establishing a Backup Policy”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.12, “Replication and `FLUSH`”
Section C.5.4.1.3, “Resetting the Root Password: Generic Instructions”
Section C.5.4.1.2, “Resetting the Root Password: Unix Systems”
Section C.5.4.1.1, “Resetting the Root Password: Windows Systems”
Section E.1, “Restrictions on Stored Programs”
Section 2.11.2, “Securing the Initial MySQL Accounts”
Section 13.3.3, “Statements That Cause an Implicit Commit”

FLUSH BINARY LOGS

Section 5.2.6, “Server Log Maintenance”

FLUSH DES_KEY_FILE

Section 12.13, “Encryption and Compression Functions”

FLUSH HOSTS

Section C.5.2.6, “Host ‘`host_name`’ is blocked”
Section 22.8.3.55, “`mysql_refresh()`”
Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”
Section 5.1.4, “Server System Variables”

FLUSH LOGS

Section 13.7.6.3, “`FLUSH` Syntax”
Section 22.8.3.55, “`mysql_refresh()`”
Section 7.3.3, “Backup Strategy Summary”
Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 5.2, “MySQL Server Logs”
Section 16.4.1.12, “Replication and `FLUSH`”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.2.6, “Server Log Maintenance”
Section 5.1.11, “Server Response to Signals”
Section 5.1.6, “Server Status Variables”
Section 5.2.2, “The Error Log”
Section 16.2.2.1, “The Slave Relay Log”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

FLUSH MASTER

Section 13.7.6.3, “`FLUSH` Syntax”
Section 16.4.1.12, “Replication and `FLUSH`”

FLUSH PRIVILEGES

Section 13.7.6.3, “`FLUSH` Syntax”

Section 22.8.3.55, “`mysql_refresh()`”
Section 22.8.3.56, “`mysql_reload()`”
Section 6.3.2, “Adding User Accounts”
Section 6.3.5, “Assigning Account Passwords”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.5.14, “Distributed MySQL Privileges for MySQL Cluster”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 17.5.11.3, “MySQL Cluster and MySQL Security Procedures”
Section 6.2.2, “Privilege System Grant Tables”
Section 16.4.1.12, “Replication and `FLUSH`”
Section 5.1.3, “Server Command Options”
Section 5.1.11, “Server Response to Signals”
Section 6.3.4, “Setting Account Resource Limits”
Section 1.2, “Typographical and Syntax Conventions”
Section 6.2.6, “When Privilege Changes Take Effect”

FLUSH QUERY CACHE

Section 13.7.6.3, “`FLUSH` Syntax”
Section 8.9.3.4, “Query Cache Status and Maintenance”

FLUSH SLAVE

Section 13.7.6.3, “`FLUSH` Syntax”
Section 16.4.1.12, “Replication and `FLUSH`”

FLUSH STATUS

Section 5.1.6, “Server Status Variables”

FLUSH TABLE

Section 13.7.6.3, “`FLUSH` Syntax”
Section 8.6.2, “Bulk Data Loading for `MyISAM` Tables”

FLUSH TABLES

Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.2.4, “`HANDLER` Syntax”
Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 14.10.2, “`MERGE` Table Problems”
Section 22.8.3.55, “`mysql_refresh()`”
Section 13.5.4, “Automatic Prepared Statement Repreparation”
Section 8.6.2, “Bulk Data Loading for `MyISAM` Tables”
Section 4.6.3, “`myisamchk` — `MyISAM` Table-Maintenance Utility”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 8.12.5.2, “General Thread States”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”

Section 14.5.4.2, “Problems from Tables Not Being Closed Properly”
Section 8.9.3.4, “Query Cache Status and Maintenance”
Section 16.4.1.12, “Replication and `FLUSH`”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

FLUSH TABLES `tbl_list` WITH READ LOCK

Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

FLUSH TABLES `tbl_name ...` WITH READ LOCK

Section 13.7.6.3, “`FLUSH` Syntax”

FLUSH TABLES `tbl_name` WITH READ LOCK

Section 13.2.4, “`HANDLER` Syntax”

FLUSH TABLES WITH READ LOCK

Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.3.5, “`LOCK TABLES` and `UNLOCK TABLES` Syntax”
Section 13.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”
Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”
Section 7.2, “Database Backup Methods”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.3.1, “Establishing a Backup Policy”
Section 8.12.5.2, “General Thread States”
Section 13.3.5.1, “Interaction of Table Locking and Transactions”
Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 16.4.1.12, “Replication and `FLUSH`”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”

FLUSH USER_RESOURCES

Section 13.7.6.3, “`FLUSH` Syntax”
Section 6.3.4, “Setting Account Resource Limits”

G

[index top [4407]]

GRANT

Section 13.1.11, “`CREATE EVENT` Syntax”
Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 13.1.19, “`CREATE TRIGGER` Syntax”
Section 13.1.20, “`CREATE VIEW` Syntax”
Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.7.1.3, “`GRANT` Syntax”
Section 13.7.1.5, “`REVOKE` Syntax”
Section 13.7.1.6, “`SET PASSWORD` Syntax”
Section 13.7.5.22, “`SHOW GRANTS` Syntax”
Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 6.3.2, “Adding User Accounts”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 6.3.5, “Assigning Account Passwords”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 5.1.9.3, “Connecting Using the IPv6 Local Host Address”
Section 16.1.1.3, “Creating a User for Replication”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”
Section 12.14, “Information Functions”
Section 6.3.6.2.1, “Installing the PAM Authentication Plugin”
Section 6.3.6.3.1, “Installing the Windows Authentication Plugin”
Section 5.1.9, “IPv6 Support”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”
Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Section 8.2.3, “Optimizing Database Privileges”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 6.1.2.3, “Passwords and Logging”
Section 17.6.5, “Preparing the MySQL Cluster for Replication”
Section 6.2.2, “Privilege System Grant Tables”
Section 6.2.1, “Privileges Provided by MySQL”
Section 2.11.1.1, “Problems Running `mysql_install_db`”
Section 6.3.7, “Proxy Users”

Section 16.4.1.12, “Replication and `FLUSH`”
Section 16.4.1.6, “Replication of `CURRENT_USER()`”
Section 16.4.1.22, “Replication of the `mysql` System Database”
Section 6.1.1, “Security Guidelines”
Section 5.1.3, “Server Command Options”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 6.3.4, “Setting Account Resource Limits”
Section 6.2.3, “Specifying Account Names”
Section 6.3.8.4, “SSL Command Options”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 19.4.6, “The Event Scheduler and MySQL Privileges”
Section 6.2, “The MySQL Access Privilege System”
Section 6.3.1, “User Names and Passwords”
Section 6.3.8.3, “Using SSL Connections”
Section 6.3.8, “Using SSL for Secure Connections”
Section 6.3.6.2.2, “Using the PAM Authentication Plugin”
Section 6.3.6.3.2, “Using the Windows Authentication Plugin”
Section 6.2.6, “When Privilege Changes Take Effect”

GRANT ALL

Section 13.7.1.3, “`GRANT` Syntax”

GRANT EVENT

Section 19.4.6, “The Event Scheduler and MySQL Privileges”

GRANT USAGE

Section 13.7.1.3, “`GRANT` Syntax”
Section 6.3.5, “Assigning Account Passwords”
Section 6.3.4, “Setting Account Resource Limits”

H

[index top [4407]]

HANDLER

Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”
Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.2.4, “`HANDLER` Syntax”
Section 22.8.3.3, “`mysql_change_user()`”
Section 22.8.12, “Controlling Automatic Reconnection Behavior”
Section 1.8, “MySQL Standards Compliance”
Section 5.1.4, “Server System Variables”

HANDLER ... CLOSE

Section 13.7.5.25, “`SHOW OPEN TABLES` Syntax”

HANDLER ... OPEN

Section 13.7.5.25, “`SHOW OPEN TABLES` Syntax”

HANDLER ... READ

Section E.1, “Restrictions on Stored Programs”

HANDLER OPEN

Section 13.2.4, “`HANDLER` Syntax”
Section 13.1.33, “`TRUNCATE TABLE` Syntax”

HELP

Section 13.8.3, “`HELP` Syntax”
Section 5.1.10, “Server-Side Help”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”

I

[index top [4407]]

IF

Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”
Section 13.6.5.2, “`IF` Syntax”
Section 12.4, “Control Flow Functions”
Section 13.6.5, “Flow Control Statements”

INSERT

Section 4.5.1.1, “`mysql` Options”
Section 13.1.13, “`CREATE INDEX` Syntax”
Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 13.1.17.1, “`CREATE TABLE ... SELECT` Syntax”
Section 13.1.19, “`CREATE TRIGGER` Syntax”
Section 13.1.20, “`CREATE VIEW` Syntax”
Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”
Section 13.2.2, “`DELETE` Syntax”
Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”
Section 14.3.5.4, “`FOREIGN KEY` Constraints”
Section 13.7.1.3, “`GRANT` Syntax”
Chapter 20, *INFORMATION_SCHEMA Tables*
Section 14.3.13.1, “InnoDB Error Codes”
Section 14.3.9.4, “InnoDB Record, Gap, and Next-Key Locks”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 13.2.5.3, “`INSERT ... ON DUPLICATE KEY UPDATE` Syntax”
Section 13.2.5.1, “`INSERT ... SELECT` Syntax”
Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 13.2.5, “`INSERT` Syntax”

Section 18.2.2, “LIST Partitioning”

Section 13.2.6, “LOAD DATA INFILE Syntax”

Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”

Section 14.10.2, “MERGE Table Problems”

Section C.5.2.9, “MySQL server has gone away”

Section 22.8.3.1, “mysql_affected_rows()”

Section 22.8.3.37, “mysql_insert_id()”

Section 22.8.3.48, “mysql_num_rows()”

Section 22.8.7.10, “mysql_stmt_execute()”

Section 22.8.7.13, “mysql_stmt_field_count()”

Section 22.8.7.16, “mysql_stmt_insert_id()”

Section 22.8.7.18, “mysql_stmt_num_rows()”

Section 22.8.7.21, “mysql_stmt_prepare()”

Section 22.8.3.69, “mysql_store_result()”

Section 13.7.2.4, “OPTIMIZE TABLE Syntax”

Section 1.8.6.1, “PRIMARY KEY and UNIQUE Index Constraints”

Section 18.2.1, “RANGE Partitioning”

Section 13.2.8, “REPLACE Syntax”

Section 13.7.5.28, “SHOW PROCEDURE CODE Syntax”

Section 13.7.5.41, “SHOW WARNINGS Syntax”

Section 6.2.5, “Access Control, Stage 2: Request Verification”

Section 6.3.2, “Adding User Accounts”

Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”

Section 13.5.4, “Automatic Prepared Statement Repreparation”

Section 7.1, “Backup and Recovery Types”

Section 19.7, “Binary Logging of Stored Programs”

Section 8.5.4, “Bulk Data Loading for InnoDB Tables”

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”

Section 22.8.2, “C API Function Overview”

Section 22.8.6, “C API Prepared Statement Function Overview”

Section 22.8.13, “C API Support for Multiple Statement Execution”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 10.1.13, “Column Character Set Conversion”

Section 8.10.3, “Concurrent Inserts”

Section 14.3.5.3.2, “Configurable InnoDB Auto-Increment Locking”

Section 1.8.6.2, “Constraints on Invalid Data”

Section 14.11.2.1, “Creating a FEDERATED Table Using CONNECTION”

Section 11.5, “Data Type Default Values”

Section 11.1.2, “Date and Time Type Overview”

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

Section 7.3.1, “Establishing a Backup Policy”

Section 12.18.3, “Expression Handling”

Section 14.3.7.2, “Forcing InnoDB Recovery”

Section 1.8.5.4, “Foreign Key Differences”

Section 8.12.5.2, “General Thread States”

Section 8.9.3.1, “How the Query Cache Operates”

Section 22.8.11.3, “How to Get the Unique ID for the Last Inserted Row”

Section 12.14, “Information Functions”

Section 8.10.1, “Internal Locking Methods”

Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”

Section 3.3.3, “Loading Data into a Table”

Section 14.3.9.6, “Locks Set by Different SQL Statements in InnoDB”

Section 5.2.4.4, “Logging Format for Changes to mysql Database Tables”

Section 18.3.1, “Management of RANGE and LIST Partitions”

Section 12.15, “Miscellaneous Functions”

Section 17.5.12.1, “MySQL Cluster Disk Data Objects”

Section 17.2.5, “MySQL Cluster Example with Tables and Data”

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

Section 1.8.4, “MySQL Extensions to Standard SQL”

Section 17.5.15, “NDB API Statistics Counters and Variables”

Section 8.6.1, “Optimizing MyISAM Queries”

Section 8.2.2, “Optimizing DML Statements”

Section 11.2.6, “Out-of-Range and Overflow Handling”

Section 18.5.4, “Partitioning and Table-Level Locking”

Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”

Section 12.17.4.4, “Populating Spatial Columns”

Section 6.2.2, “Privilege System Grant Tables”

Section 6.2.1, “Privileges Provided by MySQL”

Section 8.12.5.4, “Query Cache Thread States”

Section 16.4.1.1, “Replication and AUTO_INCREMENT”

Section 16.4.1.27, “Replication and Server SQL Mode”

Section 16.4.1.13, “Replication and System Functions”

Section 16.4.1.35, “Replication and Variables”

Section 16.1.3.2, “Replication Master Options and Variables”

Section 16.2.3.3, “Replication Rule Application”

Section 18.5, “Restrictions and Limitations on Partitioning”

Section E.1, “Restrictions on Stored Programs”

Section 5.4.1.7, “Row-Level Probes”

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”

Section 5.1.3, “Server Command Options”

Section 5.1.7, “Server SQL Modes”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”
Section 16.4.1.26, “Slave Errors During Replication”
Section 8.2.2.1, “Speed of `INSERT` Statements”
Section 5.4.1.12, “Statement Probes”
Section 13.2.10, “Subquery Syntax”
Section 8.10.2, “Table Locking Issues”
Section 10.1.7.6, “The `_bin` and `binary` Collations”
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 14.9, “The `BLACKHOLE` Storage Engine”
Section 20.27, “The `INFORMATION_SCHEMA VIEWS` Table”
Section 14.10, “The `MERGE` Storage Engine”
Section 14.5, “The `MyISAM` Storage Engine”
Section 5.2.4, “The Binary Log”
Section 6.2, “The MySQL Access Privilege System”
Section 8.9.3, “The MySQL Query Cache”
Section 5.1.12, “The Shutdown Process”
Section 14.3.5.3.1, “Traditional `InnoDB` Auto-Increment Locking”
Section 19.3.1, “Trigger Syntax”
Section 19.5.3, “Updatable and Insertable Views”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 3.6.9, “Using `AUTO_INCREMENT`”
Section 14.3.5.1, “Using `InnoDB` Transactions”
Section 19.3, “Using Triggers”
Section 22.8.11.2, “What Results You Can Get from a Query”
Section 6.2.6, “When Privilege Changes Take Effect”
Section 22.8.11.1, “Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success”

INSERT ... ()

Section 5.4.1.12, “Statement Probes”

INSERT ... ON DUPLICATE KEY UPDATE

Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”
Section 13.2.5.3, “`INSERT ... ON DUPLICATE KEY UPDATE` Syntax”
Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 13.2.5, “`INSERT` Syntax”
Section 14.10.2, “`MERGE` Table Problems”
Section 22.8.3.1, “`mysql_affected_rows()`”
Section 22.8.3.37, “`mysql_insert_id()`”
Section 14.3.5.3.2, “Configurable `InnoDB` Auto-Increment Locking”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.14, “Information Functions”
Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”
Section 12.15, “Miscellaneous Functions”
Section 18.5.4, “Partitioning and Table-Level Locking”

INSERT ... SELECT

Section 13.1.17.1, “`CREATE TABLE ... SELECT` Syntax”
Section 13.2.5.3, “`INSERT ... ON DUPLICATE KEY UPDATE` Syntax”
Section 13.2.5.1, “`INSERT ... SELECT` Syntax”
Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 13.2.5, “`INSERT` Syntax”
Section 22.8.3.37, “`mysql_insert_id()`”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 8.10.3, “Concurrent Inserts”
Section 14.3.5.3.2, “Configurable `InnoDB` Auto-Increment Locking”
Section C.5.8, “Known Issues in MySQL”
Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 18.5.4, “Partitioning and Table-Level Locking”
Section 16.4.1.14, “Replication and `LIMIT`”
Section 5.1.4, “Server System Variables”
Section 5.4.1.12, “Statement Probes”
Section 5.2.4, “The Binary Log”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

INSERT ... SELECT ON DUPLICATE KEY UPDATE

Section 13.2.5.3, “`INSERT ... ON DUPLICATE KEY UPDATE` Syntax”
Section 13.2.5.1, “`INSERT ... SELECT` Syntax”

INSERT ... SET

Section 13.2.5, “`INSERT` Syntax”

INSERT ... VALUES

Section 13.2.5, “`INSERT` Syntax”
Section 22.8.3.35, “`mysql_info()`”

INSERT DELAYED

Section 13.1.7, “`ALTER TABLE` Syntax”
Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 13.2.5, “`INSERT` Syntax”
Section 13.7.6.4, “`KILL` Syntax”
Section 14.10.2, “`MERGE` Table Problems”
Section 8.6.2, “Bulk Data Loading for `MyISAM` Tables”
Section 8.12.5.3, “Delayed-Insert Thread States”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section C.5.8, “Known Issues in MySQL”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 23.1.1, “MySQL Threads”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 18.5.4, “Partitioning and Table-Level Locking”
Section 18.5, “Restrictions and Limitations on Partitioning”
Section E.1, “Restrictions on Stored Programs”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 8.10.2, “Table Locking Issues”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 14.8, “The ARCHIVE Storage Engine”
Section 14.6, “The MEMORY Storage Engine”
Section 1.8.5.3, “Transaction and Atomic Operation Differences”
Section 19.5.3, “Updatable and Insertable Views”

INSERT IGNORE

Section 13.1.17.1, “CREATE TABLE ... SELECT Syntax”
Section 1.8.6.3, “ENUM and SET Constraints”
Section 13.2.5, “INSERT Syntax”
Section 1.8.6.2, “Constraints on Invalid Data”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 12.14, “Information Functions”
Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”
Section 5.1.7, “Server SQL Modes”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

INSERT IGNORE ... SELECT

Section 13.2.5.1, “INSERT ... SELECT Syntax”

INSERT INTO ... SELECT

Section 13.1.11, “CREATE EVENT Syntax”
Section 13.2.5, “INSERT Syntax”
Section 1.8.5.1, “SELECT INTO TABLE Differences”
Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 14.3.9.2, “Consistent Nonlocking Reads”
Section 1.8.6.2, “Constraints on Invalid Data”
Section 16.4.1.4, “Replication of CREATE TABLE ... SELECT Statements”
Section 5.1.4, “Server System Variables”
Section 14.6, “The MEMORY Storage Engine”

INSERT INTO ... SELECT ...

Section 14.11.3, “FEDERATED Storage Engine Notes and Tips”
Section 22.8.3.35, “mysql_info()”
Section 22.8.11.2, “What Results You Can Get from a Query”

INSERT INTO ... SELECT FROM memory_table

Section 16.4.1.20, “Replication and MEMORY Tables”

INSTALL PLUGIN

Section 13.7.6.3, “FLUSH Syntax”
Section 13.7.3.3, “INSTALL PLUGIN Syntax”
Section 13.7.5.26, “SHOW PLUGINS Syntax”
Section 6.3.10.4, “Audit Log Plugin Options and Variables”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 6.3.10.1, “Installing the Audit Log Plugin”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 5.1.8.2, “Obtaining Server Plugin Information”
Section 6.3.6, “Pluggable Authentication”
Section 14.2.1, “Pluggable Storage Engine Architecture”
Section 23.2.2, “Plugin API Components”
Section 23.2.4.2, “Plugin Data Structures”
Section 16.3.8.1, “Semisynchronous Replication Administrative Interface”
Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.3, “Server Command Options”
Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”
Section 20.13, “The INFORMATION_SCHEMA PLUGINS Table”
Section 23.2, “The MySQL Plugin API”
Section 23.2.4.6, “Writing INFORMATION_SCHEMA Plugins”
Section 23.2.4.8, “Writing Audit Plugins”
Section 23.2.4.5, “Writing Daemon Plugins”
Section 23.2.4.4, “Writing Full-Text Parser Plugins”
Section 23.2.4.9.1, “Writing the Server-Side Authentication Plugin”

ITERATE

Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.6.5.3, “ITERATE Syntax”
Section 13.6.5, “Flow Control Statements”
Section 13.6.2, “Statement Label Syntax”

K

[[index top \[4407\]](#)]

KILL

Section 13.7.1.3, “[GRANT Syntax](#)”
Section 13.7.6.4, “[KILL Syntax](#)”
Section C.5.2.9, “[MySQL server has gone away](#)”
Section 22.8.3.38, “[mysql_kill\(\)](#)”
Section 13.7.5.30, “[SHOW PROCESSLIST Syntax](#)”
Section 4.6.15, “[mysql_zap — Kill Processes That Match a Pattern](#)”
Section 8.12.5.2, “[General Thread States](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 13.3.5.3, “[Table-Locking Restrictions and Conditions](#)”

KILL CONNECTION

Section 13.7.6.4, “[KILL Syntax](#)”
Section 13.4.2.6, “[STOP SLAVE Syntax](#)”
Section 5.1.12, “[The Shutdown Process](#)”

KILL QUERY

Section 13.7.6.4, “[KILL Syntax](#)”
Section 13.4.2.6, “[STOP SLAVE Syntax](#)”
Section 5.1.12, “[The Shutdown Process](#)”

L

[[index top \[4407\]](#)]

LEAVE

Section 13.6.7.2, “[DECLARE . . . HANDLER Syntax](#)”
Section 13.6.5.4, “[LEAVE Syntax](#)”
Section 13.6.5.5, “[LOOP Syntax](#)”
Section 13.6.5.7, “[RETURN Syntax](#)”
Section 13.6.5, “[Flow Control Statements](#)”
Section E.1, “[Restrictions on Stored Programs](#)”
Section 13.6.2, “[Statement Label Syntax](#)”

LOAD DATA

Section 13.1.19, “[CREATE TRIGGER Syntax](#)”
Section 14.3.5.4, “[FOREIGN KEY Constraints](#)”
Section 13.2.6, “[LOAD DATA INFILE Syntax](#)”
Section 13.2.7, “[LOAD XML Syntax](#)”
Section 8.10.3, “[Concurrent Inserts](#)”
Section 14.3.5.3.2, “[Configurable InnoDB Auto-Increment Locking](#)”
Section C.5.8, “[Known Issues in MySQL](#)”
Section 17.1.6.3, “[Limits Relating to Transaction Handling in MySQL Cluster](#)”
Section 3.3.3, “[Loading Data into a Table](#)”

Section 6.1.3, “[Making MySQL Secure Against Attackers](#)”
Section 18.5.4, “[Partitioning and Table-Level Locking](#)”
Section 18.5, “[Restrictions and Limitations on Partitioning](#)”
Section E.1, “[Restrictions on Stored Programs](#)”
Section 6.1.6, “[Security Issues with LOAD DATA LOCAL](#)”
Section 3.3.4.1, “[Selecting All Data](#)”
Section 5.1.3, “[Server Command Options](#)”
Section 5.1.4, “[Server System Variables](#)”
Section 11.4.4, “[The ENUM Type](#)”
Section 9.4, “[User-Defined Variables](#)”

LOAD DATA FROM MASTER

Section 17.1.6.3, “[Limits Relating to Transaction Handling in MySQL Cluster](#)”

LOAD DATA INFILE

Section 4.5.1.1, “[mysql Options](#)”
Section 4.6.7.1, “[mysqlbinlog Hex Dump Format](#)”
Section 13.2.6, “[LOAD DATA INFILE Syntax](#)”
Section 14.5.1, “[MyISAM Startup Options](#)”
Section 9.1.7, “[NULL Values](#)”
Section 13.2.9.1, “[SELECT . . . INTO Syntax](#)”
Section 13.7.5.41, “[SHOW WARNINGS Syntax](#)”
Section 6.3.10.3, “[Audit Log Plugin Logging Control](#)”
Section 6.3.10.5, “[Audit Log Plugin Restrictions](#)”
Section 16.3.1.2, “[Backing Up Raw Data from a Slave](#)”
Section 7.1, “[Backup and Recovery Types](#)”
Section 8.6.2, “[Bulk Data Loading for MyISAM Tables](#)”
Section 6.2.7, “[Causes of Access-Denied Errors](#)”
Section 8.10.3, “[Concurrent Inserts](#)”
Section 7.2, “[Database Backup Methods](#)”
Section 10.1.3.2, “[Database Character Set and Collation](#)”
Section 17.4.21, “[ndb_show_tables — Display List of NDB Tables](#)”
Section 4.6.7, “[mysqlbinlog — Utility for Processing Binary Log Files](#)”
Section 4.5.4, “[mysqldump — A Database Backup Program](#)”
Section 4.5.5, “[mysqlimport — A Data Import Program](#)”
Section 16.1.2.3, “[Determination of Safe and Unsafe Statements in Binary Logging](#)”
Section C.5.4.3, “[How MySQL Handles a Full Disk](#)”
Section 12.14, “[Information Functions](#)”
Section C.5.8, “[Known Issues in MySQL](#)”
Section 17.1.6.3, “[Limits Relating to Transaction Handling in MySQL Cluster](#)”
Section 5.2.4.4, “[Logging Format for Changes to mysql Database Tables](#)”
Section 1.8.4, “[MySQL Extensions to Standard SQL](#)”
Section 2.10.4, “[MySQL Source-Configuration Options](#)”
Section 11.2.6, “[Out-of-Range and Overflow Handling](#)”
Section 4.1, “[Overview of MySQL Programs](#)”

Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 6.2.1, “Privileges Provided by MySQL”
Section C.5.5.3, “Problems with `NULL` Values”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 16.4.1.15, “Replication and `LOAD DATA INFILE`”
Section 16.4.2, “Replication Compatibility Between MySQL Versions”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 8.12.5.7, “Replication Slave SQL Thread States”
Section E.7, “Restrictions on Character Sets”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 8.2.2.1, “Speed of `INSERT` Statements”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.2.10, “Subquery Syntax”
Section 14.6, “The `MEMORY` Storage Engine”
Section 6.2, “The MySQL Access Privilege System”
Section 13.2.10.1, “The Subquery as Scalar Operand”
Section C.5.4.4, “Where MySQL Stores Temporary Files”
Section E.10.5, “Windows Platform Limitations”

LOAD DATA INFILE ...

Section 22.8.3.35, “`mysql_info()`”
Section 22.8.11.2, “What Results You Can Get from a Query”

LOAD DATA LOCAL

Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 22.8.3.49, “`mysql_options()`”
Section 22.8.3.52, “`mysql_real_connect()`”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”

LOAD DATA LOCAL INFILE

Section 22.8.3.63,
“`mysql_set_local_infile_handler()`”
Section 22.8.2, “C API Function Overview”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

LOAD INDEX INTO CACHE

Section 13.7.6.2, “`CACHE INDEX` Syntax”
Section 13.7.6.5, “`LOAD INDEX INTO CACHE` Syntax”
Section 8.9.2.4, “Index Preloading”
Section 18.5, “Restrictions and Limitations on Partitioning”

Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 1.4, “What Is New in MySQL 5.5”

LOAD INDEX INTO CACHE ... IGNORE LEAVES

Section 13.7.6.5, “`LOAD INDEX INTO CACHE` Syntax”

LOAD TABLE FROM MASTER

Section 17.1.6.6, “Unsupported or Missing Features in MySQL Cluster”

LOAD XML

Section 13.2.7, “`LOAD XML` Syntax”

LOAD XML INFILE

Section 13.2.7, “`LOAD XML` Syntax”
Section 1.4, “What Is New in MySQL 5.5”

LOAD XML LOCAL

Section 13.2.7, “`LOAD XML` Syntax”

LOAD XML LOCAL INFILE

Section 13.2.7, “`LOAD XML` Syntax”

LOCK TABLE

Section 8.10.3, “Concurrent Inserts”
Section 8.12.5.2, “General Thread States”
Section C.5.7.1, “Problems with `ALTER TABLE`”

LOCK TABLES

Section 13.1.10, “`CREATE DATABASE` Syntax”
Section 13.1.17, “`CREATE TABLE` Syntax”
Section 13.1.19, “`CREATE TRIGGER` Syntax”
Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.7.1.3, “`GRANT` Syntax”
Section 14.3.4, “`InnoDB` Startup Options and System Variables”
Section 13.3.5, “`LOCK TABLES` and `UNLOCK TABLES` Syntax”
Section 13.3.5.2, “`LOCK TABLES` and Triggers”
Section 14.10.2, “`MERGE` Table Problems”
Section 13.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”
Section 8.6.2, “Bulk Data Loading for `MyISAM` Tables”
Section 14.3.9.8, “Deadlock Detection and Rollback”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 14.3.9.9, “How to Cope with Deadlocks”

Section 13.3.5.1, “Interaction of Table Locking and Transactions”
Section 8.10.1, “Internal Locking Methods”
Section 17.1.6.10, “Limitations Relating to Multiple MySQL Cluster Nodes”
Section 14.3.15, “Limits on InnoDB Tables”
Section 14.3.9.6, “Locks Set by Different SQL Statements in InnoDB”
Section 18.5.4, “Partitioning and Table-Level Locking”
Section 6.2.1, “Privileges Provided by MySQL”
Section 14.5.4.2, “Problems from Tables Not Being Closed Properly”
Section E.1, “Restrictions on Stored Programs”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 8.11.1, “System Factors and Startup Parameter Tuning”
Section 8.10.2, “Table Locking Issues”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 1.8.5.3, “Transaction and Atomic Operation Differences”

LOCK TABLES ... READ

Section 13.7.6.3, “FLUSH Syntax”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 14.3.15, “Limits on InnoDB Tables”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

LOCK TABLES ... WRITE

Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 14.3.15, “Limits on InnoDB Tables”

LOOP

Section 13.6.5.3, “ITERATE Syntax”
Section 13.6.5.4, “LEAVE Syntax”
Section 13.6.5.5, “LOOP Syntax”
Section 13.6.5, “Flow Control Statements”
Section 13.6.2, “Statement Label Syntax”

O

[index top [4407]]

OPEN

Section 13.6.6.4, “Cursor OPEN Syntax”

OPTIMIZE TABLE

Section 4.6.3.1, “myisamchk General Options”
Section 13.2.2, “DELETE Syntax”
Section 13.7.6.4, “KILL Syntax”
Section 14.10.2, “MERGE Table Problems”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 7.6.4, “MyISAM Table Optimization”
Section 13.7.2.4, “OPTIMIZE TABLE Syntax”
Section 17.5.13.2, “Adding MySQL Cluster Data Nodes Online: Basic procedure”
Section 17.5.13.3, “Adding MySQL Cluster Data Nodes Online: Detailed Example”
Section 23.4.1, “Debugging a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 14.5.3.2, “Dynamic Table Characteristics”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 8.12.5.2, “General Thread States”
Section C.5.4.3, “How MySQL Handles a Full Disk”
Section C.5.8, “Known Issues in MySQL”
Section 17.1.6.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”
Section 18.3.3, “Maintenance of Partitions”
Section 17.5.12.3, “MySQL Cluster Disk Data Storage Requirements”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.2.5, “Other Optimization Tips”
Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.12, “Replication and FLUSH”
Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 8.2.2.2, “Speed of UPDATE Statements”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 14.5.3.1, “Static (Fixed-Length) Table Characteristics”
Section 14.8, “The ARCHIVE Storage Engine”
Section 5.1.12, “The Shutdown Process”
Section 5.2.5, “The Slow Query Log”
Section 8.11.3.1.2, “Using Symbolic Links for MyISAM Tables on Unix”

P

[[index top \[4407\]](#)]

PREPARE

Section 13.2.1, “[CALL Syntax](#)”
Section 13.5.3, “[DEALLOCATE PREPARE Syntax](#)”
Section 13.5.2, “[EXECUTE Syntax](#)”
Section 13.5.1, “[PREPARE Syntax](#)”
Section 13.5.4, “[Automatic Prepared Statement Repreparation](#)”
Section 22.8.16, “[C API Support for Prepared CALL Statements](#)”
Section 9.2.2, “[Identifier Case Sensitivity](#)”
Section 8.10.4, “[Metadata Locking Within Transactions](#)”
Section E.1, “[Restrictions on Stored Programs](#)”
Section E.5, “[Restrictions on Views](#)”
Section 5.1.6, “[Server Status Variables](#)”
Section 13.5, “[SQL Syntax for Prepared Statements](#)”

PURGE BINARY LOGS

Section 13.7.1.3, “[GRANT Syntax](#)”
Section 13.4.1.1, “[PURGE BINARY LOGS Syntax](#)”
Section 13.4.1.2, “[RESET MASTER Syntax](#)”
Section 4.5.4, “[mysqldump — A Database Backup Program](#)”
Section 7.3.1, “[Establishing a Backup Policy](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 5.2.6, “[Server Log Maintenance](#)”
Section 5.1.4, “[Server System Variables](#)”
Section 5.2.4, “[The Binary Log](#)”

R

[[index top \[4407\]](#)]

RELEASE SAVEPOINT

Section 13.3.4, “[SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax](#)”

RENAME TABLE

Section 13.1.7, “[ALTER TABLE Syntax](#)”
Section 13.2.2, “[DELETE Syntax](#)”
Section 13.1.32, “[RENAME TABLE Syntax](#)”
Section 4.5.4, “[mysqldump — A Database Backup Program](#)”
Section 8.12.5.2, “[General Thread States](#)”
Section 9.2.2, “[Identifier Case Sensitivity](#)”
Section 1.8.4, “[MySQL Extensions to Standard SQL](#)”
Section 5.2.1, “[Selecting General Query and Slow Query Log Output Destinations](#)”
Section 13.3.3, “[Statements That Cause an Implicit Commit](#)”

Section 14.3.3, “[Using Per-Table Tablespaces](#)”
Section 8.11.3.1.2, “[Using Symbolic Links for MyISAM Tables on Unix](#)”

RENAME USER

Section 13.7.1.3, “[GRANT Syntax](#)”
Section 13.7.1.4, “[RENAME USER Syntax](#)”
Section 12.14, “[Information Functions](#)”
Section 5.2.4.4, “[Logging Format for Changes to mysql Database Tables](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 16.4.1.6, “[Replication of CURRENT_USER \(\)](#)”
Section 13.3.3, “[Statements That Cause an Implicit Commit](#)”
Section 19.4.6, “[The Event Scheduler and MySQL Privileges](#)”
Section 6.2.6, “[When Privilege Changes Take Effect](#)”

REPAIR TABLE

Section 4.6.3.1, “[myisamchk General Options](#)”
Section 13.1.7.1, “[ALTER TABLE Partition Operations](#)”
Section 13.1.7, “[ALTER TABLE Syntax](#)”
Section 13.7.6.4, “[KILL Syntax](#)”
Section 13.2.6, “[LOAD DATA INFILE Syntax](#)”
Section 14.10.2, “[MERGE Table Problems](#)”
Section 14.5.1, “[MyISAM Startup Options](#)”
Section 7.6, “[MyISAM Table Maintenance and Crash Recovery](#)”
Section 13.7.2.5, “[REPAIR TABLE Syntax](#)”
Section 14.5.4.1, “[Corrupted MyISAM Tables](#)”
Section 7.2, “[Database Backup Methods](#)”
Section 4.6.3, “[myisamchk — MyISAM Table-Maintenance Utility](#)”
Section 4.5.3, “[mysqlcheck — A Table Maintenance Program](#)”
Section 8.10.5, “[External Locking](#)”
Section 12.9.6, “[Fine-Tuning MySQL Full-Text Search](#)”
Section 8.12.5.2, “[General Thread States](#)”
Section C.5.4.3, “[How MySQL Handles a Full Disk](#)”
Section 7.6.3, “[How to Repair MyISAM Tables](#)”
Section 1.7, “[How to Report Bugs or Problems](#)”
Section C.5.8, “[Known Issues in MySQL](#)”
Section 18.3.3, “[Maintenance of Partitions](#)”
Section 1.8.4, “[MySQL Extensions to Standard SQL](#)”
Section 6.2.1, “[Privileges Provided by MySQL](#)”
Section 14.5.4.2, “[Problems from Tables Not Being Closed Properly](#)”
Section C.5.7.1, “[Problems with ALTER TABLE](#)”
Section 2.12.4, “[Rebuilding or Repairing Tables or Indexes](#)”
Section 16.4.1.12, “[Replication and FLUSH](#)”
Section 16.4.1.17, “[Replication and REPAIR TABLE](#)”

Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.1.4, “Server System Variables”
Section 7.6.5, “Setting Up a `MyISAM` Table Maintenance Schedule”
Section 8.6.3, “Speed of `REPAIR TABLE` Statements”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 5.1.12, “The Shutdown Process”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”
Section 8.11.3.1.2, “Using Symbolic Links for `MyISAM` Tables on Unix”

REPEAT

Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”
Section 13.6.5.3, “`ITERATE` Syntax”
Section 13.6.5.4, “`LEAVE` Syntax”
Section 13.6.5.6, “`REPEAT` Syntax”
Section 19.1, “Defining Stored Programs”
Section 13.6.5, “Flow Control Statements”
Section 13.6.2, “Statement Label Syntax”

REPLACE

Section 13.1.17.1, “`CREATE TABLE ... SELECT` Syntax”
Section 13.1.19, “`CREATE TRIGGER` Syntax”
Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 13.2.5, “`INSERT` Syntax”
Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 13.2.7, “`LOAD XML` Syntax”
Section 14.10.2, “`MERGE` Table Problems”
Section C.5.2.9, “MySQL server has gone away”
Section 22.8.3.1, “`mysql_affected_rows()`”
Section 13.2.8, “`REPLACE` Syntax”
Section 14.3.5.3.2, “Configurable `InnoDB` Auto-Increment Locking”
Section 11.5, “Data Type Default Values”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 12.14, “Information Functions”
Section C.5.8, “Known Issues in MySQL”
Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 18.5.4, “Partitioning and Table-Level Locking”
Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”
Section 5.1.3, “Server Command Options”

Section 13.2.10, “Subquery Syntax”
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

REPLACE ... SELECT

Section 14.3.5.3.2, “Configurable `InnoDB` Auto-Increment Locking”
Section C.5.8, “Known Issues in MySQL”

RESET

Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.7.6.6, “`RESET` Syntax”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 13.3.3, “Statements That Cause an Implicit Commit”

RESET MASTER

Section 13.7.6.3, “`FLUSH` Syntax”
Section 22.8.3.55, “`mysql_refresh()`”
Section 13.4.1.2, “`RESET MASTER` Syntax”
Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”
Section 17.6.3, “Known Issues in MySQL Cluster Replication”
Section 16.3.6, “Switching Masters During Failover”
Section 5.2.4, “The Binary Log”

RESET QUERY CACHE

Section 8.12.5.4, “Query Cache Thread States”

RESET SLAVE

Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”
Section 13.7.6.3, “`FLUSH` Syntax”
Section 22.8.3.55, “`mysql_refresh()`”
Section 13.4.1.2, “`RESET MASTER` Syntax”
Section 13.4.2.3, “`RESET SLAVE` Syntax”
Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”
Section 17.6.3, “Known Issues in MySQL Cluster Replication”

RESIGNAL

Section 13.6.7.1, “`DECLARE ... CONDITION` Syntax”
Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”
Section 13.6.7.3.1, “`RESIGNAL` Alone”
Section 13.6.7.3.4, “`RESIGNAL` Requires an Active Handler”
Section 13.6.7.3, “`RESIGNAL` Syntax”
Section 13.6.7.3.3, “`RESIGNAL` with a Condition Value and Optional New Signal Information”
Section 13.6.7.3.2, “`RESIGNAL` with New Signal Information”
Section 13.6.7, “Condition Handling”
Section E.2, “Restrictions on Condition Handling”

Section E.1, “Restrictions on Stored Programs”
Section 13.6.7.4.1, “Signal Condition Information Items”
Section 1.4, “What Is New in MySQL 5.5”

RETURN

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.6.5.5, “LOOP Syntax”
Section 13.6.5.7, “RETURN Syntax”
Section 13.6.5, “Flow Control Statements”
Section E.1, “Restrictions on Stored Programs”

REVOKE

Section 13.7.6.3, “FLUSH Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 13.7.1.5, “REVOKE Syntax”
Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 12.14, “Information Functions”
Section 5.1.9, “IPv6 Support”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”
Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”
Section 1.8.5, “MySQL Differences from Standard SQL”
Section 6.2.2, “Privilege System Grant Tables”
Section 6.2.1, “Privileges Provided by MySQL”
Section 2.11.1.1, “Problems Running `mysql_install_db`”
Section 6.3.7, “Proxy Users”
Section 16.4.1.6, “Replication of `CURRENT_USER()`”
Section 16.4.1.22, “Replication of the `mysql` System Database”
Section 6.1.1, “Security Guidelines”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 19.4.6, “The Event Scheduler and MySQL Privileges”
Section 6.2, “The MySQL Access Privilege System”
Section 6.3.1, “User Names and Passwords”
Section 6.2.6, “When Privilege Changes Take Effect”

REVOKE ALL PRIVILEGES

Section 13.7.1.3, “GRANT Syntax”
Section 6.2.1, “Privileges Provided by MySQL”

ROLLBACK

Section 14.3.13, “InnoDB Error Handling”
Section 22.8.3.3, “`mysql_change_user()`”

Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 19.7, “Binary Logging of Stored Programs”
Section 14.3.9.8, “Deadlock Detection and Rollback”
Section 12.14, “Information Functions”
Section 13.3.5.1, “Interaction of Table Locking and Transactions”
Section 13.3, “MySQL Transactional and Locking Statements”
Section 16.4.1.31, “Replication and Transactions”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section C.5.5.5, “Rollback Failure for Nontransactional Tables”
Section 5.1.4, “Server System Variables”
Section 13.3.2, “Statements That Cannot Be Rolled Back”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 14.3.9, “The InnoDB Transaction Model and Locking”
Section 5.2.4, “The Binary Log”
Section 1.8.5.3, “Transaction and Atomic Operation Differences”
Section 19.3.1, “Trigger Syntax”
Section 14.3.5.1, “Using InnoDB Transactions”

ROLLBACK TO SAVEPOINT

Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax”

S

[index top [4407]]

SAVEPOINT

Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax”

SELECT

Section 4.5.1.1, “`mysql` Options”
Section 13.1.7, “ALTER TABLE Syntax”
Section 13.1.9, “ALTER VIEW Syntax”
Section 13.2.1, “CALL Syntax”
Section 13.1.11, “CREATE EVENT Syntax”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.17.1, “CREATE TABLE ... SELECT Syntax”
Section 13.1.17, “CREATE TABLE Syntax”
Section 13.1.20, “CREATE VIEW Syntax”

[Section 13.2.2, “DELETE Syntax”](#)
[Section 8.8.3, “EXPLAIN EXTENDED Output Format”](#)
[Section 8.8.2, “EXPLAIN Output Format”](#)
[Section 13.8.2, “EXPLAIN Syntax”](#)
[Section 14.11.3, “FEDERATED Storage Engine Notes and Tips”](#)
[Section 13.7.1.3, “GRANT Syntax”](#)
[Section 12.16.1, “GROUP BY \(Aggregate\) Functions”](#)
[Section 13.2.4, “HANDLER Syntax”](#)
[Chapter 20, *INFORMATION_SCHEMA Tables*](#)
[Section 14.3.4, “InnoDB Startup Options and System Variables”](#)
[Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#)
[Section 13.2.5.1, “INSERT ... SELECT Syntax”](#)
[Section 13.2.5.2, “INSERT DELAYED Syntax”](#)
[Section 13.2.5, “INSERT Syntax”](#)
[Section 13.2.9.2, “JOIN Syntax”](#)
[Section 13.7.6.4, “KILL Syntax”](#)
[Section 13.2.7, “LOAD XML Syntax”](#)
[Section 14.10.2, “MERGE Table Problems”](#)
[Section 7.6.4, “MyISAM Table Optimization”](#)
[Section 22.8.3.1, “mysql_affected_rows\(\)”](#)
[Section 22.8.3.17, “mysql_fetch_field\(\)”](#)
[Section 22.8.3.22, “mysql_field_count\(\)”](#)
[Section 22.8.3.47, “mysql_num_fields\(\)”](#)
[Section 22.8.3.48, “mysql_num_rows\(\)”](#)
[Section 22.8.7.10, “mysql_stmt_execute\(\)”](#)
[Section 22.8.7.11, “mysql_stmt_fetch\(\)”](#)
[Section 22.8.7.18, “mysql_stmt_num_rows\(\)”](#)
[Section 22.8.7.28, “mysql_stmt_store_result\(\)”](#)
[Section 22.8.3.69, “mysql_store_result\(\)”](#)
[Section 22.8.3.71, “mysql_use_result\(\)”](#)
[Section 18.2.3.1, “RANGE COLUMNS partitioning”](#)
[Section 13.2.8, “REPLACE Syntax”](#)
[Section 14.3.9.3, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”](#)
[Section 13.2.9.1, “SELECT ... INTO Syntax”](#)
[Section 13.2.9, “SELECT Syntax”](#)
[Section 13.3.6, “SET TRANSACTION Syntax”](#)
[Section 13.7.4, “SET Syntax”](#)
[Section 13.7.5.3, “SHOW BINLOG EVENTS Syntax”](#)
[Section 13.7.5.11, “SHOW CREATE PROCEDURE Syntax”](#)
[Section 13.7.5.14, “SHOW CREATE VIEW Syntax”](#)
[Section 13.7.5.18, “SHOW ERRORS Syntax”](#)
[Section 13.7.5.28, “SHOW PROCEDURE CODE Syntax”](#)
[Section 13.7.5.30, “SHOW PROCESSLIST Syntax”](#)
[Section 13.7.5.33, “SHOW RELAYLOG EVENTS Syntax”](#)
[Section 13.7.5.40, “SHOW VARIABLES Syntax”](#)
[Section 13.7.5.41, “SHOW WARNINGS Syntax”](#)
[Section 13.7.5, “SHOW Syntax”](#)
[Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#)
[Section 13.2.9.4, “UNION Syntax”](#)
[Section 13.2.11, “UPDATE Syntax”](#)
[Section 12.3.4, “Assignment Operators”](#)
[Section 13.5.4, “Automatic Prepared Statement Repreparation”](#)
[Section 14.3.9.5, “Avoiding the Phantom Problem Using Next-Key Locking”](#)
[Section 19.7, “Binary Logging of Stored Programs”](#)
[Section 8.6.2, “Bulk Data Loading for MyISAM Tables”](#)
[Section 22.8.1, “C API Data Structures”](#)
[Section 22.8.2, “C API Function Overview”](#)
[Section 22.8.6, “C API Prepared Statement Function Overview”](#)
[Section 22.8.13, “C API Support for Multiple Statement Execution”](#)
[Section 22.8.16, “C API Support for Prepared CALL Statements”](#)
[Section 12.3.2, “Comparison Functions and Operators”](#)
[Section 8.3.8, “Comparison of B-Tree and Hash Indexes”](#)
[Section 8.10.3, “Concurrent Inserts”](#)
[Section 14.3.5.3.2, “Configurable InnoDB Auto-Increment Locking”](#)
[Section 10.1.4, “Connection Character Sets and Collations”](#)
[Section 14.3.9.2, “Consistent Nonlocking Reads”](#)
[Section 14.11.2.1, “Creating a FEDERATED Table Using CONNECTION”](#)
[Section 3.3.1, “Creating and Selecting a Database”](#)
[Section 13.6.6.2, “Cursor DECLARE Syntax”](#)
[Section 13.6.6.3, “Cursor FETCH Syntax”](#)
[Section 14.3.9.8, “Deadlock Detection and Rollback”](#)
[Section 17.4.19, “ndb_select_all — Print Rows from an NDB Table”](#)
[Section 4.5.4, “mysqldump — A Database Backup Program”](#)
[Section 4.5.7, “mysqlslap — Load Emulation Client”](#)
[Section 8.4.3.2, “Disadvantages of Creating Many Tables in the Same Database”](#)
[Section 5.1.5.2, “Dynamic System Variables”](#)
[Section 3.2, “Entering Queries”](#)
[Section 19.4.2, “Event Scheduler Configuration”](#)
[Section 10.1.7.8, “Examples of the Effect of Collation”](#)
[Section 14.3.7.2, “Forcing InnoDB Recovery”](#)
[Section 1.8.5.4, “Foreign Key Differences”](#)
[Chapter 12, *Functions and Operators*](#)
[Section 8.12.5.2, “General Thread States”](#)
[Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”](#)
[Section 18.2.7, “How MySQL Partitioning Handles NULL”](#)
[Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”](#)
[Section 8.9.3.1, “How the Query Cache Operates”](#)
[Section 14.3.9.9, “How to Cope with Deadlocks”](#)
[Section 1.7, “How to Report Bugs or Problems”](#)
[Section 9.2.1, “Identifier Qualifiers”](#)
[Section 13.2.9.3, “Index Hint Syntax”](#)

Section 12.14, “Information Functions”
Section 8.10.1, “Internal Locking Methods”
Section C.5.8, “Known Issues in MySQL”
Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 14.3.9.6, “Locks Set by Different SQL Statements in InnoDB”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 18.3.1, “Management of `RANGE` and `LIST` Partitions”
Section 8.3.5, “Multiple-Column Indexes”
Section 17.5.12.1, “MySQL Cluster Disk Data Objects”
Section 17.2.5, “MySQL Cluster Example with Tables and Data”
Section 17.6.4, “MySQL Cluster Replication Schema and Tables”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Chapter 21, *MySQL Performance Schema*
Section 12.9.1, “Natural Language Full-Text Searches”
Section 18.3.4, “Obtaining Information About Partitions”
Section 8.3, “Optimization and Indexes”
Section C.5.6, “Optimizer-Related Issues”
Section 8.6.1, “Optimizing `MyISAM` Queries”
Section 8.2.1, “Optimizing `SELECT` Statements”
Section 8.8.1, “Optimizing Queries with `EXPLAIN`”
Section 8.13.12, “Optimizing Subqueries with `EXISTS` Strategy”
Section 4.6.3.4, “Other `myisamchk` Options”
Section 18.5.4, “Partitioning and Table-Level Locking”
Section 6.2.1, “Privileges Provided by MySQL”
Section C.5.5.2, “Problems Using `DATE` Columns”
Section C.5.5.8, “Problems with Floating-Point Values”
Section 8.9.3.2, “Query Cache `SELECT` Options”
Section 8.9.3.4, “Query Cache Status and Maintenance”
Section 8.12.5.4, “Query Cache Thread States”
Section 17.5.9, “Quick Reference: MySQL Cluster SQL Statements”
Section 14.7.1, “Repairing and Checking CSV Tables”
Section 16.2, “Replication Implementation”
Section 16.1.3.2, “Replication Master Options and Variables”
Section 16.4.1.3, “Replication of `CREATE ... IF NOT EXISTS` Statements”
Section 16.4.1.10, “Replication of Invoked Features”
Section 9.3, “Reserved Words”
Section E.1, “Restrictions on Stored Programs”
Section E.5, “Restrictions on Views”
Section 3.3.4, “Retrieving Information from a Table”
Section 3.6.7, “Searching on Two Keys”
Section 2.11.2, “Securing the Initial MySQL Accounts”
Section 3.3.4.1, “Selecting All Data”
Section 3.3.4.2, “Selecting Particular Rows”
Section 5.1.4, “Server System Variables”
Section C.5.5.7, “Solving Problems with No Matching Rows”
Section 8.2.1.1, “Speed of `SELECT` Statements”
Section 8.2.2.2, “Speed of `UPDATE` Statements”
Section 5.4.1.12, “Statement Probes”
Section 19.2.1, “Stored Routine Syntax”
Section 9.1.1, “String Literals”
Section 13.2.10.8, “Subqueries in the `FROM` Clause”
Section 13.2.10.6, “Subqueries with `EXISTS` or `NOT EXISTS`”
Section 13.2.10.9, “Subquery Errors”
Section 13.2.10, “Subquery Syntax”
Section 8.10.2, “Table Locking Issues”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 11.4.4, “The `ENUM` Type”
Section 20.4, “The `INFORMATION_SCHEMA COLUMNS` Table”
Section 20.7, “The `INFORMATION_SCHEMA EVENTS` Table”
Section 20.27, “The `INFORMATION_SCHEMA VIEWS` Table”
Section 14.10, “The `MERGE` Storage Engine”
Section 17.5.10.13, “The `ndbinfo nodes` Table”
Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”
Section 5.2.4, “The Binary Log”
Section 6.2, “The MySQL Access Privilege System”
Section 8.9.3, “The MySQL Query Cache”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”
Section 13.2.10.1, “The Subquery as Scalar Operand”
Section 19.3.1, “Trigger Syntax”
Section 1.2, “Typographical and Syntax Conventions”
Section 2.11.1, “Unix Postinstallation Procedures”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”
Section 9.4, “User-Defined Variables”
Section 14.3.5.1, “Using `InnoDB` Transactions”
Section 8.4.2.4, “Using `PROCEDURE ANALYSE`”
Section 12.17.6.2, “Using a Spatial Index”
Section 23.4.1.6, “Using Server Logs to Find Causes of Errors in `mysqld`”
Section 5.1.5, “Using System Variables”
Section 4.5.1.6.2, “Using the `--safe-updates` Option”
Section 10.1.12, “UTF-8 for Metadata”
Section 19.5.1, “View Syntax”
Section C.5.4.4, “Where MySQL Stores Temporary Files”
Section 23.2.4.8, “Writing Audit Plugins”

SELECT *

Section 11.4.3, “The `BLOB` and `TEXT` Types”

SELECT * INTO OUTFILE

'file_name' FROM tbl_name

Section 7.2, “Database Backup Methods”

SELECT ... FOR UPDATE

Section 14.3.9.1, “InnoDB Lock Modes”

Section 14.3.9.3, “`SELECT ... FOR UPDATE` and

`SELECT ... LOCK IN SHARE MODE` Locking Reads”

Section 14.3.9.9, “How to Cope with Deadlocks”

Section 14.3.9.6, “Locks Set by Different SQL

Statements in InnoDB”

SELECT ... FROM

Section 14.3.9.6, “Locks Set by Different SQL

Statements in InnoDB”

SELECT ... FROM ... FOR

UPDATE

Section 14.3.9.6, “Locks Set by Different SQL

Statements in InnoDB”

SELECT ... FROM ... LOCK IN

SHARE MODE

Section 14.3.9.6, “Locks Set by Different SQL

Statements in InnoDB”

SELECT ... INTO

Section 13.1.11, “`CREATE EVENT` Syntax”

Section 13.2.9.1, “`SELECT ... INTO` Syntax”

Section 1.8.5.1, “`SELECT INTO TABLE` Differences”

Section 13.2.9, “`SELECT` Syntax”

Section 13.6.4.2, “Local Variable Scope and Resolution”

Section 16.4.1.13, “Replication and System Functions”

SELECT ... INTO DUMPFILE

Section 6.1.3, “Making MySQL Secure Against Attackers”

Section 5.1.4, “Server System Variables”

Section 2.11.1, “Unix Postinstallation Procedures”

SELECT ... INTO OUTFILE

Section 13.2.6, “`LOAD DATA INFILE` Syntax”

Section 9.1.7, “`NULL` Values”

Section 13.2.9.1, “`SELECT ... INTO` Syntax”

Section 1.8.5.1, “`SELECT INTO TABLE` Differences”

Section 7.1, “Backup and Recovery Types”

Section 6.2.7, “Causes of Access-Denied Errors”

Section 7.4.3, “Dumping Data in Delimited-Text Format with `mysqldump`”

Section 14.3.7.2, “Forcing InnoDB Recovery”

Section 6.1.3, “Making MySQL Secure Against Attackers”

Section 6.2.1, “Privileges Provided by MySQL”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

Section 1.2, “Typographical and Syntax Conventions”

Section E.10.5, “Windows Platform Limitations”

SELECT ... INTO OUTFILE

'file_name'

Section 13.2.9.1, “`SELECT ... INTO` Syntax”

SELECT ... INTO var_list

Section E.1, “Restrictions on Stored Programs”

Section 13.6.4, “Variables in Stored Programs”

SELECT ... LOCK IN SHARE MODE

Section 14.3.9.1, “InnoDB Lock Modes”

Section 14.3.9.3, “`SELECT ... FOR UPDATE` and

`SELECT ... LOCK IN SHARE MODE` Locking Reads”

Section 13.3.6, “`SET TRANSACTION` Syntax”

Section 14.3.9.6, “Locks Set by Different SQL

Statements in InnoDB”

SELECT DISTINCT

Section 8.12.5.2, “General Thread States”

Section E.4, “Restrictions on Subqueries”

SEMAPHORES

Section 14.3.11.4, “Adaptive Hash Indexes”

SET

Section 13.7.4, “`SET` Syntax”

Section 13.7.5.40, “`SHOW VARIABLES` Syntax”

Section 12.3.4, “Assignment Operators”

Section 16.1.3.4, “Binary Log Options and Variables”

Section 19.7, “Binary Logging of Stored Programs”

Section 19.1, “Defining Stored Programs”

Section 4.6.11, “`mysql_find_rows` — Extract SQL Statements from Files”

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 5.1.5.2, “Dynamic System Variables”

Section 19.4.2, “Event Scheduler Configuration”

Chapter 12, *Functions and Operators*

Section 12.14, “Information Functions”

Section 1.8.4, “MySQL Extensions to Standard SQL”

Section 8.9.3.3, “Query Cache Configuration”

Section 16.1.3.2, “Replication Master Options and Variables”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 13.2.10, “Subquery Syntax”
Section 19.3.1, “Trigger Syntax”
Section 9.4, “User-Defined Variables”
Section 4.2.3.5, “Using Options to Set Program Variables”
Section 5.1.5, “Using System Variables”
Section 4.5.1.6.2, “Using the `--safe-updates` Option”
Section 13.6.4, “Variables in Stored Programs”

SET autocommit

Section 8.5.4, “Bulk Data Loading for InnoDB Tables”
Section 13.3, “MySQL Transactional and Locking Statements”

SET GLOBAL

Section 13.7.1.3, “GRANT Syntax”
Section 13.7.4, “SET Syntax”
Section 5.1.5.2, “Dynamic System Variables”
Section 8.9.2.2, “Multiple Key Caches”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.5, “Using System Variables”

SET GLOBAL

sql_slave_skip_counter

Section 13.4.2.4, “SET GLOBAL `sql_slave_skip_counter` Syntax”

SET NAMES

Section 10.1.6, “Character Set for Error Messages”
Section 10.1.4, “Connection Character Sets and Collations”
Section 12.2, “Type Conversion in Expression Evaluation”

SET PASSWORD

Section C.5.2.4, “Client does not support authentication protocol”
Section 16.4.1.25, “SET PASSWORD and Row-Based Replication”
Section 13.7.1.6, “SET PASSWORD Syntax”
Section 13.7.4, “SET Syntax”
Section 6.3.5, “Assigning Account Passwords”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 12.14, “Information Functions”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”

Section 6.1.2.4, “Password Hashing in MySQL”
Section 6.1.2.3, “Passwords and Logging”
Section 16.4.1.35, “Replication and Variables”
Section 16.4.1.6, “Replication of `CURRENT_USER()`”
Section 2.11.2, “Securing the Initial MySQL Accounts”
Section 5.1.4, “Server System Variables”
Section 6.2.3, “Specifying Account Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.2.6, “When Privilege Changes Take Effect”

SET SESSION

Section 13.7.4, “SET Syntax”
Section 5.1.5.2, “Dynamic System Variables”
Section 5.1.5, “Using System Variables”

SET sql_mode='mode_value'

Section 1.8.2, “Selecting SQL Modes”

SET TIMESTAMP = value

Section 8.12.5, “Examining Thread Information”

SET TRANSACTION

Section 13.3.6, “SET TRANSACTION Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 14.3.9, “The InnoDB Transaction Model and Locking”

SET TRANSACTION ISOLATION

LEVEL

Section 13.3.6, “SET TRANSACTION Syntax”
Section 13.7.4, “SET Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”

SHOW

Section 13.1.11, “CREATE EVENT Syntax”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Chapter 20, *INFORMATION_SCHEMA Tables*
Section 22.8.7.28, “`mysql_stmt_store_result()`”
Section 22.8.3.69, “`mysql_store_result()`”
Section 22.8.3.71, “`mysql_use_result()`”
Section 13.7.5.6, “SHOW COLUMNS Syntax”
Section 13.7.5.23, “SHOW INDEX Syntax”
Section 13.7.5.25, “SHOW OPEN TABLES Syntax”
Section 13.7.5.38, “SHOW TABLES Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”

Section 13.7.5, “SHOW Syntax”
Section 22.8.1, “C API Data Structures”
Section 22.8.2, “C API Function Overview”
Section 14.3.5, “Creating and Using InnoDB Tables”
Section 3.3, “Creating and Using a Database”
Section 13.6.6.2, “Cursor DECLARE Syntax”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 20.31, “Extensions to SHOW Statements”
Section 9.2.3, “Mapping of Identifiers to File Names”
Section 17.6.4, “MySQL Cluster Replication Schema and Tables”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 21.1, “Performance Schema Quick Start”
Section E.1, “Restrictions on Stored Programs”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 20.1, “The INFORMATION_SCHEMA CHARACTER_SETS Table”
Section 20.3, “The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table”
Section 20.2, “The INFORMATION_SCHEMA COLLATIONS Table”
Section 20.5, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”
Section 20.4, “The INFORMATION_SCHEMA COLUMNS Table”
Section 20.6, “The INFORMATION_SCHEMA ENGINES Table”
Section 20.7, “The INFORMATION_SCHEMA EVENTS Table”
Section 20.29.1, “The INFORMATION_SCHEMA FILES Table”
Section 20.8, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”
Section 20.9, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”
Section 20.10, “The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table”
Section 20.29.2, “The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table”
Section 20.12, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 20.13, “The INFORMATION_SCHEMA PLUGINS Table”
Section 20.14, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 20.15, “The INFORMATION_SCHEMA PROFILING Table”
Section 20.16, “The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table”
Section 20.19, “The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table”

Section 20.18, “The INFORMATION_SCHEMA SCHEMATA Table”
Section 20.20, “The INFORMATION_SCHEMA STATISTICS Table”
Section 20.23, “The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table”
Section 20.24, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”
Section 20.21, “The INFORMATION_SCHEMA TABLES Table”
Section 20.22, “The INFORMATION_SCHEMA TABLESPACES Table”
Section 20.25, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 20.26, “The INFORMATION_SCHEMA USER_PRIVILEGES Table”
Section 20.27, “The INFORMATION_SCHEMA VIEWS Table”
Section 5.2.4, “The Binary Log”
Section 10.1.12, “UTF-8 for Metadata”

SHOW AUTHORS

Section 13.7.5.1, “SHOW AUTHORS Syntax”

SHOW BINARY LOGS

Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 13.7.5.2, “SHOW BINARY LOGS Syntax”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.4.1, “SQL Statements for Controlling Master Servers”

SHOW BINLOG EVENTS

Section 13.7.5.3, “SHOW BINLOG EVENTS Syntax”
Section 13.4.2.5, “START SLAVE Syntax”
Section 17.6.4, “MySQL Cluster Replication Schema and Tables”
Section E.3, “Restrictions on Server-Side Cursors”
Section 13.4.1, “SQL Statements for Controlling Master Servers”

SHOW CHARACTER SET

Section 13.1.1, “ALTER DATABASE Syntax”
Section 13.7.5.4, “SHOW CHARACTER SET Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 10.1.2, “Character Sets and Collations in MySQL”
Section 10.1.14, “Character Sets and Collations That MySQL Supports”
Section 20.31, “Extensions to SHOW Statements”

SHOW COLLATION

Section 13.1.1, “ALTER DATABASE Syntax”

Section 13.7.5.5, “`SHOW COLLATION` Syntax”
Section 10.1.9.3, “`SHOW` Statements and `INFORMATION_SCHEMA`”
Section 22.8.1, “C API Data Structures”
Section 10.5, “Character Set Configuration”
Section 10.1.2, “Character Sets and Collations in MySQL”
Section 10.1.3.5, “Character String Literal Character Set and Collation”
Section 10.4.2, “Choosing a Collation ID”
Section 10.1.3.4, “Column Character Set and Collation”
Section 10.1.3.2, “Database Character Set and Collation”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 10.1.3.3, “Table Character Set and Collation”
Section 20.3, “The `INFORMATION_SCHEMA` `COLLATION_CHARACTER_SET_APPLICABILITY` Table”
Section 20.2, “The `INFORMATION_SCHEMA` `COLLATIONS` Table”

SHOW COLUMNS

Section 13.8.1, “`DESCRIBE` Syntax”
Section 13.8.2, “`EXPLAIN` Syntax”
Section 13.7.5.6, “`SHOW COLUMNS` Syntax”
Section 10.1.9.3, “`SHOW` Statements and `INFORMATION_SCHEMA`”
Section 20.31, “Extensions to `SHOW` Statements”
Section 8.2.1.3, “Optimizing `LIMIT` Queries”
Section 8.11.3, “Optimizing Disk I/O”
Section 8.8.1, “Optimizing Queries with `EXPLAIN`”

SHOW COLUMNS FROM tbl_name LIKE 'enum_col'

Section 11.4.4, “The `ENUM` Type”

SHOW CONTRIBUTORS

Section 13.7.5.7, “`SHOW CONTRIBUTORS` Syntax”

SHOW COUNT ()

Section 13.7.5.18, “`SHOW ERRORS` Syntax”
Section 13.7.5.41, “`SHOW WARNINGS` Syntax”

SHOW CREATE DATABASE

Section 13.7.5.8, “`SHOW CREATE DATABASE` Syntax”
Section 10.1.9.3, “`SHOW` Statements and `INFORMATION_SCHEMA`”
Section 5.1.4, “Server System Variables”

SHOW CREATE EVENT

Section 13.7.5.9, “`SHOW CREATE EVENT` Syntax”
Section 13.7.5.19, “`SHOW EVENTS` Syntax”
Section 19.4.4, “Event Metadata”

Section 19.4.6, “The Event Scheduler and MySQL Privileges”

SHOW CREATE FUNCTION

Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 13.7.5.10, “`SHOW CREATE FUNCTION` Syntax”
Section 13.7.5.11, “`SHOW CREATE PROCEDURE` Syntax”
Section 1.7, “How to Report Bugs or Problems”
Section 19.2.3, “Stored Routine Metadata”

SHOW CREATE PROCEDURE

Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 13.7.5.10, “`SHOW CREATE FUNCTION` Syntax”
Section 13.7.5.11, “`SHOW CREATE PROCEDURE` Syntax”
Section 1.7, “How to Report Bugs or Problems”
Section 19.2.3, “Stored Routine Metadata”

SHOW CREATE SCHEMA

Section 13.7.5.8, “`SHOW CREATE DATABASE` Syntax”

SHOW CREATE TABLE

Section 13.1.17, “`CREATE TABLE` Syntax”
Section 13.8.1, “`DESCRIBE` Syntax”
Section 14.3.5.4, “`FOREIGN KEY` Constraints”
Section 18.2.5, “`KEY` Partitioning”
Section 13.7.5.6, “`SHOW COLUMNS` Syntax”
Section 13.7.5.12, “`SHOW CREATE TABLE` Syntax”
Section 10.1.9.3, “`SHOW` Statements and `INFORMATION_SCHEMA`”
Section 11.5, “Data Type Default Values”
Section 3.4, “Getting Information About Databases and Tables”
Section 14.11.2, “How to Create `FEDERATED` Tables”
Section 7.6.3, “How to Repair `MyISAM` Tables”
Section 18.3.1, “Management of `RANGE` and `LIST` Partitions”
Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”
Section 18.3.4, “Obtaining Information About Partitions”
Section 2.12.4, “Rebuilding or Repairing Tables or Indexes”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 13.1.17.2, “Silent Column Specification Changes”
Section 3.6.6, “Using Foreign Keys”

SHOW CREATE TRIGGER

Section 13.7.5.13, “`SHOW CREATE TRIGGER` Syntax”

SHOW CREATE VIEW

Section 13.7.1.3, “`GRANT` Syntax”

Section 13.7.5.14, “`SHOW CREATE VIEW` Syntax”
Section 6.2.1, “Privileges Provided by MySQL”
Section E.5, “Restrictions on Views”
Section 20.27, “The `INFORMATION_SCHEMA VIEWS` Table”
Section 19.5.4, “View Metadata”

SHOW DATABASES

Section 13.1.10, “`CREATE DATABASE` Syntax”
Section 13.7.1.3, “`GRANT` Syntax”
Chapter 20, *INFORMATION_SCHEMA Tables*
Section 13.7.5.15, “`SHOW DATABASES` Syntax”
Section 3.3, “Creating and Using a Database”
Section 20.31, “Extensions to `SHOW` Statements”
Section 3.4, “Getting Information About Databases and Tables”
Section 9.2.2, “Identifier Case Sensitivity”
Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”
Section 21.2.1, “Performance Schema Build Configuration”
Section 6.2.2, “Privilege System Grant Tables”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”

SHOW ENGINE

Section 13.7.5.16, “`SHOW ENGINE` Syntax”

SHOW ENGINE INNODB MUTEX

Section 13.7.5.16, “`SHOW ENGINE` Syntax”
Section 5.1.4, “Server System Variables”
Section 1.4, “What Is New in MySQL 5.5”

SHOW ENGINE INNODB STATUS

Section 14.3.5.4, “`FOREIGN KEY` Constraints”
Section 14.3.14.2.1, “InnoDB Standard Monitor and Lock Monitor Output”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 14.3.14.2, “`SHOW ENGINE INNODB STATUS` and the InnoDB Monitors”
Section 13.7.5.16, “`SHOW ENGINE` Syntax”
Section 1.4.3, “Diagnostic and Monitoring Capabilities”
Section 14.3.9.9, “How to Cope with Deadlocks”
Section C.1, “Sources of Error Information”
Section 20.28.8, “The `INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS` Table”
Section 14.3.3, “Using Per-Table Tablespace”
Section 1.4, “What Is New in MySQL 5.5”

SHOW ENGINE NDB STATUS

Section 13.7.5.16, “`SHOW ENGINE` Syntax”
Section 17.2.2.3, “Initial Startup of MySQL Cluster on Windows”
Section 17.5, “Management of MySQL Cluster”
Section 17.6.4, “MySQL Cluster Replication Schema and Tables”
Section 17.5.9, “Quick Reference: MySQL Cluster SQL Statements”

SHOW ENGINE NDBCLUSTER STATUS

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 17.5.9, “Quick Reference: MySQL Cluster SQL Statements”

SHOW ENGINE

PERFORMANCE_SCHEMA STATUS

Section 13.7.5.16, “`SHOW ENGINE` Syntax”
Section 21.5, “Performance Schema Status Monitoring”

SHOW ENGINES

Section 13.7.5.17, “`SHOW ENGINES` Syntax”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 21.2.1, “Performance Schema Build Configuration”
Section 21.1, “Performance Schema Quick Start”
Section 17.5.9, “Quick Reference: MySQL Cluster SQL Statements”
Section 2.3.7.3, “Selecting a MySQL Server Type”
Section 5.1.4, “Server System Variables”
Chapter 14, *Storage Engines*
Section 14.8, “The `ARCHIVE` Storage Engine”
Section 14.9, “The `BLACKHOLE` Storage Engine”
Section 14.3, “The `InnoDB` Storage Engine”
Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”
Section 1.4, “What Is New in MySQL 5.5”

SHOW ERRORS

Section 13.6.7.3.3, “`RESIGNAL` with a Condition Value and Optional New Signal Information”
Section 13.7.5.18, “`SHOW ERRORS` Syntax”
Section 13.7.5.41, “`SHOW WARNINGS` Syntax”
Section 5.1.4, “Server System Variables”
Section 13.6.7.4.1, “Signal Condition Information Items”
Section C.1, “Sources of Error Information”

SHOW EVENTS

Section 13.7.5.19, “`SHOW EVENTS` Syntax”

Section 19.4.4, “Event Metadata”
Section 16.4.1.10, “Replication of Invoked Features”
Section 20.7, “The `INFORMATION_SCHEMA` `EVENTS` Table”
Section 19.4.6, “The Event Scheduler and MySQL Privileges”

SHOW FULL COLUMNS

Section 13.1.17, “`CREATE TABLE` Syntax”
Section 10.1.9.3, “`SHOW` Statements and `INFORMATION_SCHEMA`”
Section 20.5, “The `INFORMATION_SCHEMA` `COLUMN_PRIVILEGES` Table”

SHOW FULL PROCESSLIST

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 8.12.5, “Examining Thread Information”

SHOW FULL TABLES

Section 13.7.5.38, “`SHOW TABLES` Syntax”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

SHOW FUNCTION CODE

Section 13.7.5.20, “`SHOW FUNCTION CODE` Syntax”
Section 13.7.5.28, “`SHOW PROCEDURE CODE` Syntax”

SHOW FUNCTION STATUS

Section 13.7.5.21, “`SHOW FUNCTION STATUS` Syntax”
Section 13.7.5.29, “`SHOW PROCEDURE STATUS` Syntax”
Section 19.2.3, “Stored Routine Metadata”

SHOW GLOBAL STATUS

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 5.1.4, “Server System Variables”
Section 20.8, “The `INFORMATION_SCHEMA` `GLOBAL_STATUS` and `SESSION_STATUS` Tables”

SHOW GLOBAL VARIABLES

Section 20.9, “The `INFORMATION_SCHEMA` `GLOBAL_VARIABLES` and `SESSION_VARIABLES` Tables”

SHOW GRANTS

Section 13.7.1.3, “`GRANT` Syntax”
Section 13.7.1.5, “`REVOKE` Syntax”
Section 13.7.5.22, “`SHOW GRANTS` Syntax”
Section 13.7.5.27, “`SHOW PRIVILEGES` Syntax”
Section 6.3.2, “Adding User Accounts”
Section 6.2.2, “Privilege System Grant Tables”
Section 6.1.1, “Security Guidelines”
Section 6.2, “The MySQL Access Privilege System”

SHOW INDEX

Section 13.7.2.1, “`ANALYZE TABLE` Syntax”
Section 13.8.1, “`DESCRIBE` Syntax”
Section 8.8.2, “`EXPLAIN` Output Format”
Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 13.7.5.6, “`SHOW COLUMNS` Syntax”
Section 13.7.5.23, “`SHOW INDEX` Syntax”
Section 13.2.9.3, “Index Hint Syntax”
Section 14.3.15, “Limits on InnoDB Tables”
Section 4.6.3.4, “Other `myisamchk` Options”
Section 20.20, “The `INFORMATION_SCHEMA` `STATISTICS` Table”
Section 20.23, “The `INFORMATION_SCHEMA` `TABLE_CONSTRAINTS` Table”

SHOW MASTER LOGS

Section 13.7.5.2, “`SHOW BINARY LOGS` Syntax”

SHOW MASTER STATUS

Section 13.7.5.24, “`SHOW MASTER STATUS` Syntax”
Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”
Section 16.4.5, “How to Report Replication Bugs or Problems”
Section 17.6.9, “MySQL Cluster Backups With MySQL Cluster Replication”
Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 16.4.4, “Troubleshooting Replication”

SHOW OPEN TABLES

Section 13.7.5.25, “`SHOW OPEN TABLES` Syntax”

SHOW PLUGINS

Section 13.7.3.3, “`INSTALL PLUGIN` Syntax”
Section 13.7.5.26, “`SHOW PLUGINS` Syntax”
Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 6.3.10.1, “Installing the Audit Log Plugin”
Section 6.3.6.2.1, “Installing the PAM Authentication Plugin”
Section 6.3.6.3.1, “Installing the Windows Authentication Plugin”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 5.1.8.2, “Obtaining Server Plugin Information”
Chapter 18, *Partitioning*
Section 6.3.6, “Pluggable Authentication”

Section 23.2.1, “Plugin API Characteristics”
Section 23.2.2, “Plugin API Components”
Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”
Section 20.29.2, “The `INFORMATION_SCHEMA` `ndb_transid_mysql_connection_map` Table”
Section 20.13, “The `INFORMATION_SCHEMA` `PLUGINS` Table”
Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”
Section 8.11.6.1, “Thread Pool Components and Installation”
Section 1.4, “What Is New in MySQL 5.5”
Section 23.2.4.6, “Writing `INFORMATION_SCHEMA` Plugins”
Section 23.2.4.8, “Writing Audit Plugins”
Section 23.2.4.5, “Writing Daemon Plugins”
Section 23.2.4.4, “Writing Full-Text Parser Plugins”
Section 23.2.4.9.1, “Writing the Server-Side Authentication Plugin”

SHOW PRIVILEGES

Section 13.7.5.27, “`SHOW PRIVILEGES` Syntax”

SHOW PROCEDURE CODE

Section 13.7.5.20, “`SHOW FUNCTION CODE` Syntax”
Section 13.7.5.28, “`SHOW PROCEDURE CODE` Syntax”

SHOW PROCEDURE STATUS

Section 13.7.5.21, “`SHOW FUNCTION STATUS` Syntax”
Section 13.7.5.29, “`SHOW PROCEDURE STATUS` Syntax”
Section 19.2.3, “Stored Routine Metadata”

SHOW PROCESSLIST

Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”
Section 13.7.1.3, “`GRANT` Syntax”
Section 14.3.13, “InnoDB Error Handling”
Section 13.7.6.4, “`KILL` Syntax”
Section 22.8.3.43, “`mysql_list_processes()`”
Section 13.7.5.30, “`SHOW PROCESSLIST` Syntax”
Section 13.7.5.31, “`SHOW PROFILE` Syntax”
Section 13.7.5.35, “`SHOW SLAVE STATUS` Syntax”
Section 13.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”
Section C.5.2.7, “Too many connections”
Section 16.1.4.1, “Checking Replication Status”
Section 5.4.1.2, “Command Probes”
Section 5.4.1.1, “Connection Probes”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 19.4.2, “Event Scheduler Configuration”

Section 8.12.5.2, “General Thread States”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 6.2.1, “Privileges Provided by MySQL”
Section 5.4.1.6, “Query Execution Probes”
Section 5.4.1.3, “Query Probes”
Section 16.2.1, “Replication Implementation Details”
Section 16.3.6, “Switching Masters During Failover”
Section 20.29.2, “The `INFORMATION_SCHEMA` `ndb_transid_mysql_connection_map` Table”
Section 17.5.10.15, “The `ndbinfo` `server_operations` Table”
Section 17.5.10.16, “The `ndbinfo` `server_transactions` Table”
Section 16.4.4, “Troubleshooting Replication”

SHOW PROFILE

Section 13.7.5.31, “`SHOW PROFILE` Syntax”
Section 13.7.5.32, “`SHOW PROFILES` Syntax”
Section 8.12.5, “Examining Thread Information”
Section 8.12.5.2, “General Thread States”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 5.1.4, “Server System Variables”
Section 20.15, “The `INFORMATION_SCHEMA` `PROFILING` Table”

SHOW PROFILES

Section 13.7.5.31, “`SHOW PROFILE` Syntax”
Section 13.7.5.32, “`SHOW PROFILES` Syntax”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 5.1.4, “Server System Variables”
Section 20.15, “The `INFORMATION_SCHEMA` `PROFILING` Table”

SHOW RELAYLOG EVENTS

Section 13.7.5.3, “`SHOW BINLOG EVENTS` Syntax”
Section 13.7.5.33, “`SHOW RELAYLOG EVENTS` Syntax”
Section 13.4.2, “SQL Statements for Controlling Slave Servers”

SHOW SCHEMAS

Section 13.7.5.15, “`SHOW DATABASES` Syntax”

SHOW SESSION STATUS

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 20.8, “The `INFORMATION_SCHEMA` `GLOBAL_STATUS` and `SESSION_STATUS` Tables”

SHOW SESSION VARIABLES

Section 20.9, “The `INFORMATION_SCHEMA` `GLOBAL_VARIABLES` and `SESSION_VARIABLES` Tables”

SHOW SLAVE HOSTS

Section 13.7.5.34, “SHOW SLAVE HOSTS Syntax”
Section 16.1.4.1, “Checking Replication Status”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 13.4.1, “SQL Statements for Controlling Master Servers”

SHOW SLAVE STATUS

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
Section 13.4.2.5, “START SLAVE Syntax”
Section 16.1.4.1, “Checking Replication Status”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 16.4.5, “How to Report Replication Bugs or Problems”
Section 17.6.3, “Known Issues in MySQL Cluster Replication”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.2.1, “Replication Implementation Details”
Section 8.12.5.6, “Replication Slave I/O Thread States”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 16.3.7, “Setting Up Replication Using SSL”
Section 16.4.1.26, “Slave Errors During Replication”
Section 16.2.2.2, “Slave Status Logs”
Section C.1, “Sources of Error Information”
Section 13.4.2, “SQL Statements for Controlling Slave Servers”
Section 16.4.4, “Troubleshooting Replication”

SHOW STATUS

Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.7.5.36, “SHOW STATUS Syntax”
Section 17.3.2.7, “Defining SQL and Other API Nodes in a MySQL Cluster”
Section 17.5, “Management of MySQL Cluster”
Section 17.6, “MySQL Cluster Replication”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 17.5.15, “NDB API Statistics Counters and Variables”
Section 23.2.1, “Plugin API Characteristics”
Section 8.9.3.4, “Query Cache Status and Maintenance”
Section 17.5.9, “Quick Reference: MySQL Cluster SQL Statements”
Section 16.4.1.21, “Replication and Temporary Tables”
Section 16.2.1, “Replication Implementation Details”
Section 16.4.1.28, “Replication Retries and Timeouts”
Section E.1, “Restrictions on Stored Programs”

Section 16.3.8.3, “Semisynchronous Replication Monitoring”
Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”
Section 23.2.4.2.2, “Server Plugin Status and System Variables”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 23.2.4.8, “Writing Audit Plugins”
Section 23.2.4.4, “Writing Full-Text Parser Plugins”
Section 23.2.4, “Writing Plugins”

SHOW STATUS LIKE 'perf%'

Section 21.5, “Performance Schema Status Monitoring”

SHOW TABLE STATUS

Section 13.1.17, “CREATE TABLE Syntax”
Section 13.8.1, “DESCRIBE Syntax”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 13.7.5.6, “SHOW COLUMNS Syntax”
Section 13.7.5.37, “SHOW TABLE STATUS Syntax”
Section 14.3.5, “Creating and Using InnoDB Tables”
Section 14.3.12.2, “File Space Management”
Section 14.3.15, “Limits on InnoDB Tables”
Section 18.3.4, “Obtaining Information About Partitions”
Section 14.3.11.5, “Physical Row Structure”
Section 14.8, “The ARCHIVE Storage Engine”
Section 14.3.5.3.1, “Traditional InnoDB Auto-Increment Locking”

SHOW TABLES

Chapter 20, *INFORMATION_SCHEMA Tables*
Section 13.7.5.37, “SHOW TABLE STATUS Syntax”
Section 13.7.5.38, “SHOW TABLES Syntax”
Section C.5.2.16, “Table 'tbl_name' doesn't exist”
Section C.5.7.2, “TEMPORARY Table Problems”
Section 3.3.2, “Creating a Table”
Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”
Section 20.31, “Extensions to SHOW Statements”
Section 9.2.2, “Identifier Case Sensitivity”
Section 9.2.3, “Mapping of Identifiers to File Names”
Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”
Section 17.5.10, “The ndbinfo MySQL Cluster Information Database”

SHOW TABLES FROM some_ndb_database

Section 17.5.11.2, “MySQL Cluster and MySQL Privileges”

SHOW TRIGGERS

- Section 13.7.5.39, “SHOW TRIGGERS Syntax”
- Section 20.25, “The INFORMATION_SCHEMA TRIGGERS Table”
- Section 19.3.2, “Trigger Metadata”

SHOW VARIABLES

- Section 13.7.4, “SET Syntax”
- Section 13.7.5.40, “SHOW VARIABLES Syntax”
- Section 23.2.4.3, “Compiling and Installing Plugin Libraries”
- Section 19.4.2, “Event Scheduler Configuration”
- Chapter 18, *Partitioning*
- Section 23.2.1, “Plugin API Characteristics”
- Section 5.3, “Running Multiple MySQL Instances on One Machine”
- Section 16.3.8.3, “Semisynchronous Replication Monitoring”
- Section 5.1.4, “Server System Variables”
- Section 17.6.6, “Starting MySQL Cluster Replication (Single Replication Channel)”
- Section 5.1.5, “Using System Variables”
- Section 23.2.4, “Writing Plugins”

SHOW WARNINGS

- Section 13.1.7, “ALTER TABLE Syntax”
- Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Syntax”
- Section 8.8.3, “EXPLAIN EXTENDED Output Format”
- Section 8.8.2, “EXPLAIN Output Format”
- Section 13.2.6, “LOAD DATA INFILE Syntax”
- Section 1.8.6.1, “PRIMARY KEY and UNIQUE Index Constraints”
- Section 13.7.5.18, “SHOW ERRORS Syntax”
- Section 13.7.5.41, “SHOW WARNINGS Syntax”
- Section 13.6.7.4, “SIGNAL Syntax”
- Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
- Section 13.6.7, “Condition Handling”
- Section 13.6.7.4.2, “Effect of Signals on Handlers, Cursors, and Statements”
- Section 9.2.4, “Function Name Parsing and Resolution”
- Section 5.2.4.3, “Mixed Binary Logging Format”
- Section 8.13.12, “Optimizing Subqueries with EXISTS Strategy”
- Section 5.1.4, “Server System Variables”
- Section 13.6.7.4.1, “Signal Condition Information Items”
- Section C.1, “Sources of Error Information”

SIGNAL

- Section 13.6.7.1, “DECLARE ... CONDITION Syntax”
- Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
- Section 13.6.7.3, “RESIGNAL Syntax”

- Section 13.6.7.4, “SIGNAL Syntax”
- Section 13.6.7, “Condition Handling”
- Section 13.6.7.4.2, “Effect of Signals on Handlers, Cursors, and Statements”
- Section 12.14, “Information Functions”
- Section E.2, “Restrictions on Condition Handling”
- Section E.1, “Restrictions on Stored Programs”
- Section 13.6.7.4.1, “Signal Condition Information Items”
- Section 1.4, “What Is New in MySQL 5.5”

START SLAVE

- Section 13.4.2.1, “CHANGE MASTER TO Syntax”
- Section 13.4.2.3, “RESET SLAVE Syntax”
- Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
- Section 13.4.2.5, “START SLAVE Syntax”
- Section 13.4.2.6, “STOP SLAVE Syntax”
- Section 17.4.18, “ndb_restore — Restore a MySQL Cluster Backup”
- Section 4.5.4, “mysqldump — A Database Backup Program”
- Section 17.6.8, “Implementing Failover with MySQL Cluster Replication”
- Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”
- Section 16.1.4.2, “Pausing Replication on the Slave”
- Section 16.3.4, “Replicating Different Databases to Different Slaves”
- Section 16.2.1, “Replication Implementation Details”
- Section 16.1.3.3, “Replication Slave Options and Variables”
- Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
- Section 16.4.1.26, “Slave Errors During Replication”
- Section 17.6.6, “Starting MySQL Cluster Replication (Single Replication Channel)”
- Section 16.3.6, “Switching Masters During Failover”
- Section 16.4.4, “Troubleshooting Replication”
- Section 17.6.7, “Using Two Replication Channels for MySQL Cluster Replication”

START TRANSACTION

- Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”
- Section 14.3.13, “InnoDB Error Handling”
- Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”
- Section 14.3.9.3, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”
- Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
- Section 4.5.4, “mysqldump — A Database Backup Program”
- Section 14.3.9.9, “How to Cope with Deadlocks”

Section 13.3.5.1, “Interaction of Table Locking and Transactions”
Section 13.3, “MySQL Transactional and Locking Statements”
Section E.1, “Restrictions on Stored Programs”
Section 16.3.8, “Semisynchronous Replication”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 14.3.9, “The `InnoDB` Transaction Model and Locking”
Section 19.3.1, “Trigger Syntax”
Section 14.3.5.1, “Using `InnoDB` Transactions”
Section 13.3.7.2, “XA Transaction States”

START TRANSACTION WITH CONSISTENT SNAPSHOT

Section 14.3.9.2, “Consistent Nonlocking Reads”

STOP SLAVE

Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”
Section 13.4.1.2, “`RESET MASTER` Syntax”
Section 13.4.2.3, “`RESET SLAVE` Syntax”
Section 13.4.2.5, “`START SLAVE` Syntax”
Section 13.4.2.6, “`STOP SLAVE` Syntax”
Section 16.1.4.1, “Checking Replication Status”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”
Section 16.1.4.2, “Pausing Replication on the Slave”
Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 16.3.6, “Switching Masters During Failover”
Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

STOP SLAVE SQL_THREAD

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

T

[index top [4407]]

TRUNCATE TABLE

Section 13.1.19, “`CREATE TRIGGER` Syntax”
Section 13.2.2, “`DELETE` Syntax”
Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”
Section 13.2.4, “`HANDLER` Syntax”
Section 14.10.2, “`MERGE` Table Problems”

Section 13.1.33, “`TRUNCATE TABLE` Syntax”
Section 17.5.13.1, “Adding MySQL Cluster Data Nodes Online: General Issues”
Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 14.5.3.3, “Compressed Table Characteristics”
Section 17.4.8, “`ndb_delete_all` — Delete All Rows from an NDB Table”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 21.2.3.2.1, “Event Pre-Filtering”
Section 21.7.4.1, “Event Wait Summary Tables”
Section 21.7.4.2, “File I/O Summary Tables”
Section 8.9.3.1, “How the Query Cache Operates”
Section 17.1.6.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”
Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”
Section 18.3.3, “Maintenance of Partitions”
Section 18.3.1, “Management of `RANGE` and `LIST` Partitions”
Section 8.5.6, “Optimizing `InnoDB` DDL Operations”
Section 21.2.3.1, “Performance Schema Event Timing”
Section 21.6, “Performance Schema General Table Characteristics”
Section 21.7.4, “Performance Schema Summary Tables”
Section 17.6.9.2, “Point-In-Time Recovery Using MySQL Cluster Replication”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.20, “Replication and `MEMORY` Tables”
Section 16.4.1.34, “Replication and `TRUNCATE TABLE`”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 21.7.3.1, “The `events_waits_current` Table”
Section 21.7.3.2, “The `events_waits_history` Table”
Section 21.7.3.3, “The `events_waits_history_long` Table”
Section 14.6, “The `MEMORY` Storage Engine”
Section 21.7.1.3, “The `setup_timers` Table”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 22.8.11.2, “What Results You Can Get from a Query”

U

[index top [4407]]

UNINSTALL PLUGIN

Section 13.7.6.3, “`FLUSH` Syntax”

Section 13.7.3.3, “`INSTALL PLUGIN` Syntax”
Section 13.7.5.26, “`SHOW PLUGINS` Syntax”
Section 13.7.3.4, “`UNINSTALL PLUGIN` Syntax”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 21.10, “Performance Schema and Plugins”
Section 14.2.1, “Pluggable Storage Engine Architecture”
Section 23.2.2, “Plugin API Components”
Section 23.2.4.2.1, “Server Plugin Library and Plugin Descriptors”
Section 20.13, “The `INFORMATION_SCHEMA PLUGINS` Table”
Section 23.2.4.6, “Writing `INFORMATION_SCHEMA` Plugins”
Section 23.2.4.8, “Writing Audit Plugins”
Section 23.2.4.5, “Writing Daemon Plugins”
Section 23.2.4.4, “Writing Full-Text Parser Plugins”
Section 23.2.4.9.1, “Writing the Server-Side Authentication Plugin”

UNION

Section 13.1.17, “`CREATE TABLE` Syntax”
Section 13.1.20, “`CREATE VIEW` Syntax”
Section 8.8.2, “`EXPLAIN` Output Format”
Section 13.2.9, “`SELECT` Syntax”
Section 13.2.9.4, “`UNION` Syntax”
Section 22.8.1, “C API Data Structures”
Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”
Section 12.14, “Information Functions”
Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”
Section 11.2.5, “Numeric Type Attributes”
Section E.5, “Restrictions on Views”
Section 10.1.9.1, “Result Strings”
Section 3.6.7, “Searching on Two Keys”
Section 5.1.6, “Server Status Variables”
Section 13.2.10, “Subquery Syntax”
Section 14.10, “The `MERGE` Storage Engine”
Section 8.13.1.1, “The Range Access Method for Single-Part Indexes”
Section 19.5.3, “Updatable and Insertable Views”
Section 19.5.2, “View Processing Algorithms”
Section 19.5.1, “View Syntax”
Section 12.11, “XML Functions”

UNION ALL

Section 13.2.9.4, “`UNION` Syntax”
Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”
Section 12.14, “Information Functions”

Section 19.5.3, “Updatable and Insertable Views”
Section 19.5.2, “View Processing Algorithms”

UNION DISTINCT

Section 13.2.9.4, “`UNION` Syntax”

UNLOCK TABLES

Section 13.7.6.3, “`FLUSH` Syntax”
Section 13.3.5, “`LOCK TABLES` and `UNLOCK TABLES` Syntax”
Section 13.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”
Section 8.6.2, “Bulk Data Loading for `MyISAM` Tables”
Section 7.2, “Database Backup Methods”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 14.3.9.9, “How to Cope with Deadlocks”
Section 13.3.5.1, “Interaction of Table Locking and Transactions”
Section 14.3.15, “Limits on `InnoDB` Tables”
Section E.1, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 8.11.1, “System Factors and Startup Parameter Tuning”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 1.8.5.3, “Transaction and Atomic Operation Differences”

UPDATE

Section 4.5.1.1, “`mysql` Options”
Section 4.6.7.2, “`mysqlbinlog` Row Event Display”
Section 13.7.2.2, “`CHECK TABLE` Syntax”
Section 13.1.17, “`CREATE TABLE` Syntax”
Section 13.1.19, “`CREATE TRIGGER` Syntax”
Section 13.1.20, “`CREATE VIEW` Syntax”
Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”
Section 14.3.5.4, “`FOREIGN KEY` Constraints”
Section 13.7.1.3, “`GRANT` Syntax”
Chapter 20, *`INFORMATION_SCHEMA` Tables*
Section 14.3.9.4, “`InnoDB` Record, Gap, and Next-Key Locks”
Section 14.3.4, “`InnoDB` Startup Options and System Variables”
Section 13.2.5.3, “`INSERT ... ON DUPLICATE KEY UPDATE` Syntax”
Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 13.2.5, “`INSERT` Syntax”
Section 13.2.9.2, “`JOIN` Syntax”
Section 13.7.6.4, “`KILL` Syntax”
Section 13.2.6, “`LOAD DATA INFILE` Syntax”

Section 22.8.3.1, “`mysql_affected_rows()`”
Section 22.8.3.35, “`mysql_info()`”
Section 22.8.3.37, “`mysql_insert_id()`”
Section 22.8.3.48, “`mysql_num_rows()`”
Section 22.8.3.49, “`mysql_options()`”
Section 22.8.7.10, “`mysql_stmt_execute()`”
Section 22.8.7.16, “`mysql_stmt_insert_id()`”
Section 22.8.7.18, “`mysql_stmt_num_rows()`”
Section 1.8.6.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 14.3.9.3, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”
Section 13.7.1.6, “SET PASSWORD Syntax”
Section 13.3.6, “SET TRANSACTION Syntax”
Section 13.7.5.41, “SHOW WARNINGS Syntax”
Section 1.8.5.2, “UPDATE Differences”
Section 13.2.11, “UPDATE Syntax”
Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 6.3.2, “Adding User Accounts”
Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 6.3.5, “Assigning Account Passwords”
Section 12.3.4, “Assignment Operators”
Section 13.5.4, “Automatic Prepared Statement Repreparation”
Section 16.1.3.4, “Binary Log Options and Variables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 22.8.2, “C API Function Overview”
Section 22.8.6, “C API Prepared Statement Function Overview”
Section 22.8.13, “C API Support for Multiple Statement Execution”
Section 6.2.7, “Causes of Access-Denied Errors”
Section 10.1.13, “Column Character Set Conversion”
Section 14.3.5.3.2, “Configurable InnoDB Auto-Increment Locking”
Section 14.3.9.2, “Consistent Nonlocking Reads”
Section 1.8.6.2, “Constraints on Invalid Data”
Section 14.11.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 11.5, “Data Type Default Values”
Section 11.1.2, “Date and Time Type Overview”
Section 14.3.7.2, “Forcing InnoDB Recovery”
Chapter 12, *Functions and Operators*
Section 8.12.5.2, “General Thread States”
Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”
Section 8.9.3.1, “How the Query Cache Operates”
Section 12.14, “Information Functions”
Section 8.10.1, “Internal Locking Methods”
Section C.5.8, “Known Issues in MySQL”
Section 17.6.3, “Known Issues in MySQL Cluster Replication”
Section 14.3.9.6, “Locks Set by Different SQL Statements in InnoDB”
Section 5.2.4.4, “Logging Format for Changes to mysql Database Tables”
Section 12.15, “Miscellaneous Functions”
Section 17.5.12.1, “MySQL Cluster Disk Data Objects”
Section 1.8.4, “MySQL Extensions to Standard SQL”
Section 8.2.2, “Optimizing DML Statements”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 18.5.4, “Partitioning and Table-Level Locking”
Section 6.2.2, “Privilege System Grant Tables”
Section 6.2.1, “Privileges Provided by MySQL”
Section C.5.5.2, “Problems Using DATE Columns”
Section 16.4.1.14, “Replication and LIMIT”
Section 16.4.1.23, “Replication and the Query Optimizer”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section C.5.4.1.2, “Resetting the Root Password: Unix Systems”
Section C.5.4.1.1, “Resetting the Root Password: Windows Systems”
Section 18.5, “Restrictions and Limitations on Partitioning”
Section E.5, “Restrictions on Views”
Section 13.2.10.11, “Rewriting Subqueries as Joins”
Section 2.11.2, “Securing the Initial MySQL Accounts”
Section 3.3.4.1, “Selecting All Data”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.3, “Server Command Options”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 16.4.1.26, “Slave Errors During Replication”
Section 8.2.2.2, “Speed of UPDATE Statements”
Section 5.4.1.12, “Statement Probes”
Section 13.2.10.9, “Subquery Errors”
Section 13.2.10, “Subquery Syntax”
Section 8.10.2, “Table Locking Issues”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 10.1.7.6, “The `_bin` and `binary` Collations”
Section 14.8, “The ARCHIVE Storage Engine”
Section 14.9, “The BLACKHOLE Storage Engine”
Section 20.27, “The INFORMATION_SCHEMA VIEWS Table”
Section 14.10, “The MERGE Storage Engine”
Section 14.5, “The MyISAM Storage Engine”
Section 5.2.4, “The Binary Log”
Section 6.2, “The MySQL Access Privilege System”
Section 5.1.12, “The Shutdown Process”
Section 19.3.1, “Trigger Syntax”
Section 19.5.3, “Updatable and Insertable Views”
Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

Section 3.6.9, “Using `AUTO_INCREMENT`”
Section 4.5.1.6.2, “Using the `--safe-updates` Option”
Section 19.3, “Using Triggers”
Section 22.8.11.2, “What Results You Can Get from a Query”
Section 6.2.6, “When Privilege Changes Take Effect”
Section 22.8.11.1, “Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success”

UPDATE ... ()

Section 14.3.9.2, “Consistent Nonlocking Reads”

UPDATE ... WHERE ...

Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”

UPDATE IGNORE

Section 13.2.11, “`UPDATE` Syntax”
Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”

UPDATE t1,t2 ...

Section 5.4.1.12, “Statement Probes”

USE

Section 4.5.1.1, “`mysql` Options”
Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Chapter 20, *INFORMATION_SCHEMA* Tables
Section 13.8.4, “`USE` Syntax”
Section 16.1.3.4, “Binary Log Options and Variables”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 3.3.1, “Creating and Selecting a Database”
Section 3.3, “Creating and Using a Database”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.1, “Dumping Data in SQL Format with `mysqldump`”
Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 13.2.9.3, “Index Hint Syntax”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 16.2.3.3, “Replication Rule Application”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 19.2.1, “Stored Routine Syntax”
Section 17.5.10, “The `ndbinfo` MySQL Cluster Information Database”

USE db2

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

USE db_name

Section 4.5.1.1, “`mysql` Options”

USE test

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

W

[[index top \[4407\]](#)]

WHILE

Section 13.6.5.3, “`ITERATE` Syntax”
Section 13.6.5.4, “`LEAVE` Syntax”
Section 13.6.5.8, “`WHILE` Syntax”
Section 13.6.5, “Flow Control Statements”
Section 13.6.2, “Statement Label Syntax”

X

[[index top \[4407\]](#)]

XA COMMIT

Section 5.1.4, “Server System Variables”
Section 13.3.7.2, “XA Transaction States”

XA END

Section E.6, “Restrictions on XA Transactions”
Section 13.3.7.1, “XA Transaction SQL Syntax”
Section 13.3.7.2, “XA Transaction States”

XA PREPARE

Section 13.3.7.2, “XA Transaction States”

XA RECOVER

Section 13.3.7.1, “XA Transaction SQL Syntax”
Section 13.3.7.2, “XA Transaction States”

XA ROLLBACK

Section 5.1.4, “Server System Variables”
Section 13.3.7.2, “XA Transaction States”

XA START

Section E.6, “Restrictions on XA Transactions”
Section 13.3.7.1, “XA Transaction SQL Syntax”
Section 13.3.7.2, “XA Transaction States”

System Variable Index

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#)

A

[[index top \[4449\]](#)]

[audit_log_buffer_size](#)

Section 6.3.10.3, “Audit Log Plugin Logging Control”
Section 6.3.10.4, “Audit Log Plugin Options and Variables”

[audit_log_file](#)

Section 6.3.10.3, “Audit Log Plugin Logging Control”
Section 6.3.10.4, “Audit Log Plugin Options and Variables”
Section 6.3.10.1, “Installing the Audit Log Plugin”
Section 6.3.10, “MySQL Enterprise Audit”

[audit_log_flush](#)

Section 6.3.10.3, “Audit Log Plugin Logging Control”
Section 6.3.10.4, “Audit Log Plugin Options and Variables”

[audit_log_policy](#)

Section 6.3.10.3, “Audit Log Plugin Logging Control”
Section 6.3.10.4, “Audit Log Plugin Options and Variables”
Section 6.3.10, “MySQL Enterprise Audit”

[audit_log_rotate_on_size](#)

Section 6.3.10.3, “Audit Log Plugin Logging Control”
Section 6.3.10.4, “Audit Log Plugin Options and Variables”

[audit_log_strategy](#)

Section 6.3.10.3, “Audit Log Plugin Logging Control”
Section 6.3.10.4, “Audit Log Plugin Options and Variables”

[authentication_windows_log_level](#)

Section 5.1.4, “Server System Variables”
Section 6.3.6.3.2, “Using the Windows Authentication Plugin”

[authentication_windows_use_principal_name](#)

Section 5.1.4, “Server System Variables”
Section 6.3.6.3.2, “Using the Windows Authentication Plugin”

[auto_increment_increment](#)

Section 5.2.4.3, “Mixed Binary Logging Format”
Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”
Section 16.4.1.35, “Replication and Variables”
Section 16.1.3.2, “Replication Master Options and Variables”
Section 14.3.5.3.1, “Traditional InnoDB Auto-Increment Locking”
Section 3.6.9, “Using AUTO_INCREMENT”

[auto_increment_offset](#)

Section 5.2.4.3, “Mixed Binary Logging Format”
Section 17.1.6.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1, MySQL Cluster NDB 6.x, and MySQL Cluster NDB 7.x”
Section 16.4.1.35, “Replication and Variables”
Section 16.1.3.2, “Replication Master Options and Variables”
Section 14.3.5.3.1, “Traditional InnoDB Auto-Increment Locking”
Section 3.6.9, “Using AUTO_INCREMENT”

[AUTOCOMMIT](#)

Section 16.4.1.31, “Replication and Transactions”

[autocommit](#)

Section 13.2.2, “DELETE Syntax”
Section 14.3.4, “InnoDB Startup Options and System Variables”
Section 14.3.9.3, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 14.3.9.8, “Deadlock Detection and Rollback”
Section 13.3.5.1, “Interaction of Table Locking and Transactions”
Section 14.3.15, “Limits on InnoDB Tables”
Section 14.3.9.6, “Locks Set by Different SQL Statements in InnoDB”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 16.4.1.31, “Replication and Transactions”
Section 5.1.4, “Server System Variables”
Section 8.11.6.2, “Thread Pool Operation”
Section 1.8.5.3, “Transaction and Atomic Operation Differences”

[alter_procedure](#)

Section 13.1.5, “ALTER PROCEDURE Syntax”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”

Section 5.1.4, “Server System Variables”
Section 19.2.2, “Stored Routines and MySQL Privileges”

B

[index top [4449]]

back_log

Section 5.1.4, “Server System Variables”

basedir

Section 13.7.3.3, “INSTALL PLUGIN Syntax”
Section 5.1.4, “Server System Variables”

big_tables

Section 5.1.4, “Server System Variables”

binlog_cache_size

Section 16.1.3.4, “Binary Log Options and Variables”
Section 5.1.6, “Server Status Variables”
Section 5.2.4, “The Binary Log”

binlog_direct_non_transactional_updates

Section 16.1.3.4, “Binary Log Options and Variables”
Section 16.4.1.31, “Replication and Transactions”

binlog_format

Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 16.1.3.4, “Binary Log Options and Variables”
Section 12.7, “Date and Time Functions”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 17.6.2, “General Requirements for MySQL Cluster Replication”
Section 12.14, “Information Functions”
Section 5.2.4.4, “Logging Format for Changes to `mysql` Database Tables”
Section 12.6.2, “Mathematical Functions”
Section 12.15, “Miscellaneous Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.20, “Replication and `MEMORY` Tables”
Section 16.4.1.31, “Replication and Transactions”
Section 16.1.2, “Replication Formats”
Section 16.4.1.22, “Replication of the `mysql` System Database”
Section 5.1.3, “Server Command Options”
Section 5.2.4.2, “Setting The Binary Log Format”
Section 14.9, “The `BLACKHOLE` Storage Engine”

binlog_stmt_cache_size

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.1.6, “Server Status Variables”

bulk_insert_buffer_size

Section 14.5.1, “MyISAM Startup Options”
Section 5.1.4, “Server System Variables”
Section 8.2.2.1, “Speed of `INSERT` Statements”

C

[index top [4449]]

character_set_client

Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 13.7.4, “SET Syntax”
Section 13.7.5.9, “SHOW CREATE EVENT Syntax”
Section 13.7.5.11, “SHOW CREATE PROCEDURE Syntax”
Section 13.7.5.14, “SHOW CREATE VIEW Syntax”
Section 13.7.5.19, “SHOW EVENTS Syntax”
Section 13.7.5.29, “SHOW PROCEDURE STATUS Syntax”
Section 13.7.5.39, “SHOW TRIGGERS Syntax”
Section 22.8.5.1, “C API Prepared Statement Type Codes”
Section 10.5, “Character Set Configuration”
Section 10.1.4, “Connection Character Sets and Collations”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Section 20.7, “The `INFORMATION_SCHEMA` EVENTS Table”
Section 20.17, “The `INFORMATION_SCHEMA` ROUTINES Table”
Section 20.25, “The `INFORMATION_SCHEMA` TRIGGERS Table”
Section 20.27, “The `INFORMATION_SCHEMA` VIEWS Table”
Section 5.2.4, “The Binary Log”

character_set_connection

Section 10.1.9.2, “`CONVERT()` and `CAST()`”
Section 13.7.4, “SET Syntax”
Section 10.1.3.5, “Character String Literal Character Set and Collation”
Section 10.1.7.5, “Collation of Expressions”
Section 10.1.4, “Connection Character Sets and Collations”
Section 12.7, “Date and Time Functions”
Section 12.13, “Encryption and Compression Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 10.7, “MySQL Server Locale Support”
Section 16.4.1.35, “Replication and Variables”
Section 10.1.9.1, “Result Strings”
Section 5.1.4, “Server System Variables”

Section 9.1.1, “String Literals”
Section 10.1.8, “String Repertoire”
Section 12.2, “Type Conversion in Expression Evaluation”

character_set_database

Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 13.7.4, “`SET` Syntax”
Section 10.1.4, “Connection Character Sets and Collations”
Section 10.1.3.2, “Database Character Set and Collation”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”

character_set_filesystem

Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 13.2.9.1, “`SELECT . . . INTO` Syntax”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 12.5, “String Functions”

character_set_results

Section 13.7.4, “`SET` Syntax”
Section 10.1.6, “Character Set for Error Messages”
Section 10.1.4, “Connection Character Sets and Collations”
Section 5.1.4, “Server System Variables”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”
Section 10.1.12, “UTF-8 for Metadata”

character_set_server

Section 10.5, “Character Set Configuration”
Section 10.1.4, “Connection Character Sets and Collations”
Section 10.1.3.2, “Database Character Set and Collation”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.2, “Replication and Character Sets”
Section 16.4.1.35, “Replication and Variables”
Section 10.1.3.1, “Server Character Set and Collation”
Section 5.1.4, “Server System Variables”

character_set_system

Section 10.5, “Character Set Configuration”
Section 5.1.4, “Server System Variables”
Section 10.1.12, “UTF-8 for Metadata”

character_sets_dir

Section 10.4.3, “Adding a Simple Collation to an 8-Bit Character Set”

Section 10.4.4.1, “Defining a UCA Collation using LDML Syntax”
Section 5.1.4, “Server System Variables”

collation_connection

Section 10.1.9.2, “`CONVERT()` and `CAST()`”
Section 13.7.4, “`SET` Syntax”
Section 13.7.5.9, “`SHOW CREATE EVENT` Syntax”
Section 13.7.5.11, “`SHOW CREATE PROCEDURE` Syntax”
Section 13.7.5.14, “`SHOW CREATE VIEW` Syntax”
Section 13.7.5.19, “`SHOW EVENTS` Syntax”
Section 13.7.5.29, “`SHOW PROCEDURE STATUS` Syntax”
Section 13.7.5.39, “`SHOW TRIGGERS` Syntax”
Section 10.1.3.5, “Character String Literal Character Set and Collation”
Section 10.1.7.5, “Collation of Expressions”
Section 10.1.4, “Connection Character Sets and Collations”
Section 12.7, “Date and Time Functions”
Section 12.13, “Encryption and Compression Functions”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 10.1.9.1, “Result Strings”
Section 5.1.4, “Server System Variables”
Section 20.7, “The `INFORMATION_SCHEMA EVENTS` Table”
Section 20.17, “The `INFORMATION_SCHEMA ROUTINES` Table”
Section 20.25, “The `INFORMATION_SCHEMA TRIGGERS` Table”
Section 20.27, “The `INFORMATION_SCHEMA VIEWS` Table”
Section 5.2.4, “The Binary Log”
Section 12.2, “Type Conversion in Expression Evaluation”

collation_database

Section 13.1.15, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”
Section 10.1.4, “Connection Character Sets and Collations”
Section 10.1.3.2, “Database Character Set and Collation”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Section 5.2.4, “The Binary Log”

collation_server

Section 10.1.4, “Connection Character Sets and Collations”
Section 10.1.3.2, “Database Character Set and Collation”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.2, “Replication and Character Sets”

Section 16.4.1.35, “Replication and Variables”
Section 10.1.3.1, “Server Character Set and Collation”
Section 5.1.4, “Server System Variables”
Section 5.2.4, “The Binary Log”

completion_type

Section 22.8.3.6, “`mysql_commit()`”
Section 22.8.3.57, “`mysql_rollback()`”
Section 13.3.1, “`START TRANSACTION, COMMIT, and ROLLBACK` Syntax”
Section 5.1.4, “Server System Variables”

concurrent_insert

Section 8.10.3, “Concurrent Inserts”
Section 8.10.1, “Internal Locking Methods”
Section 8.6.1, “Optimizing `MyISAM` Queries”
Section 5.1.4, “Server System Variables”

connect_timeout

Section C.5.2.3, “Lost connection to MySQL server”
Section 22.8.3.52, “`mysql_real_connect()`”
Section C.5.2.11, “Communication Errors and Aborted Connections”
Section 5.1.4, “Server System Variables”

D

[index top [4449]]

datadir

Section 2.3, “Installing MySQL on Microsoft Windows”
Section 5.1.4, “Server System Variables”

date_format

Section 5.1.4, “Server System Variables”

datetime_format

Section 5.1.4, “Server System Variables”

debug

Section 5.1.4, “Server System Variables”

debug_sync

Section 5.1.4, “Server System Variables”

default_storage_engine

Section 13.1.14, “`CREATE LOGFILE GROUP` Syntax”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Section 14.1, “Setting the Storage Engine”

Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”

default_week_format

Section 12.7, “Date and Time Functions”
Section 18.5.3, “Partitioning Limitations Relating to Functions”
Section 5.1.4, “Server System Variables”

delay_key_write

Section 13.1.17, “`CREATE TABLE` Syntax”
Section 5.1.4, “Server System Variables”

delayed_insert_limit

Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 5.1.4, “Server System Variables”

delayed_insert_timeout

Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 5.1.4, “Server System Variables”

delayed_queue_size

Section 13.2.5.2, “`INSERT DELAYED` Syntax”
Section 5.1.4, “Server System Variables”

div_precision_increment

Section 12.6.1, “Arithmetic Operators”
Section 5.1.4, “Server System Variables”

E

[index top [4449]]

engine_condition_pushdown

Section 8.13.3, “Engine Condition Pushdown Optimization”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

error_count

Section 13.7.5.18, “`SHOW ERRORS` Syntax”
Section 5.1.4, “Server System Variables”
Section C.1, “Sources of Error Information”

event_scheduler

Section 19.4.2, “Event Scheduler Configuration”
Section 22.7.2, “Restrictions When Using the Embedded MySQL Server”
Section 5.1.4, “Server System Variables”
Section 19.4.6, “The Event Scheduler and MySQL Privileges”

expire_logs_days

Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 5.2.6, “Server Log Maintenance”
Section 5.1.4, “Server System Variables”

external_user

Section 23.2.4.9.4, “Implementing Proxy User Support in Authentication Plugins”
Section 6.3.7, “Proxy Users”
Section 5.1.4, “Server System Variables”
Section 23.2.4.9.1, “Writing the Server-Side Authentication Plugin”

F

[index top [4449]]

flush

Section 5.1.4, “Server System Variables”

flush_time

Section 5.1.4, “Server System Variables”

foreign_key_checks

Section 14.3.5.4, “FOREIGN KEY Constraints”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 5.2.4, “The Binary Log”

ft_boolean_syntax

Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 5.1.4, “Server System Variables”

ft_max_word_len

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 23.2.3.2, “Full-Text Parser Plugins”
Section 5.1.4, “Server System Variables”

ft_min_word_len

Section 12.9.2, “Boolean Full-Text Searches”
Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 23.2.3.2, “Full-Text Parser Plugins”
Section 5.1.4, “Server System Variables”

ft_query_expansion_limit

Section 5.1.4, “Server System Variables”

ft_stopword_file

Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 5.1.4, “Server System Variables”

G

[index top [4449]]

general_log

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 5.2.3, “The General Query Log”

general_log_file

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.4, “Server System Variables”
Section 5.2.3, “The General Query Log”

group_concat_max_len

Section 12.16.1, “GROUP BY (Aggregate) Functions”
Section 5.1.4, “Server System Variables”

H

[index top [4449]]

have_compress

Section 5.1.4, “Server System Variables”

have_crypt

Section 5.1.4, “Server System Variables”

have_csv

Section 5.1.4, “Server System Variables”

have_dynamic_loading

Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.4, “Server System Variables”

have_geometry

Section 5.1.4, “Server System Variables”

have_innodb

Section C.5.5.5, “Rollback Failure for Nontransactional Tables”

Section 5.1.4, “Server System Variables”

have_ndbcluster

Section 17.3.4.3, “MySQL Cluster System Variables”

have_openssl

Section 5.1.4, “Server System Variables”

have_partitioning

Chapter 18, *Partitioning*

Section 5.1.4, “Server System Variables”

have_profiling

Section 5.1.4, “Server System Variables”

have_query_cache

Section 8.9.3.3, “Query Cache Configuration”

Section 5.1.4, “Server System Variables”

have_rtree_keys

Section 5.1.4, “Server System Variables”

have_ssl

Section 6.3.8.2, “Configuring MySQL for SSL”

Section 5.1.4, “Server System Variables”

have_symlink

Section 5.1.4, “Server System Variables”

Section 8.11.3.1.2, “Using Symbolic Links for MyISAM Tables on Unix”

Section 8.11.3.1.3, “Using Symbolic Links for Databases on Windows”

hostname

Section 5.1.4, “Server System Variables”

I

[[index top](#) [4449]]

identity

Section 5.2.4.3, “Mixed Binary Logging Format”

Section 16.4.1.35, “Replication and Variables”

Section 5.1.4, “Server System Variables”

init_connect

Section 10.1.5, “Configuring the Character Set and Collation for Applications”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.1.4, “Server System Variables”

init_file

Section 5.1.4, “Server System Variables”

init_slave

Section 16.1.3.3, “Replication Slave Options and Variables”

innodb_adaptive_hash_index

Section 14.3.11.4, “Adaptive Hash Indexes”

innodb_autoextend_increment

Section 14.3.4, “InnoDB Startup Options and System Variables”

Section 14.3.6, “Adding, Removing, or Resizing InnoDB Data and Log Files”

Section 14.3.2, “Configuring InnoDB”

innodb_autoinc_lock_mode

Section 8.5.4, “Bulk Data Loading for InnoDB Tables”

Section 14.3.5.3.2, “Configurable InnoDB Auto-Increment Locking”

innodb_buffer_pool_instances

Section 14.3.4, “InnoDB Startup Options and System Variables”

Section 8.9.1, “The InnoDB Buffer Pool”

innodb_buffer_pool_size

Section 14.3.13.1, “InnoDB Error Codes”

Section 14.3.4, “InnoDB Startup Options and System Variables”

Section 8.5.7, “Optimizing InnoDB Disk I/O”

Section 8.9.1, “The InnoDB Buffer Pool”

innodb_checksums

Section 14.3.4, “InnoDB Startup Options and System Variables”

innodb_commit_concurrency

Section 14.3.4, “InnoDB Startup Options and System Variables”

innodb_concurrency_tickets

Section 14.3.4, “InnoDB Startup Options and System Variables”

Section 8.5.8, “Optimizing InnoDB Configuration Variables”

innodb_data_file_path

Section 14.3.4, “InnoDB Startup Options and System Variables”

Section 14.3.6, “Adding, Removing, or Resizing `InnoDB` Data and Log Files”

Section 14.3.2, “Configuring `InnoDB`”

Section 14.3.3.3, “Troubleshooting `InnoDB` I/O Problems”

Section 14.3.3.1, “Using Raw Devices for the Shared Tablespace”

`innodb_data_home_dir`

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

Section 14.3.2, “Configuring `InnoDB`”

Section 14.3.3.3, “Troubleshooting `InnoDB` I/O Problems”

`innodb_fast_shutdown`

Section 14.3.6, “Adding, Removing, or Resizing `InnoDB` Data and Log Files”

Section 5.1.12, “The Shutdown Process”

`innodb_file_format`

Section 13.1.17, “`CREATE TABLE` Syntax”

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

`innodb_file_format_check`

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

`innodb_file_format_max`

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

`innodb_file_per_table`

Section 13.1.17, “`CREATE TABLE` Syntax”

Section 14.3.5.5, “`InnoDB` and MySQL Replication”

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

Section 14.3.14.2.2, “`InnoDB` Tablespace Monitor Output”

Section 13.7.2.4, “`OPTIMIZE TABLE` Syntax”

Section 14.3.3.2, “Creating the `InnoDB` Tablespace”

Section 16.3.4, “Replicating Different Databases to Different Slaves”

Section 18.5, “Restrictions and Limitations on Partitioning”

Section 14.3.14.4, “Troubleshooting `InnoDB` Data Dictionary Operations”

Section 14.3.3, “Using Per-Table Tablespaces”

`innodb_flush_log_at_trx_commit`

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

Section 8.5.2, “Optimizing `InnoDB` Transaction Management”

`innodb_flush_method`

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

Section 8.5.7, “Optimizing `InnoDB` Disk I/O”

`innodb_force_load_corrupted`

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

`innodb_force_recovery`

Section 14.3.7.2, “Forcing `InnoDB` Recovery”

Section 1.7, “How to Report Bugs or Problems”

`innodb_io_capacity`

Section 1.4.2, “`InnoDB` I/O Subsystem Changes”

`innodb_large_prefix`

Section 2.12.3, “Checking Whether Tables or Indexes Must Be Rebuilt”

Section 14.3.15, “Limits on `InnoDB` Tables”

`innodb_lock_wait_timeout`

Section 14.3.13.1, “`InnoDB` Error Codes”

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

Section 14.3.9.8, “Deadlock Detection and Rollback”

Section 16.4.1.28, “Replication Retries and Timeouts”

Section 16.1.3.3, “Replication Slave Options and Variables”

`innodb_locks_unsafe_for_binlog`

Section 14.3.9.4, “`InnoDB` Record, Gap, and Next-Key Locks”

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

Section 13.3.6, “`SET TRANSACTION` Syntax”

Section 14.3.9.2, “Consistent Nonlocking Reads”

Section 14.3.9.6, “Locks Set by Different SQL Statements in `InnoDB`”

`innodb_max_purge_lag`

Section 14.3.10, “`InnoDB` Multi-Versioning”

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

`innodb_old_blocks_pct`

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

Section 8.9.1, “The `InnoDB` Buffer Pool”

innodb_old_blocks_time

Section 8.9.1, “The `InnoDB` Buffer Pool”

innodb_read_ahead_threshold

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

innodb_read_io_threads

Section 1.4.2, “`InnoDB` I/O Subsystem Changes”
Section 14.3.14.2.1, “`InnoDB` Standard Monitor and Lock Monitor Output”
Section 23.1.1, “MySQL Threads”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

innodb_stats_method

Section 8.3.7, “`InnoDB` and `MyISAM` Index Statistics Collection”

innodb_stats_sample_pages

Section 14.3.15, “Limits on `InnoDB` Tables”

innodb_strict_mode

Section 13.1.13, “`CREATE INDEX` Syntax”
Section 13.1.17, “`CREATE TABLE` Syntax”
Section 5.1.7, “Server SQL Modes”

innodb_support_xa

Section 8.5.2, “Optimizing `InnoDB` Transaction Management”

innodb_table_locks

Section 14.3.4, “`InnoDB` Startup Options and System Variables”
Section 14.3.15, “Limits on `InnoDB` Tables”

innodb_thread_concurrency

Section 14.3.14.2.1, “`InnoDB` Standard Monitor and Lock Monitor Output”
Section 14.3.4, “`InnoDB` Startup Options and System Variables”
Section 8.5.8, “Optimizing `InnoDB` Configuration Variables”

innodb_use_native_aio

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

innodb_write_io_threads

Section 1.4.2, “`InnoDB` I/O Subsystem Changes”
Section 14.3.14.2.1, “`InnoDB` Standard Monitor and Lock Monitor Output”

Section 14.3.4, “`InnoDB` Startup Options and System Variables”

Section 23.1.1, “MySQL Threads”

Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”

insert_id

Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”
Section 5.1.4, “Server System Variables”

interactive_timeout

Section 22.8.3.52, “`mysql_real_connect()`”
Section C.5.2.11, “Communication Errors and Aborted Connections”
Section 5.1.4, “Server System Variables”

J

[index top [4449]]

join_buffer_size

Section 8.13.6, “Nested-Loop Join Algorithms”
Section 5.1.4, “Server System Variables”

K

[index top [4449]]

keep_files_on_create

Section 5.1.4, “Server System Variables”

key_buffer_size

Section 8.6.2, “Bulk Data Loading for `MyISAM` Tables”
Section 8.8.4, “Estimating Query Performance”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 7.6.3, “How to Repair `MyISAM` Tables”
Section C.5.8, “Known Issues in MySQL”
Section 8.9.2.2, “Multiple Key Caches”
Section 8.9.2.6, “Restructuring a Key Cache”
Section 5.1.3, “Server Command Options”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 8.2.2.3, “Speed of `DELETE` Statements”
Section 8.6.3, “Speed of `REPAIR TABLE` Statements”
Section 5.1.5.1, “Structured System Variables”
Section 8.9.2, “The `MyISAM` Key Cache”
Section 8.11.2, “Tuning Server Parameters”
Section 4.2.3.3, “Using Option Files”

key_cache_age_threshold

Section 8.9.2.3, “Midpoint Insertion Strategy”
Section 5.1.4, “Server System Variables”
Section 5.1.5.1, “Structured System Variables”

key_cache_block_size

Section 8.9.2.5, “Key Cache Block Size”
Section 8.9.2.6, “Restructuring a Key Cache”
Section 5.1.4, “Server System Variables”
Section 5.1.5.1, “Structured System Variables”

key_cache_division_limit

Section 8.9.2.3, “Midpoint Insertion Strategy”
Section 5.1.4, “Server System Variables”
Section 5.1.5.1, “Structured System Variables”

L

[[index top \[4449\]](#)]

language

Section 5.1.4, “Server System Variables”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”

large_files_support

Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.1.4, “Server System Variables”

large_page_size

Section 5.1.4, “Server System Variables”

large_pages

Section 5.1.4, “Server System Variables”

last_insert_id

Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”

lc_messages

Section 5.1.4, “Server System Variables”
Section 10.2, “Setting the Error Message Language”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”
Section 1.4, “What Is New in MySQL 5.5”

lc_messages_dir

Section 5.1.4, “Server System Variables”
Section 10.2, “Setting the Error Message Language”
Section 2.12.1.1, “Upgrading from MySQL 5.1 to 5.5”
Section 2.12.1.2, “Upgrading from MySQL 5.4 to 5.5”
Section 1.4, “What Is New in MySQL 5.5”

lc_time_names

Section 12.7, “Date and Time Functions”

Section 5.2.4.3, “Mixed Binary Logging Format”
Section 10.7, “MySQL Server Locale Support”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Section 12.5, “String Functions”

license

Section 5.1.4, “Server System Variables”

local

Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 13.2.7, “LOAD XML Syntax”
Section 2.10.4, “MySQL Source-Configuration Options”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

local_infile

Section 5.1.4, “Server System Variables”

lock_wait_timeout

Section 5.1.4, “Server System Variables”

locked_in_memory

Section 5.1.4, “Server System Variables”

log

Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

log_bin

Section 16.1.3.4, “Binary Log Options and Variables”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 5.1.4, “Server System Variables”

log_bin_trust_function_creators

Section 16.1.3.4, “Binary Log Options and Variables”
Section 19.7, “Binary Logging of Stored Programs”
Section 5.1.4, “Server System Variables”
Section 1.4, “What Is New in MySQL 5.5”

log_bin_use_v

Section 16.1.3.4, “Binary Log Options and Variables”

log_error

Section 5.1.4, “Server System Variables”

log_output

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.4, “Server System Variables”
Section 5.2.3, “The General Query Log”

Section 5.2.5, “The Slow Query Log”

log_queries_not_using_indexes

Section 5.1.4, “Server System Variables”

Section 5.2.5, “The Slow Query Log”

log_slave_updates

Section 16.1.3.4, “Binary Log Options and Variables”

log_slow_queries

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

log_warnings

Section 5.1.4, “Server System Variables”

Section 5.2.2, “The Error Log”

long_query_time

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

Section 5.2, “MySQL Server Logs”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.1.3, “Server Command Options”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

Section 5.2.5, “The Slow Query Log”

low_priority_updates

Section 5.1.4, “Server System Variables”

Section 8.10.2, “Table Locking Issues”

lower_case_file_system

Section 5.1.4, “Server System Variables”

lower_case_table_names

Section 14.3.5.4, “FOREIGN KEY Constraints”

Section 13.7.1.3, “GRANT Syntax”

Section 13.7.1.5, “REVOKE Syntax”

Section 10.1.7.9, “Collation and INFORMATION_SCHEMA Searches”

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

Section 1.7, “How to Report Bugs or Problems”

Section 9.2.2, “Identifier Case Sensitivity”

Section 21.1, “Performance Schema Quick Start”

Section 16.4.1.35, “Replication and Variables”

Section 5.1.4, “Server System Variables”

max_allowed_packet

Section 12.16.1, “GROUP BY (Aggregate) Functions”

Section C.5.2.3, “Lost connection to MySQL server”

Section C.5.2.9, “MySQL server has gone away”

Section 22.8.7.26, “mysql_stmt_send_long_data()”

Section 22.8.3.71, “mysql_use_result()”

Section C.5.2.10, “Packet too large”

Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”

Section C.5.2.11, “Communication Errors and Aborted Connections”

Section 12.3.2, “Comparison Functions and Operators”

Section C.5.5.6, “Deleting Rows from Related Tables”

Section 8.11.4.1, “How MySQL Uses Memory”

Section 22.8, “MySQL C API”

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

Section 16.4.1.19, “Replication and

max_allowed_packet”

Section 16.1.3.2, “Replication Master Options and Variables”

Section 5.1.4, “Server System Variables”

Section 12.5, “String Functions”

Section 11.4.3, “The BLOB and TEXT Types”

Section 4.2.3.3, “Using Option Files”

max_binlog_cache_size

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.2.4, “The Binary Log”

max_binlog_size

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.2, “MySQL Server Logs”

Section 5.2.6, “Server Log Maintenance”

Section 5.1.4, “Server System Variables”

Section 5.2.4, “The Binary Log”

Section 16.2.2.1, “The Slave Relay Log”

max_binlog_stmt_cache_size

Section 16.1.3.4, “Binary Log Options and Variables”

max_connect_errors

Section 13.7.6.3, “FLUSH Syntax”

Section C.5.2.6, “Host 'host_name' is blocked”

Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”

Section 5.1.4, “Server System Variables”

max_connections

Section C.5.2.18, “'File' Not Found and Similar Errors”

Section C.5.2.7, “Too many connections”

M

[[index top \[4449\]](#)]

Section 23.4.1.4, “Debugging `mysqld` under `gdb`”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”
Section 21.8, “Performance Schema System Variables”
Section 6.2.1, “Privileges Provided by MySQL”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 8.11.6.2, “Thread Pool Operation”

`max_delayed_threads`

Section 21.8, “Performance Schema System Variables”
Section 5.1.4, “Server System Variables”

`max_error_count`

Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 13.6.7.3, “`RESIGNAL` Syntax”
Section 13.7.5.18, “`SHOW ERRORS` Syntax”
Section 13.7.5.41, “`SHOW WARNINGS` Syntax”
Section 5.1.4, “Server System Variables”

`max_heap_table_size`

Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”
Section E.10.3, “Limits on Table Size”
Section 16.4.1.20, “Replication and `MEMORY` Tables”
Section 16.4.1.35, “Replication and Variables”
Section E.3, “Restrictions on Server-Side Cursors”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 14.6, “The `MEMORY` Storage Engine”

`max_insert_delayed_threads`

Section 5.1.4, “Server System Variables”

`max_join_size`

Section 4.5.1.1, “`mysql` Options”
Section 8.8.2, “`EXPLAIN` Output Format”
Section 13.7.4, “`SET` Syntax”
Section 5.1.4, “Server System Variables”
Section 5.1.5, “Using System Variables”

`max_length_for_sort_data`

Section 8.13.9, “`ORDER BY` Optimization”
Section 5.1.4, “Server System Variables”

`max_long_data_size`

Section 22.8.7.26, “`mysql_stmt_send_long_data()`”
Section 5.1.4, “Server System Variables”

`max_prepared_stmt_count`

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”
Section 13.5, “SQL Syntax for Prepared Statements”

`max_relay_log_size`

Section 16.1.3.4, “Binary Log Options and Variables”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.1.4, “Server System Variables”
Section 16.2.2.1, “The Slave Relay Log”

`max_seeks_for_key`

Section 14.3.15, “Limits on `InnoDB` Tables”
Section 5.1.4, “Server System Variables”

`max_sort_length`

Section 13.1.17, “`CREATE TABLE` Syntax”
Section C.5.8, “Known Issues in MySQL”
Section 5.1.4, “Server System Variables”
Section 11.4.3, “The `BLOB` and `TEXT` Types”

`max_sp_recursion_depth`

Section 5.1.4, “Server System Variables”
Section 19.2.1, “Stored Routine Syntax”

`max_tmp_tables`

Section 5.1.4, “Server System Variables”

`max_user_connections`

Section 13.7.1.3, “`GRANT` Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.4, “Server System Variables”
Section 6.3.4, “Setting Account Resource Limits”

`max_write_lock_count`

Section 5.1.4, “Server System Variables”
Section 8.10.2, “Table Locking Issues”

`metadata_locks_cache_size`

Section 5.1.4, “Server System Variables”

`min_examined_row_limit`

Section 5.1.4, “Server System Variables”
Section 5.2.5, “The Slow Query Log”

`myisam_data_pointer_size`

Section 13.1.17, “`CREATE TABLE` Syntax”
Section E.10.3, “Limits on Table Size”
Section 5.1.4, “Server System Variables”

`myisam_max_sort_file_size`

Section 14.5.1, “`MyISAM` Startup Options”

Section 18.5, “Restrictions and Limitations on Partitioning”
Section 5.1.4, “Server System Variables”
Section 8.6.3, “Speed of `REPAIR TABLE` Statements”

`myisam_mmap_size`

Section 5.1.4, “Server System Variables”

`myisam_recover_options`

Section 5.1.4, “Server System Variables”

`myisam_repair_threads`

Section 5.1.4, “Server System Variables”

`myisam_sort_buffer_size`

Section 4.6.3.1, “`myisamchk` General Options”
Section 13.1.7, “`ALTER TABLE` Syntax”
Section 14.5.1, “`MyISAM` Startup Options”
Section 5.1.4, “Server System Variables”
Section 8.6.3, “Speed of `REPAIR TABLE` Statements”

`myisam_stats_method`

Section 8.3.7, “`InnoDB` and `MyISAM` Index Statistics Collection”
Section 5.1.4, “Server System Variables”

`myisam_use_mmap`

Section 8.11.4.1, “How MySQL Uses Memory”
Section 5.1.4, “Server System Variables”

N

[index top [4449]]

`named_pipe`

Section 5.1.4, “Server System Variables”

`ndb_autoincrement_prefetch_sz`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_cache_check_time`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_deferred_constraints`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_distribution`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_extra_logging`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_force_send`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_join_pushdown`

Section 8.8.2, “`EXPLAIN` Output Format”
Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_log_apply_status`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_log_bin`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_log_binlog_index`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_log_empty_epochs`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_log_transaction_id`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_optimized_node_selection`

Section 17.3.4.3, “MySQL Cluster System Variables”
Section 17.5.6.3, “Using `CLUSTERLOG STATISTICS` in the MySQL Cluster Management Client”

`ndb_report_thresh_binlog_epoch_slip`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_report_thresh_binlog_mem_usage`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_table_no_logging`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_table_temporary`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_use_copying_alter_table`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_use_exact_count`

Section 17.3.4.3, “MySQL Cluster System Variables”

`ndb_use_transactions`

Section 17.3.4.3, “MySQL Cluster System Variables”

ndbinfo_database

Section 17.3.4.3, “MySQL Cluster System Variables”

ndbinfo_max_bytes

Section 17.3.4.3, “MySQL Cluster System Variables”

ndbinfo_max_rows

Section 17.3.4.3, “MySQL Cluster System Variables”

ndbinfo_show_hidden

Section 17.3.4.3, “MySQL Cluster System Variables”

Section 17.5.10.4, “The `ndbinfo cluster_operations` Table”

Section 17.5.10.5, “The `ndbinfo cluster_transactions` Table”

Section 17.5.10.15, “The `ndbinfo server_operations` Table”

Section 17.5.10.16, “The `ndbinfo server_transactions` Table”

ndbinfo_table_prefix

Section 17.3.4.3, “MySQL Cluster System Variables”

ndbinfo_version

Section 17.3.4.3, “MySQL Cluster System Variables”

net_buffer_length

Section 4.5.1.1, “mysql Options”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 8.11.4.1, “How MySQL Uses Memory”

Section 22.8, “MySQL C API”

Section 5.1.4, “Server System Variables”

net_read_timeout

Section C.5.2.3, “Lost connection to MySQL server”

Section 5.1.4, “Server System Variables”

net_retry_count

Section 22.8.3.49, “mysql_options()”

Section 5.1.4, “Server System Variables”

net_write_timeout

Section 5.1.4, “Server System Variables”

new

Section 5.1.4, “Server System Variables”

O

[index top [4449]]

old

Section 13.2.9.3, “Index Hint Syntax”

Section 5.1.4, “Server System Variables”

old_alter_table

Section 13.1.7, “ALTER TABLE Syntax”

Section 5.1.4, “Server System Variables”

old_passwords

Section C.5.2.4, “Client does not support authentication protocol”

Section 13.7.1.6, “SET PASSWORD Syntax”

Section 6.3.5, “Assigning Account Passwords”

Section 12.13, “Encryption and Compression Functions”

Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”

Section 22.9, “MySQL PHP API”

Section 6.1.2.4, “Password Hashing in MySQL”

Section 5.1.4, “Server System Variables”

one_shot

Section 5.1.4, “Server System Variables”

open_files_limit

Section C.5.2.18, “'File' Not Found and Similar Errors”

Section 21.8, “Performance Schema System Variables”

Section 18.5, “Restrictions and Limitations on Partitioning”

Section 5.1.4, “Server System Variables”

optimizer_prune_level

Section 8.8.5.1, “Controlling Query Plan Evaluation”

Section 5.1.4, “Server System Variables”

optimizer_search_depth

Section 8.8.5.1, “Controlling Query Plan Evaluation”

Section 5.1.4, “Server System Variables”

optimizer_switch

Section 8.8.5.2, “Controlling Switchable Optimizations”

Section 8.13.3, “Engine Condition Pushdown Optimization”

Section 5.1.4, “Server System Variables”

P

[index top [4449]]

performance_schema

Section 21.1, “Performance Schema Quick Start”

Section 21.2.2, “Performance Schema Startup Configuration”
Section 21.8, “Performance Schema System Variables”

performance_schema_events_waits_history_long_size
Section 13.7.5.16, “SHOW ENGINE Syntax”
Section 21.8, “Performance Schema System Variables”
Section 21.7, “Performance Schema Table Descriptions”
Section 21.7.3.3, “The `events_waits_history_long` Table”

performance_schema_events_waits_history_size
Section 13.7.5.16, “SHOW ENGINE Syntax”
Section 21.8, “Performance Schema System Variables”
Section 21.7, “Performance Schema Table Descriptions”
Section 21.7.3.2, “The `events_waits_history` Table”

performance_schema_max_cond_classes
Section 21.8, “Performance Schema System Variables”

performance_schema_max_cond_instances
Section 21.8, “Performance Schema System Variables”

performance_schema_max_file_classes
Section 21.8, “Performance Schema System Variables”

performance_schema_max_file_handles
Section 21.8, “Performance Schema System Variables”

performance_schema_max_file_instances
Section 21.8, “Performance Schema System Variables”

performance_schema_max_mutex_classes
Section 21.5, “Performance Schema Status Monitoring”
Section 21.8, “Performance Schema System Variables”

performance_schema_max_mutex_instances
Section 21.8, “Performance Schema System Variables”

performance_schema_max_rwlock_classes
Section 21.8, “Performance Schema System Variables”

performance_schema_max_rwlock_instances
Section 21.8, “Performance Schema System Variables”

performance_schema_max_table_handles
Section 21.8, “Performance Schema System Variables”

performance_schema_max_table_instances
Section 21.8, “Performance Schema System Variables”

performance_schema_max_thread_classes
Section 21.8, “Performance Schema System Variables”

performance_schema_max_thread_instances
Section 13.7.5.16, “SHOW ENGINE Syntax”
Section 21.8, “Performance Schema System Variables”

pid_file
Section 5.1.4, “Server System Variables”

plugin_dir
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 13.7.3.3, “INSTALL PLUGIN Syntax”
Section 13.7.5.26, “SHOW PLUGINS Syntax”
Section 6.1.2.2, “Administrator Guidelines for Password Security”
Section 23.2.4.3, “Compiling and Installing Plugin Libraries”
Section 23.3.2.5, “Compiling and Installing User-Defined Functions”

Section 5.1.8.1, “Installing and Uninstalling Plugins”
Section 6.3.10.1, “Installing the Audit Log Plugin”
Section 6.3.6.2.1, “Installing the PAM Authentication Plugin”
Section 6.3.6.3.1, “Installing the Windows Authentication Plugin”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.3.6, “Pluggable Authentication”
Section 14.2.1, “Pluggable Storage Engine Architecture”
Section 23.2.2, “Plugin API Components”
Section E.9, “Restrictions on Pluggable Authentication”
Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 20.13, “The `INFORMATION_SCHEMA.PLUGINS` Table”
Section 6.3.6.2, “The PAM Authentication Plugin”
Section 6.3.6.5, “The Socket Peer-Credential Authentication Plugin”
Section 6.3.6.6, “The Test Authentication Plugin”
Section 6.3.6.3, “The Windows Native Authentication Plugin”
Section 8.11.6.1, “Thread Pool Components and Installation”
Section 2.11.1, “Unix Postinstallation Procedures”
Section 23.3.2.6, “User-Defined Function Security Precautions”
Section 23.2.4.9.3, “Using the Authentication Plugins”
Section 23.2.4.6, “Writing `INFORMATION_SCHEMA.PLUGINS`”
Section 23.2.4.8, “Writing Audit Plugins”

Section 23.2.4.5, “Writing Daemon Plugins”
Section 23.2.4.4, “Writing Full-Text Parser Plugins”
Section 23.2.4.7, “Writing Semisynchronous Replication Plugins”

port

Section C.5.2.2, “Can't connect to [local] MySQL server”
Section 5.1.4, “Server System Variables”

preload_buffer_size

Section 5.1.4, “Server System Variables”

profiling

Section 13.7.5.31, “SHOW PROFILE Syntax”
Section 5.1.4, “Server System Variables”
Section 20.15, “The INFORMATION_SCHEMA PROFILING Table”

profiling_history_size

Section 13.7.5.31, “SHOW PROFILE Syntax”
Section 5.1.4, “Server System Variables”

protocol_version

Section 5.1.4, “Server System Variables”

proxy_user

Section 6.3.7, “Proxy Users”
Section 5.1.4, “Server System Variables”

pseudo_thread_id

Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”

Q

[index top [4449]]

query_alloc_block_size

Section 5.1.4, “Server System Variables”

query_cache_limit

Section 8.9.3.3, “Query Cache Configuration”
Section 5.1.4, “Server System Variables”

query_cache_min_res_unit

Section 8.9.3.3, “Query Cache Configuration”
Section 5.1.4, “Server System Variables”

query_cache_size

Section 8.9.3.3, “Query Cache Configuration”

Section 5.1.4, “Server System Variables”
Section 8.9.3, “The MySQL Query Cache”
Section 5.1.5, “Using System Variables”

query_cache_type

Section 13.2.9, “SELECT Syntax”
Section 8.9.3.2, “Query Cache SELECT Options”
Section 8.9.3.3, “Query Cache Configuration”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

query_cache_wlock_invalidate

Section 5.1.4, “Server System Variables”

query_prealloc_size

Section 5.1.4, “Server System Variables”

R

[index top [4449]]

rand_seed

Section 5.1.4, “Server System Variables”

range_alloc_block_size

Section 5.1.4, “Server System Variables”

read_buffer_size

Section 8.11.4.1, “How MySQL Uses Memory”
Section 5.1.4, “Server System Variables”
Section 8.6.3, “Speed of REPAIR TABLE Statements”
Section 1.4, “What Is New in MySQL 5.5”

read_only

Section 13.7.1.6, “SET PASSWORD Syntax”
Section 6.3.5, “Assigning Account Passwords”
Section 16.3.1.3, “Backing Up a Master or Slave by Making It Read Only”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.35, “Replication and Variables”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.4, “Server System Variables”

read_rnd_buffer_size

Section 8.13.9, “ORDER BY Optimization”
Section 8.11.4.1, “How MySQL Uses Memory”
Section 5.1.4, “Server System Variables”
Section 8.11.2, “Tuning Server Parameters”

relay_log

Section 16.1.3.3, “Replication Slave Options and Variables”

relay_log_index

Section 16.1.3.3, “Replication Slave Options and Variables”

relay_log_info_file

Section 16.1.3.3, “Replication Slave Options and Variables”

relay_log_purge

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 5.1.4, “Server System Variables”

relay_log_recovery

Section 16.1.3.3, “Replication Slave Options and Variables”

relay_log_space_limit

Section 8.12.5.6, “Replication Slave I/O Thread States”
Section 5.1.4, “Server System Variables”

report_host

Section 5.1.4, “Server System Variables”

report_password

Section 5.1.4, “Server System Variables”

report_port

Section 5.1.4, “Server System Variables”

report_user

Section 5.1.4, “Server System Variables”

Rpl_recovery_rank

Section 13.7.5.34, “SHOW SLAVE HOSTS Syntax”

rpl_recovery_rank

Section 16.1.3.3, “Replication Slave Options and Variables”

rpl_semi_sync_master_enabled

Section 16.3.8.1, “Semisynchronous Replication Administrative Interface”
Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 16.3.8.3, “Semisynchronous Replication Monitoring”
Section 5.1.4, “Server System Variables”

rpl_semi_sync_master_timeout

Section 16.3.8.1, “Semisynchronous Replication Administrative Interface”

Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.4, “Server System Variables”

rpl_semi_sync_master_trace_level

Section 5.1.4, “Server System Variables”

rpl_semi_sync_master_wait_no_slave

Section 5.1.4, “Server System Variables”

rpl_semi_sync_slave_enabled

Section 16.3.8.1, “Semisynchronous Replication Administrative Interface”
Section 16.3.8.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.4, “Server System Variables”

rpl_semi_sync_slave_trace_level

Section 5.1.4, “Server System Variables”

S

[[index top \[4449\]](#)]

secure_auth

Section 6.1.2.4, “Password Hashing in MySQL”
Section 5.1.4, “Server System Variables”

secure_file_priv

Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 13.2.9.1, “SELECT ... INTO Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.2.1, “Privileges Provided by MySQL”
Section 5.1.4, “Server System Variables”
Section 12.5, “String Functions”

server_id

Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 12.15, “Miscellaneous Functions”
Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”
Section 17.3.4.3, “MySQL Cluster System Variables”
Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”
Section 5.1.4, “Server System Variables”
Section 6.3.10.2, “The Audit Log File”
Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

server_id_bits

Section 17.3.4.3, “MySQL Cluster System Variables”

Section 17.3.4.2, “MySQL Server Options for MySQL Cluster”

shared_memory

Section 5.1.4, “Server System Variables”

shared_memory_base_name

Section 5.1.4, “Server System Variables”

skip_external_locking

Section 8.10.5, “External Locking”

Section 5.1.4, “Server System Variables”

skip_name_resolve

Section 5.1.4, “Server System Variables”

skip_networking

Section 5.1.4, “Server System Variables”

skip_show_database

Section 5.1.4, “Server System Variables”

slave_allow_batching

Section 17.3.4.3, “MySQL Cluster System Variables”

Section 17.6.6, “Starting MySQL Cluster Replication (Single Replication Channel)”

slave_compressed_protocol

Section 16.1.3.3, “Replication Slave Options and Variables”

slave_exec_mode

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 17.6.10, “MySQL Cluster Replication: Multi-Master and Circular Replication”

Section 16.4.1.20, “Replication and `MEMORY` Tables”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 16.1.2.2, “Usage of Row-Based Logging and Replication”

slave_load_tmpdir

Section 13.2.6, “`LOAD DATA INFILE` Syntax”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.1.4, “Server System Variables”

slave_max_allowed_packet

Section 16.1.3.2, “Replication Master Options and Variables”

slave_net_timeout

Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”

Section 16.1.4.1, “Checking Replication Status”

Section 16.4.1.18, “Replication and Master or Slave Shutdowns”

Section 8.12.5.6, “Replication Slave I/O Thread States”

Section 16.1.3.3, “Replication Slave Options and Variables”

Section 5.1.4, “Server System Variables”

slave_skip_errors

Section 16.1.3.3, “Replication Slave Options and Variables”

slave_transaction_retries

Section 16.4.1.28, “Replication Retries and Timeouts”

Section 16.1.3.3, “Replication Slave Options and Variables”

slave_type_conversions

Section 17.6.3, “Known Issues in MySQL Cluster Replication”

Section 16.1.3.3, “Replication Slave Options and Variables”

slow_launch_time

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

slow_query_log

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”

Section 5.1.4, “Server System Variables”

Section 5.2.5, “The Slow Query Log”

slow_query_log_file

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”

Section 5.1.4, “Server System Variables”

Section 5.2.5, “The Slow Query Log”

socket

Section 5.1.4, “Server System Variables”

sort_buffer_size

Section 8.13.9, “`ORDER BY` Optimization”

Section 7.6.3, “How to Repair `MyISAM` Tables”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

sql_auto_is_null

Section 12.3.2, “Comparison Functions and Operators”

Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Section 5.2.4, “The Binary Log”

sql_big_selects

Section 5.1.4, “Server System Variables”

sql_buffer_result

Section 5.1.4, “Server System Variables”

sql_log_bin

Section 13.4.1.3, “SET sql_log_bin Syntax”
Section 17.1.6.8, “Issues Exclusive to MySQL Cluster”
Section 17.6.3, “Known Issues in MySQL Cluster Replication”
Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”
Section 5.1.4, “Server System Variables”
Section 16.4.3, “Upgrading a Replication Setup”

sql_log_off

Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.4, “Server System Variables”
Section 5.2.3, “The General Query Log”

sql_log_update

Section 5.1.4, “Server System Variables”

sql_mode

Section 13.1.11, “CREATE EVENT Syntax”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.19, “CREATE TRIGGER Syntax”
Section 13.7.5.14, “SHOW CREATE VIEW Syntax”
Section 13.7.5.39, “SHOW TRIGGERS Syntax”
Section 13.6.7.4.2, “Effect of Signals on Handlers, Cursors, and Statements”
Section 12.18.3, “Expression Handling”
Section 1.7, “How to Report Bugs or Problems”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section C.5.5.2, “Problems Using DATE Columns”
Section 16.4.1.35, “Replication and Variables”
Section 1.8.3, “Running MySQL in ANSI Mode”
Section 1.8.2, “Selecting SQL Modes”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 20.17, “The INFORMATION_SCHEMA ROUTINES Table”
Section 20.25, “The INFORMATION_SCHEMA TRIGGERS Table”

Section 20.27, “The INFORMATION_SCHEMA VIEWS Table”

Section 5.2.4, “The Binary Log”
Section 5.1.5, “Using System Variables”

sql_notes

Section 13.7.5.41, “SHOW WARNINGS Syntax”
Section 5.1.4, “Server System Variables”

sql_quote_show_create

Section 13.7.5.8, “SHOW CREATE DATABASE Syntax”
Section 13.7.5.12, “SHOW CREATE TABLE Syntax”
Section 5.1.4, “Server System Variables”

sql_safe_updates

Section 5.1.4, “Server System Variables”

sql_select_limit

Section 5.1.4, “Server System Variables”

sql_slave_skip_counter

Section 13.4.2.4, “SET GLOBAL sql_slave_skip_counter Syntax”
Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
Section 16.1.3.3, “Replication Slave Options and Variables”

sql_warnings

Section 5.1.4, “Server System Variables”

ssl_ca

Section 5.1.4, “Server System Variables”

ssl_capath

Section 5.1.4, “Server System Variables”

ssl_cert

Section 5.1.4, “Server System Variables”

ssl_cipher

Section 5.1.4, “Server System Variables”

ssl_key

Section 5.1.4, “Server System Variables”

storage_engine

Section 13.1.14, “CREATE LOGFILE GROUP Syntax”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”

Section 1.4, “What Is New in MySQL 5.5”

stored_program_cache

Section 5.1.4, “Server System Variables”

sync_binlog

Section 14.3.4, “InnoDB Startup Options and System Variables”

Section 16.1.3.4, “Binary Log Options and Variables”

Section 16.4.1.18, “Replication and Master or Slave Shutdowns”

Section 5.2.4, “The Binary Log”

sync_frm

Section 5.1.4, “Server System Variables”

sync_master_info

Section 16.1.3.3, “Replication Slave Options and Variables”

sync_relay_log

Section 16.1.3.3, “Replication Slave Options and Variables”

sync_relay_log_info

Section 16.4.1.18, “Replication and Master or Slave Shutdowns”

Section 16.1.3.3, “Replication Slave Options and Variables”

system_time_zone

Section 10.6, “MySQL Server Time Zone Support”

Section 5.1.3, “Server Command Options”

Section 5.1.4, “Server System Variables”

T

[index top [4449]]

table_definition_cache

Section 5.1.4, “Server System Variables”

table_lock_wait_timeout

Section 5.1.4, “Server System Variables”

table_open_cache

Section C.5.2.18, “`'File' Not Found` and Similar Errors”

Section 8.12.5.2, “General Thread States”

Section 8.4.3.1, “How MySQL Opens and Closes Tables”

Section 8.11.4.1, “How MySQL Uses Memory”

Section 5.1.3, “Server Command Options”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

Section 8.11.2, “Tuning Server Parameters”

table_type

Section 5.1.4, “Server System Variables”

thread_cache_size

Section 23.4.1.4, “Debugging `mysqld` under `gdb`”

Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

thread_concurrency

Section 5.1.4, “Server System Variables”

thread_handling

Section 5.1.4, “Server System Variables”

Section 8.11.6.1, “Thread Pool Components and Installation”

thread_pool_algorithm

Section 5.1.4, “Server System Variables”

Section 20.30.2, “The `INFORMATION_SCHEMA` `TP_THREAD_GROUP_STATE` Table”

Section 8.11.6.1, “Thread Pool Components and Installation”

thread_pool_high_priority_connecti

Section 5.1.4, “Server System Variables”

Section 8.11.6.1, “Thread Pool Components and Installation”

Section 8.11.6.2, “Thread Pool Operation”

thread_pool_max_unused_threads

Section 5.1.4, “Server System Variables”

Section 8.11.6.1, “Thread Pool Components and Installation”

thread_pool_prio_kickup_timer

Section 5.1.4, “Server System Variables”

Section 20.30.2, “The `INFORMATION_SCHEMA` `TP_THREAD_GROUP_STATE` Table”

Section 20.30.3, “The `INFORMATION_SCHEMA` `TP_THREAD_GROUP_STATS` Table”

Section 8.11.6.1, “Thread Pool Components and Installation”

Section 8.11.6.2, “Thread Pool Operation”

Section 8.11.6.3, “Thread Pool Tuning”

thread_pool_size

Section 5.1.4, “Server System Variables”
Section 8.11.6.1, “Thread Pool Components and Installation”
Section 8.11.6.2, “Thread Pool Operation”
Section 8.11.6.3, “Thread Pool Tuning”

thread_pool_stall_limit

Section 5.1.4, “Server System Variables”
Section 20.30.2, “The `INFORMATION_SCHEMA` `TP_THREAD_GROUP_STATE` Table”
Section 20.30.3, “The `INFORMATION_SCHEMA` `TP_THREAD_GROUP_STATS` Table”
Section 8.11.6.1, “Thread Pool Components and Installation”
Section 8.11.6.2, “Thread Pool Operation”
Section 8.11.6.3, “Thread Pool Tuning”

thread_stack

Section 8.11.4.1, “How MySQL Uses Memory”
Section 5.1.4, “Server System Variables”
Section 19.2.1, “Stored Routine Syntax”

time_format

Section 5.1.4, “Server System Variables”

time_zone

Section 13.1.11, “`CREATE EVENT` Syntax”
Section 12.7, “Date and Time Functions”
Section 19.4.4, “Event Metadata”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 10.6, “MySQL Server Time Zone Support”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 11.3.1, “The `DATE`, `DATETIME`, and `TIMESTAMP` Types”

timed_mutexes

Section 13.7.5.16, “`SHOW ENGINE` Syntax”
Section 5.1.4, “Server System Variables”

timestamp

Section 14.11.3, “`FEDERATED` Storage Engine Notes and Tips”
Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”

tmp_table_size

Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”

Section E.3, “Restrictions on Server-Side Cursors”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

tmpdir

Section C.5.2.13, “Can't create/write to file”
Section 13.2.6, “`LOAD DATA INFILE` Syntax”
Section 8.13.9, “`ORDER BY` Optimization”
Section 16.3.1.2, “Backing Up Raw Data from a Slave”
Section 7.2, “Database Backup Methods”
Section 16.1.3.3, “Replication Slave Options and Variables”
Section 5.1.4, “Server System Variables”

transaction_alloc_block_size

Section 5.1.4, “Server System Variables”

transaction_allow_batching

Section 17.3.4.3, “MySQL Cluster System Variables”

transaction_prealloc_size

Section 5.1.4, “Server System Variables”

tx_isolation

Section 13.3.6, “`SET TRANSACTION` Syntax”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

U

[[index top \[4449\]](#)]

unique_checks

Section 5.2.4.3, “Mixed Binary Logging Format”
Section 16.4.1.35, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Section 5.2.4, “The Binary Log”

updatable_views_with_limit

Section 5.1.4, “Server System Variables”
Section 19.5.3, “Updatable and Insertable Views”

V

[[index top \[4449\]](#)]

version

Section 14.3.4, “`InnoDB` Startup Options and System Variables”
Section 12.14, “Information Functions”
Section 5.1.4, “Server System Variables”
Section 6.3.10.2, “The Audit Log File”

version_comment

Section 5.1.4, “Server System Variables”

version_compile_machine

Section 5.1.4, “Server System Variables”

version_compile_os

Section 5.1.4, “Server System Variables”

W

[[index top \[4449\]](#)]

wait_timeout

Section C.5.2.9, “MySQL server has gone away”

Section 22.8.3.52, “`mysql_real_connect()`”

Section C.5.2.11, “Communication Errors and Aborted Connections”

Section 5.1.4, “Server System Variables”

warning_count

Section 13.7.5.18, “`SHOW ERRORS` Syntax”

Section 13.7.5.41, “`SHOW WARNINGS` Syntax”

Section 13.6.7.4.2, “Effect of Signals on Handlers, Cursors, and Statements”

Section 5.1.4, “Server System Variables”

Section C.1, “Sources of Error Information”

Status Variable Index

[A](#) | [B](#) | [C](#) | [D](#) | [F](#) | [H](#) | [I](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#)

A

[\[index top \[4471\]\]](#)

Aborted_clients

Section C.5.2.11, “Communication Errors and Aborted Connections”

Section 5.1.6, “Server Status Variables”

Aborted_connects

Section C.5.2.11, “Communication Errors and Aborted Connections”

Section 5.1.6, “Server Status Variables”

B

[\[index top \[4471\]\]](#)

Binlog_cache_disk_use

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.1.6, “Server Status Variables”

Section 5.2.4, “The Binary Log”

Binlog_cache_use

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.1.6, “Server Status Variables”

Section 5.2.4, “The Binary Log”

Binlog_stmt_cache_disk_use

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.1.6, “Server Status Variables”

Binlog_stmt_cache_use

Section 16.1.3.4, “Binary Log Options and Variables”

Section 5.1.6, “Server Status Variables”

Bytes_received

Section 5.1.6, “Server Status Variables”

Bytes_sent

Section 5.1.6, “Server Status Variables”

C

[\[index top \[4471\]\]](#)

Com_flush

Section 5.1.6, “Server Status Variables”

Com_stmt_reprepare

Section 13.5.4, “Automatic Prepared Statement Repreparation”

Compression

Section 5.1.6, “Server Status Variables”

Connections

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

Created_tmp_disk_tables

Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

Created_tmp_files

Section 5.1.6, “Server Status Variables”

Created_tmp_tables

Section 13.7.5.36, “`SHOW STATUS` Syntax”

Section 8.4.3.3, “How MySQL Uses Internal Temporary Tables”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

D

[\[index top \[4471\]\]](#)

Delayed_errors

Section 5.1.6, “Server Status Variables”

Delayed_insert_threads

Section 13.2.5.2, “`INSERT DELAYED` Syntax”

Section 5.1.6, “Server Status Variables”

Delayed_writes

Section 13.2.5.2, “`INSERT DELAYED` Syntax”

Section 5.1.6, “Server Status Variables”

F

[\[index top \[4471\]\]](#)

Flush_commands

Section 5.1.6, “Server Status Variables”

H

[[index top \[4471\]](#)]

Handler_commit

Section 5.1.6, “Server Status Variables”

Handler_delete

Section 5.1.6, “Server Status Variables”

Handler_discover

Section 17.3.4.4, “MySQL Cluster Status Variables”

Handler_prepare

Section 5.1.6, “Server Status Variables”

Handler_read_first

Section 5.1.6, “Server Status Variables”

Handler_read_key

Section 5.1.6, “Server Status Variables”

Handler_read_last

Section 5.1.6, “Server Status Variables”

Handler_read_next

Section 5.1.6, “Server Status Variables”

Handler_read_prev

Section 5.1.6, “Server Status Variables”

Handler_read_rnd

Section 5.1.6, “Server Status Variables”

Handler_read_rnd_next

Section 5.1.6, “Server Status Variables”

Handler_rollback

Section 5.1.6, “Server Status Variables”

Handler_savepoint

Section 5.1.6, “Server Status Variables”

Handler_savepoint_rollback

Section 5.1.6, “Server Status Variables”

Handler_update

Section 5.1.6, “Server Status Variables”

Handler_write

Section 5.1.6, “Server Status Variables”

I

[[index top \[4471\]](#)]

Innodb_buffer_pool_pages_data

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_pages_dirty

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_pages_flushed

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_pages_free

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_pages_latched

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_pages_misc

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_pages_total

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_read_ahead

Section 5.1.6, “Server Status Variables”

Section 1.4, “What Is New in MySQL 5.5”

Innodb_buffer_pool_read_ahead_evicted

Section 5.1.6, “Server Status Variables”

Section 1.4, “What Is New in MySQL 5.5”

Innodb_buffer_pool_read_requests

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_reads

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_wait_free

Section 5.1.6, “Server Status Variables”

Innodb_buffer_pool_write_requests

Section 5.1.6, “Server Status Variables”

Innodb_data_fsyncs

Section 5.1.6, “Server Status Variables”

InnoDB_data_pending_fsyncs

Section 5.1.6, “Server Status Variables”

InnoDB_data_pending_reads

Section 5.1.6, “Server Status Variables”

InnoDB_data_pending_writes

Section 5.1.6, “Server Status Variables”

InnoDB_data_read

Section 5.1.6, “Server Status Variables”

InnoDB_data_reads

Section 5.1.6, “Server Status Variables”

InnoDB_data_writes

Section 5.1.6, “Server Status Variables”

InnoDB_data_written

Section 5.1.6, “Server Status Variables”

InnoDB_dblwr_pages_written

Section 5.1.6, “Server Status Variables”

InnoDB_dblwr_writes

Section 5.1.6, “Server Status Variables”

InnoDB_have_atomic_builtins

Section 1.4.3, “Diagnostic and Monitoring Capabilities”

Section 1.4.1, “Scalability Improvements”

Section 5.1.6, “Server Status Variables”

InnoDB_log_waits

Section 5.1.6, “Server Status Variables”

InnoDB_log_write_requests

Section 5.1.6, “Server Status Variables”

InnoDB_log_writes

Section 5.1.6, “Server Status Variables”

InnoDB_os_log_fsyncs

Section 5.1.6, “Server Status Variables”

InnoDB_os_log_pending_fsyncs

Section 5.1.6, “Server Status Variables”

InnoDB_os_log_pending_writes

Section 5.1.6, “Server Status Variables”

InnoDB_os_log_written

Section 5.1.6, “Server Status Variables”

InnoDB_page_size

Section 5.1.6, “Server Status Variables”

InnoDB_pages_created

Section 5.1.6, “Server Status Variables”

InnoDB_pages_read

Section 5.1.6, “Server Status Variables”

InnoDB_pages_written

Section 5.1.6, “Server Status Variables”

InnoDB_row_lock_current_waits

Section 5.1.6, “Server Status Variables”

InnoDB_row_lock_time

Section 5.1.6, “Server Status Variables”

InnoDB_row_lock_time_avg

Section 5.1.6, “Server Status Variables”

InnoDB_row_lock_time_max

Section 5.1.6, “Server Status Variables”

InnoDB_row_lock_waits

Section 5.1.6, “Server Status Variables”

InnoDB_rows_deleted

Section 5.1.6, “Server Status Variables”

InnoDB_rows_inserted

Section 5.1.6, “Server Status Variables”

InnoDB_rows_read

Section 5.1.6, “Server Status Variables”

InnoDB_rows_updated

Section 5.1.6, “Server Status Variables”

InnoDB_truncated_status_writes

Section 5.1.6, “Server Status Variables”

K

[index top [4471]]

Key_blocks_not_flushed

Section 5.1.6, “Server Status Variables”

Key_blocks_unused

Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

Key_blocks_used

Section 5.1.6, “Server Status Variables”

Key_read_requests

Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

Key_reads

Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

Key_write_requests

Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

Key_writes

Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

L

[[index top \[4471\]](#)]

Last_query_cost

Section 5.1.6, “Server Status Variables”

M

[[index top \[4471\]](#)]

Max_used_connections

Section 13.7.6.3, “`FLUSH` Syntax”
Section 5.1.6, “Server Status Variables”

N

[[index top \[4471\]](#)]

Ndb_api_bytes_received_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_bytes_received_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_bytes_received_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_bytes_sent_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_bytes_sent_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_bytes_sent_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_event_bytes_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_event_bytes_count_injector

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_event_data_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_event_data_count_injector

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_event_nodata_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_event_nodata_count_injector

Section 17.3.4.4, “MySQL Cluster Status Variables”

Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_close_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_close_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_close_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_commit_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_commit_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_commit_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_local_read_row_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_local_read_row_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_local_read_row_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_start_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_start_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_trans_start_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_uk_op_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_uk_op_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_uk_op_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_exec_complete_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_exec_complete_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_exec_complete_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_meta_request_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_meta_request_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_meta_request_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”

Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_nanos_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_nanos_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_nanos_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_scan_result_count

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_scan_result_count_session

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_api_wait_scan_result_count_slave

Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.5.15, “NDB API Statistics Counters and Variables”

Ndb_cluster_node_id

Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_config_from_host

Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_config_from_port

Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_conflict_fn_epoch

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

Ndb_conflict_fn_epoch_trans

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”
Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_conflict_fn_max

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

Ndb_conflict_fn_old

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

Ndb_conflict_trans_conflict_commit

Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_conflict_trans_row_conflict_co

Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_conflict_trans_row_reject_coun

Section 17.6.11, “MySQL Cluster Replication Conflict Resolution”

Ndb_execute_count

Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_number_of_data_nodes

Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_pruned_scan_count

Section 17.3.4.4, “MySQL Cluster Status Variables”

Ndb_pushed_queries_defined

Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.3.4.3, “MySQL Cluster System Variables”

Ndb_pushed_queries_dropped

Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.3.4.3, “MySQL Cluster System Variables”

Ndb_pushed_queries_executed

Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.3.4.3, “MySQL Cluster System Variables”

Ndb_pushed_reads

Section 17.1.4.1, “MySQL Cluster Development in MySQL Cluster NDB 7.2”
Section 17.3.4.4, “MySQL Cluster Status Variables”
Section 17.3.4.3, “MySQL Cluster System Variables”

Ndb_scan_count

Section 17.3.4.4, “MySQL Cluster Status Variables”

Not_flushed_delayed_rows

Section 13.2.5.2, “INSERT DELAYED Syntax”

Section 5.1.6, “Server Status Variables”

O

[index top [4471]]

Open_files

Section 5.1.6, “Server Status Variables”

Open_streams

Section 5.1.6, “Server Status Variables”

Open_table_definitions

Section 5.1.6, “Server Status Variables”

Open_tables

Section 5.1.6, “Server Status Variables”

Opened_files

Section 5.1.6, “Server Status Variables”

Opened_table_definitions

Section 5.1.6, “Server Status Variables”

Opened_tables

Section 8.4.3.1, “How MySQL Opens and Closes Tables”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

P

[index top [4471]]

Performance_schema_mutex_classes_lost

Section 21.5, “Performance Schema Status Monitoring”

Performance_schema_mutex_instances_lost

Section 21.5, “Performance Schema Status Monitoring”

Prepared_stmt_count

Section 5.1.6, “Server Status Variables”

Q

[index top [4471]]

Qcache_free_blocks

Section 8.9.3.3, “Query Cache Configuration”

Section 8.9.3.4, “Query Cache Status and Maintenance”

Section 5.1.6, “Server Status Variables”

Qcache_free_memory

Section 5.1.6, “Server Status Variables”

Qcache_hits

Section 8.9.3.1, “How the Query Cache Operates”

Section 5.1.6, “Server Status Variables”

Qcache_inserts

Section 5.1.6, “Server Status Variables”

Qcache_lowmem_prunes

Section 8.9.3.3, “Query Cache Configuration”

Section 8.9.3.4, “Query Cache Status and Maintenance”

Section 5.1.6, “Server Status Variables”

Qcache_not_cached

Section 5.1.6, “Server Status Variables”

Qcache_queries_in_cache

Section 8.9.3.3, “Query Cache Configuration”

Section 5.1.6, “Server Status Variables”

Qcache_total_blocks

Section 8.9.3.3, “Query Cache Configuration”

Section 8.9.3.4, “Query Cache Status and Maintenance”

Section 5.1.6, “Server Status Variables”

Queries

Section 5.1.6, “Server Status Variables”

Questions

Section 4.5.2, “mysqladmin — Client for Administering a

MySQL Server”

Section 5.1.6, “Server Status Variables”

R

[index top [4471]]

Rpl_semi_sync_master_clients

Section 16.3.8.1, “Semisynchronous Replication

Administrative Interface”

Section 16.3.8.3, “Semisynchronous Replication

Monitoring”

Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_net_avg_wait_time
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_net_wait_time
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_net_waits
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_no_times
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_no_tx
 Section 16.3.8.1, “Semisynchronous Replication Administrative Interface”
 Section 16.3.8.3, “Semisynchronous Replication Monitoring”
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_status
 Section 16.3.8.1, “Semisynchronous Replication Administrative Interface”
 Section 16.3.8.3, “Semisynchronous Replication Monitoring”
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_timefunc_failure_count
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_tx_avg_wait_time
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_tx_wait_time
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_tx_waits
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_wait_pos_backtraverse
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_wait_sessions
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_master_yes_tx
 Section 16.3.8.1, “Semisynchronous Replication Administrative Interface”
 Section 16.3.8.3, “Semisynchronous Replication Monitoring”
 Section 5.1.6, “Server Status Variables”

Rpl_semi_sync_slave_status
 Section 16.3.8.1, “Semisynchronous Replication Administrative Interface”
 Section 16.3.8.3, “Semisynchronous Replication Monitoring”
 Section 5.1.6, “Server Status Variables”

Rpl_status
 Section 5.1.6, “Server Status Variables”

S

[index top [4471]]

Select_full_join
 Section 5.1.6, “Server Status Variables”

Select_full_range_join
 Section 5.1.6, “Server Status Variables”

Select_range
 Section 5.1.6, “Server Status Variables”

Select_range_check
 Section 5.1.6, “Server Status Variables”

slave_heartbeat_period
 Section 5.1.6, “Server Status Variables”

slave_open_temp_tables
 Section 16.4.1.21, “Replication and Temporary Tables”
 Section 5.1.6, “Server Status Variables”

slave_received_heartbeats
 Section 5.1.6, “Server Status Variables”

slave_retried_transactions
 Section 5.1.6, “Server Status Variables”

slave_running
 Section 13.7.5.35, “SHOW SLAVE STATUS Syntax”
 Section 16.2.1, “Replication Implementation Details”
 Section 5.1.6, “Server Status Variables”

slow_launch_threads
 Section 5.1.6, “Server Status Variables”
 Section 5.1.4, “Server System Variables”

Slow_queries

Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

Sort_merge_passes

Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

Sort_range

Section 5.1.6, “Server Status Variables”

Sort_rows

Section 5.1.6, “Server Status Variables”

Sort_scan

Section 5.1.6, “Server Status Variables”

Ssl_accept_renegotiates

Section 5.1.6, “Server Status Variables”

Ssl_accepts

Section 5.1.6, “Server Status Variables”

Ssl_callback_cache_hits

Section 5.1.6, “Server Status Variables”

Ssl_cipher

Section 5.1.6, “Server Status Variables”
Section 6.3.8.3, “Using SSL Connections”

Ssl_cipher_list

Section 5.1.6, “Server Status Variables”

Ssl_client_connects

Section 5.1.6, “Server Status Variables”

Ssl_connect_renegotiates

Section 5.1.6, “Server Status Variables”

Ssl_ctx_verify_depth

Section 5.1.6, “Server Status Variables”

Ssl_ctx_verify_mode

Section 5.1.6, “Server Status Variables”

Ssl_default_timeout

Section 5.1.6, “Server Status Variables”

Ssl_finished_accepts

Section 5.1.6, “Server Status Variables”

Ssl_finished_connects

Section 5.1.6, “Server Status Variables”

Ssl_session_cache_hits

Section 5.1.6, “Server Status Variables”

Ssl_session_cache_misses

Section 5.1.6, “Server Status Variables”

Ssl_session_cache_mode

Section 5.1.6, “Server Status Variables”

Ssl_session_cache_overflows

Section 5.1.6, “Server Status Variables”

Ssl_session_cache_size

Section 5.1.6, “Server Status Variables”

Ssl_session_cache_timeouts

Section 5.1.6, “Server Status Variables”

Ssl_sessions_reused

Section 5.1.6, “Server Status Variables”

Ssl_used_session_cache_entries

Section 5.1.6, “Server Status Variables”

Ssl_verify_depth

Section 5.1.6, “Server Status Variables”

Ssl_verify_mode

Section 5.1.6, “Server Status Variables”

Ssl_version

Section 5.1.6, “Server Status Variables”

T

[[index top \[4471\]](#)]

Table_locks_immediate

Section 8.10.1, “Internal Locking Methods”
Section 5.1.6, “Server Status Variables”

Table_locks_waited

Section 8.10.1, “Internal Locking Methods”
Section 5.1.6, “Server Status Variables”

Tc_log_max_pages_used

Section 5.1.6, “Server Status Variables”

Tc_log_page_size

Section 5.1.6, “Server Status Variables”

Tc_log_page_waits

Section 5.1.6, “Server Status Variables”

Threads_cached

Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”

Section 5.1.6, “Server Status Variables”

Threads_connected

Section 5.1.6, “Server Status Variables”

Threads_created

Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”

Section 5.1.6, “Server Status Variables”

Section 5.1.4, “Server System Variables”

Threads_running

Section 5.1.6, “Server Status Variables”

U

[[index top \[4471\]](#)]

Uptime

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 5.1.6, “Server Status Variables”

Uptime_since_flush_status

Section 5.1.6, “Server Status Variables”

Transaction Isolation Level Index

R | S

R

[[index top \[4483\]](#)]

READ COMMITTED

[Section 14.3.9.4, “InnoDB Record, Gap, and Next-Key Locks”](#)

[Section 14.3.4, “InnoDB Startup Options and System Variables”](#)

[Section 17.1.5.3, “NDB and InnoDB Feature Usage Summary”](#)

[Section 13.3.6, “SET TRANSACTION Syntax”](#)

[Section 14.3.9.2, “Consistent Nonlocking Reads”](#)

[Section 17.1.5.1, “Differences Between the NDB and InnoDB Storage Engines”](#)

[Section 14.3.9.9, “How to Cope with Deadlocks”](#)

[Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”](#)

[Section 14.3.9.6, “Locks Set by Different SQL Statements in InnoDB”](#)

[Section 8.5.2, “Optimizing InnoDB Transaction Management”](#)

[Section 14.3.9, “The InnoDB Transaction Model and Locking”](#)

READ UNCOMMITTED

[Section 13.3.6, “SET TRANSACTION Syntax”](#)

[Section 14.3.9.2, “Consistent Nonlocking Reads”](#)

[Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”](#)

[Section 14.3.9, “The InnoDB Transaction Model and Locking”](#)

READ-COMMITTED

[Section 13.3.6, “SET TRANSACTION Syntax”](#)

[Section 5.1.3, “Server Command Options”](#)

READ-UNCOMMITTED

[Section 13.3.6, “SET TRANSACTION Syntax”](#)

[Section 5.1.3, “Server Command Options”](#)

REPEATABLE READ

[Section 14.3.9.4, “InnoDB Record, Gap, and Next-Key Locks”](#)

[Section 13.3.6, “SET TRANSACTION Syntax”](#)

[Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#)

[Section 14.3.9.2, “Consistent Nonlocking Reads”](#)

[Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”](#)

[Section 5.2.4.3, “Mixed Binary Logging Format”](#)

[Section 8.5.2, “Optimizing InnoDB Transaction Management”](#)

[Section 14.3.9, “The InnoDB Transaction Model and Locking”](#)

[Section 13.3.7, “XA Transactions”](#)

REPEATABLE-READ

[Section 13.3.6, “SET TRANSACTION Syntax”](#)

[Section 5.1.3, “Server Command Options”](#)

[Section 5.1.4, “Server System Variables”](#)

S

[[index top \[4483\]](#)]

SERIALIZABLE

[Section 13.3.6, “SET TRANSACTION Syntax”](#)

[Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#)

[Section 14.3.9.2, “Consistent Nonlocking Reads”](#)

[Section 8.9.3.1, “How the Query Cache Operates”](#)

[Section 17.1.6.3, “Limits Relating to Transaction Handling in MySQL Cluster”](#)

[Section 14.3.9.6, “Locks Set by Different SQL Statements in InnoDB”](#)

[Section 5.2.4.3, “Mixed Binary Logging Format”](#)

[Section 5.1.3, “Server Command Options”](#)

[Section 14.3.9, “The InnoDB Transaction Model and Locking”](#)

[Section 13.3.7, “XA Transactions”](#)

